US Patent & Trademark Office
Patent Public Search | Text View

United States Patent	12393530
Kind Code	B2
Date of Patent	August 19, 2025
Inventor(s)	Beecroft; Jonathan P. et al.

System and method for dynamic allocation of reduction engines

Abstract

A switch equipped with a reduction engine capable of being dynamically allocated in a network is provided. During operation, the reduction engine can be dynamically armed based on a multicast frame. As a result, the network can facilitate an efficient and scalable environment for high performance computing.

Inventors:	**Beecroft; Jonathan P. (Bristol, GB), Turner; Edward J. (Bristol Keynsham, GB)**
Applicant:	**Hewlett Packard Enterprise Development LP** (Houston, TX)
Family ID:	**1000008767648**
Assignee:	**Hewlett Packard Enterprise Development LP (Spring, TX)**
Appl. No.:	**17/594795**
Filed (or PCT Filed):	**March 23, 2020**
PCT No.:	**PCT/US2020/024260**
PCT Pub. No.:	**WO2020/236283**
PCT Pub. Date:	November 26, 2020

Prior Publication Data

Document Identifier	Publication Date
US 20220210055 A1	Jun. 30, 2022

Related U.S. Application Data

us-provisional-application US 62852273 20190523
us-provisional-application US 62852203 20190523

us-provisional-application US 62852289 20190523

Publication Classification

Int. Cl.: **H04L45/028** (20220101); **G06F9/50** (20060101); **G06F9/54** (20060101); **G06F12/0862** (20160101); **G06F12/1036** (20160101); **G06F12/1045** (20160101); **G06F13/14** (20060101); **G06F13/16** (20060101); **G06F13/28** (20060101); **G06F13/38** (20060101); **G06F13/40** (20060101); **G06F13/42** (20060101); **G06F15/173** (20060101); **H04L1/00** (20060101); **H04L43/0876** (20220101); **H04L43/10** (20220101); **H04L45/00** (20220101); **H04L45/02** (20220101); **H04L45/021** (20220101); **H04L45/12** (20220101); **H04L45/122** (20220101); **H04L45/125** (20220101); **H04L45/16** (20220101); **H04L45/24** (20220101); **H04L45/42** (20220101); **H04L45/745** (20220101); **H04L47/10** (20220101); **H04L47/11** (20220101); **H04L47/12** (20220101); **H04L47/122** (20220101); **H04L47/20** (20220101); **H04L47/22** (20220101); **H04L47/24** (20220101); **H04L47/2441** (20220101); **H04L47/2466** (20220101); **H04L47/2483** (20220101); **H04L47/30** (20220101); **H04L47/32** (20220101); **H04L47/34** (20220101); **H04L47/52** (20220101); **H04L47/62** (20220101); **H04L47/625** (20220101); **H04L47/6275** (20220101); **H04L47/629** (20220101); **H04L47/76** (20220101); **H04L47/762** (20220101); **H04L47/78** (20220101); **H04L47/80** (20220101); **H04L49/00** (20220101); **H04L49/101** (20220101); **H04L49/15** (20220101); **H04L49/90** (20220101); **H04L49/9005** (20220101); **H04L49/9047** (20220101); **H04L67/1097** (20220101); **H04L69/22** (20220101); **H04L69/40** (20220101); H04L45/28 (20220101); H04L45/7453 (20220101); H04L69/28 (20220101)

U.S. Cl.:

CPC **G06F13/1642** (20130101); **G06F9/505** (20130101); **G06F9/546** (20130101); **G06F12/0862** (20130101); **G06F12/1036** (20130101); **G06F12/1063** (20130101); **G06F13/14** (20130101); **G06F13/16** (20130101); **G06F13/1673** (20130101); **G06F13/28** (20130101); **G06F13/385** (20130101); **G06F13/4022** (20130101); **G06F13/4068** (20130101); **G06F13/4221** (20130101); **G06F15/17331** (20130101); **H04L1/0083** (20130101); **H04L43/0876** (20130101); **H04L43/10** (20130101); **H04L45/02** (20130101); **H04L45/021** (20130101); **H04L45/028** (20130101); **H04L45/122** (20130101); **H04L45/123** (20130101); **H04L45/125** (20130101); **H04L45/16** (20130101); **H04L45/20** (20130101); **H04L45/22** (20130101); **H04L45/24** (20130101); **H04L45/38** (20130101); **H04L45/42** (20130101); **H04L45/46** (20130101); **H04L45/566** (20130101); **H04L45/70** (20130101); **H04L45/745** (20130101); **H04L47/11** (20130101); **H04L47/12** (20130101); **H04L47/122** (20130101); **H04L47/18** (20130101); **H04L47/20** (20130101); **H04L47/22** (20130101); **H04L47/24** (20130101); **H04L47/2441** (20130101); **H04L47/2466** (20130101); **H04L47/2483** (20130101); **H04L47/30** (20130101); **H04L47/32** (20130101); **H04L47/323** (20130101); **H04L47/34** (20130101); **H04L47/39** (20130101); **H04L47/52** (20130101); **H04L47/621** (20130101); **H04L47/6235** (20130101); **H04L47/626** (20130101); **H04L47/6275** (20130101); **H04L47/629** (20130101); **H04L47/76** (20130101); **H04L47/762** (20130101); **H04L47/781** (20130101); **H04L47/80** (20130101); **H04L49/101** (20130101); **H04L49/15** (20130101); **H04L49/30** (20130101); **H04L49/3009** (20130101); **H04L49/3018** (20130101); **H04L49/3027** (20130101); **H04L49/90** (20130101); **H04L49/9005** (20130101); **H04L49/9021** (20130101); **H04L49/9036** (20130101); **H04L49/9047** (20130101); **H04L67/1097** (20130101); **H04L69/22** (20130101); **H04L69/40** (20130101); G06F13/1689 (20130101); G06F2212/50 (20130101);

G06F2213/0026 (20130101); G06F2213/3808 (20130101); H04L45/28 (20130101); H04L45/7453 (20130101); H04L69/28 (20130101)

Field of Classification Search

CPC: H04L (45/28); H04L (1/0083); H04L (43/0876); G06F (9/505); G06F (9/546); G06F (12/0862); G06F (12/1036); G06F (12/1063); G06F (13/14); G06F (13/16); G06F (13/1642); G06F (13/1673); G06F (13/1689); G06F (13/385); G06F (13/4022); G06F (13/4068); G06F (13/4221); G06F (15/17331)

References Cited

U.S. PATENT DOCUMENTS

Patent No.	Issued Date	Patentee Name	U.S. Cl.	CPC
4807118	12/1988	Lin et al.	N/A	N/A
5138615	12/1991	Lamport et al.	N/A	N/A
5457687	12/1994	Newman	N/A	N/A
5937436	12/1998	Watkins	N/A	N/A
5960178	12/1998	Cochinwala et al.	N/A	N/A
5970232	12/1998	Passint et al.	N/A	N/A
5983332	12/1998	Watkins	N/A	N/A
6112265	12/1999	Harriman et al.	N/A	N/A
6230252	12/2000	Passint et al.	N/A	N/A
6246682	12/2000	Roy et al.	N/A	N/A
6493347	12/2001	Sindhu et al.	N/A	N/A
6545981	12/2002	Garcia et al.	N/A	N/A
6633580	12/2002	Toerudbakken et al.	N/A	N/A
6674720	12/2003	Passint et al.	N/A	N/A
6714553	12/2003	Poole et al.	N/A	N/A
6728211	12/2003	Peris et al.	N/A	N/A
6732212	12/2003	Sugahara et al.	N/A	N/A
6735173	12/2003	Lenoski et al.	N/A	N/A
6894974	12/2004	Aweva et al.	N/A	N/A
7023856	12/2005	Washabaugh et al.	N/A	N/A
7133940	12/2005	Blightman et al.	N/A	N/A
7218637	12/2006	Best et al.	N/A	N/A
7269180	12/2006	Bly et al.	N/A	N/A
7305487	12/2006	Blumrich et al.	N/A	N/A
7337285	12/2007	Tanoue	N/A	N/A
7397797	12/2007	Alfieri et al.	N/A	N/A
7430559	12/2007	Lomet	N/A	N/A
7441006	12/2007	Biran et al.	N/A	N/A
7464174	12/2007	Ngai	N/A	N/A
7483442	12/2008	Torudbakken et al.	N/A	N/A
7562366	12/2008	Pope et al.	N/A	N/A
7593329	12/2008	Kwan et al.	N/A	N/A
7596628	12/2008	Aloni et al.	N/A	N/A
7620791	12/2008	Wentzlaff et al.	N/A	N/A

7633869	12/2008	Morris et al.	N/A	N/A
7639616	12/2008	Manula et al.	N/A	N/A
7734894	12/2009	Wentzlaff et al.	N/A	N/A
7774461	12/2009	Tanaka et al.	N/A	N/A
7782869	12/2009	Chitlur Srinivasa	N/A	N/A
7796579	12/2009	Bruss	N/A	N/A
7856026	12/2009	Finan et al.	N/A	N/A
7933282	12/2010	Gupta et al.	N/A	N/A
7953002	12/2010	Opsasnick	N/A	N/A
7975120	12/2010	Sabbatini, Jr. et al.	N/A	N/A
8014278	12/2010	Subramanian et al.	N/A	N/A
8023521	12/2010	Woo et al.	N/A	N/A
8050180	12/2010	Judd	N/A	N/A
8077606	12/2010	Litwack	N/A	N/A
8103788	12/2011	Miranda	N/A	N/A
8160085	12/2011	Voruganti et al.	N/A	N/A
8175107	12/2011	Yalagandula et al.	N/A	N/A
8249072	12/2011	Sugumar et al.	N/A	N/A
8281013	12/2011	Mundkur et al.	N/A	N/A
8352727	12/2012	Chen et al.	N/A	N/A
8353003	12/2012	Noehring et al.	N/A	N/A
8443151	12/2012	Tang et al.	N/A	N/A
8473783	12/2012	Andrade et al.	N/A	N/A
8543534	12/2012	Alves et al.	N/A	N/A
8619793	12/2012	Lavian et al.	N/A	N/A
8626957	12/2013	Blumrich et al.	N/A	N/A
8650582	12/2013	Archer et al.	N/A	N/A
8706832	12/2013	Blocksome	N/A	N/A
8719543	12/2013	Kaminski et al.	N/A	N/A
8811183	12/2013	Anand et al.	N/A	N/A
8948175	12/2014	Bly et al.	N/A	N/A
8971345	12/2014	McCanne et al.	N/A	N/A
9001663	12/2014	Attar et al.	N/A	N/A
9053012	12/2014	Northcott et al.	N/A	N/A
9088496	12/2014	Vaidya et al.	N/A	N/A
9094327	12/2014	Jacobs et al.	N/A	N/A
9178782	12/2014	Matthews et al.	N/A	N/A
9208071	12/2014	Talagala et al.	N/A	N/A
9218278	12/2014	Talagala et al.	N/A	N/A
9231876	12/2015	Mir et al.	N/A	N/A
9231888	12/2015	Bogdanski et al.	N/A	N/A
9239804	12/2015	Kegel et al.	N/A	N/A
9269438	12/2015	Nachimuthu et al.	N/A	N/A
9276864	12/2015	Pradeep	N/A	N/A
9436651	12/2015	Underwood et al.	N/A	N/A
9455915	12/2015	Sinha et al.	N/A	N/A
9460178	12/2015	Bashyam et al.	N/A	N/A
9479426	12/2015	Munger et al.	N/A	N/A
9496991	12/2015	Plamondon et al.	N/A	N/A
9544234	12/2016	Markine	N/A	N/A

9548924	12/2016	Pettit et al.	N/A	N/A
9594521	12/2016	Blagodurov et al.	N/A	N/A
9635121	12/2016	Mathew et al.	N/A	N/A
9742855	12/2016	Shuler et al.	N/A	N/A
9762488	12/2016	Previdi et al.	N/A	N/A
9762497	12/2016	Kishore et al.	N/A	N/A
9830273	12/2016	Bk et al.	N/A	N/A
9838500	12/2016	Ilan et al.	N/A	N/A
9853900	12/2016	Mula et al.	N/A	N/A
9887923	12/2017	Chorafakis et al.	N/A	N/A
10003544	12/2017	Liu et al.	N/A	N/A
10009270	12/2017	Stark et al.	N/A	N/A
10031857	12/2017	Menachem et al.	N/A	N/A
10050896	12/2017	Yang et al.	N/A	N/A
10061613	12/2017	Brooker et al.	N/A	N/A
10063481	12/2017	Jiang et al.	N/A	N/A
10089220	12/2017	McKelvie et al.	N/A	N/A
10169060	12/2018	Vincent et al.	N/A	N/A
10178035	12/2018	Dillon	N/A	N/A
10200279	12/2018	Aljaedi	N/A	N/A
10218634	12/2018	Aldebert et al.	N/A	N/A
10270700	12/2018	Burnette et al.	N/A	N/A
10305772	12/2018	Zur et al.	N/A	N/A
10331590	12/2018	MacNamara et al.	N/A	N/A
10353833	12/2018	Hagspiel et al.	N/A	N/A
10454835	12/2018	Contavalli et al.	N/A	N/A
10498672	12/2018	Graham et al.	N/A	N/A
10567307	12/2019	Fairhurst et al.	N/A	N/A
10728173	12/2019	Agrawal et al.	N/A	N/A
10802828	12/2019	Volpe et al.	N/A	N/A
10817502	12/2019	Talagala et al.	N/A	N/A
11128561	12/2020	Matthews et al.	N/A	N/A
11271869	12/2021	Agrawal et al.	N/A	N/A
11416749	12/2021	Bshara et al.	N/A	N/A
11444886	12/2021	Stawitzky et al.	N/A	N/A
2001/0010692	12/2000	Sindhu et al.	N/A	N/A
2001/0047438	12/2000	Forin	N/A	N/A
2002/0174279	12/2001	Wynne et al.	N/A	N/A
2003/0016808	12/2002	Hu et al.	N/A	N/A
2003/0041168	12/2002	Musoll	N/A	N/A
2003/0110455	12/2002	Baumgartner et al.	N/A	N/A
2003/0174711	12/2002	Shankar	N/A	N/A
2003/0200363	12/2002	Futral	N/A	N/A
2003/0223420	12/2002	Ferolito	N/A	N/A
2004/0008716	12/2003	Stiliadis	N/A	N/A
2004/0059828	12/2003	Hooper et al.	N/A	N/A
2004/0095882	12/2003	Hamzah et al.	N/A	N/A
2004/0133634	12/2003	Luke et al.	N/A	N/A
2004/0223452	12/2003	Santos et al.	N/A	N/A
2005/0021837	12/2004	Haselhorst et al.	N/A	N/A

2005/0047334	12/2004	Paul et al.	N/A	N/A
2005/0088969	12/2004	Carlsen et al.	N/A	N/A
2005/0091396	12/2004	Nilakantan et al.	N/A	N/A
2005/0108444	12/2004	Flauaus et al.	N/A	N/A
2005/0108518	12/2004	Pandya	N/A	N/A
2005/0152274	12/2004	Simpson	N/A	N/A
2005/0182854	12/2004	Pinkerton et al.	N/A	N/A
2005/0270974	12/2004	Mayhew	N/A	N/A
2005/0270976	12/2004	Yang et al.	N/A	N/A
2006/0023705	12/2005	Zoranovic et al.	N/A	N/A
2006/0067347	12/2005	Naik et al.	N/A	N/A
2006/0075480	12/2005	Noehring et al.	N/A	N/A
2006/0174251	12/2005	Pope et al.	N/A	N/A
2006/0203728	12/2005	Kwan et al.	N/A	N/A
2007/0061433	12/2006	Reynolds et al.	N/A	N/A
2007/0070901	12/2006	Aloni et al.	N/A	N/A
2007/0198804	12/2006	Moyer	N/A	N/A
2007/0211746	12/2006	Oshikiri et al.	N/A	N/A
2007/0242611	12/2006	Archer et al.	N/A	N/A
2007/0268825	12/2006	Corwin et al.	N/A	N/A
2008/0013453	12/2007	Chiang et al.	N/A	N/A
2008/0013549	12/2007	Okagawa et al.	N/A	N/A
2008/0071757	12/2007	Ichiriu et al.	N/A	N/A
2008/0084864	12/2007	Archer et al.	N/A	N/A
2008/0091915	12/2007	Moertl et al.	N/A	N/A
2008/0147881	12/2007	Krishnamurthy	709/238	H04L 45/02
2008/0159138	12/2007	Shepherd et al.	N/A	N/A
2008/0253289	12/2007	Naven et al.	N/A	N/A
2009/0003212	12/2008	Kwan et al.	N/A	N/A
2009/0010157	12/2008	Holmes et al.	N/A	N/A
2009/0013175	12/2008	Elliott	N/A	N/A
2009/0055496	12/2008	Garg et al.	N/A	N/A
2009/0064140	12/2008	Arimilli	718/100	G06F 15/17356
2009/0092046	12/2008	Naven et al.	N/A	N/A
2009/0141621	12/2008	Fan et al.	N/A	N/A
2009/0198958	12/2008	Arimilli et al.	N/A	N/A
2009/0259713	12/2008	Blumrich et al.	N/A	N/A
2009/0285222	12/2008	Hoover et al.	N/A	N/A
2010/0061241	12/2009	Sindhu et al.	N/A	N/A
2010/0169608	12/2009	Kuo et al.	N/A	N/A
2010/0172260	12/2009	Kwan et al.	N/A	N/A
2010/0183024	12/2009	Gupta	N/A	N/A
2010/0220595	12/2009	Petersen	N/A	N/A
2010/0274876	12/2009	Kagan et al.	N/A	N/A
2010/0302942	12/2009	Shankar et al.	N/A	N/A
2010/0316053	12/2009	Miyoshi et al.	N/A	N/A
2011/0051724	12/2010	Scott et al.	N/A	N/A
2011/0066824	12/2010	Bestler	N/A	N/A
2011/0072179	12/2010	Lacroute et al.	N/A	N/A

2011/0099326	12/2010	Jung et al.	N/A	N/A
2011/0110383	12/2010	Yang et al.	N/A	N/A
2011/0128959	12/2010	Bando et al.	N/A	N/A
2011/0158096	12/2010	Leung et al.	N/A	N/A
2011/0158248	12/2010	Vorunganti et al.	N/A	N/A
2011/0164496	12/2010	Loh et al.	N/A	N/A
2011/0173370	12/2010	Jacobs et al.	N/A	N/A
2011/0264822	12/2010	Ferguson et al.	N/A	N/A
2011/0276699	12/2010	Pedersen	N/A	N/A
2011/0280125	12/2010	Jayakumar	N/A	N/A
2011/0320724	12/2010	Mejdrich et al.	N/A	N/A
2012/0093505	12/2011	Yeap et al.	N/A	N/A
2012/0102506	12/2011	Hopmann et al.	N/A	N/A
2012/0117423	12/2011	Andrade et al.	N/A	N/A
2012/0137075	12/2011	Vorbach	N/A	N/A
2012/0144064	12/2011	Parker et al.	N/A	N/A
2012/0144065	12/2011	Parker et al.	N/A	N/A
2012/0147752	12/2011	Ashwood-Smith et al.	N/A	N/A
2012/0170462	12/2011	Sinha	N/A	N/A
2012/0170575	12/2011	Mehra	N/A	N/A
2012/0213118	12/2011	Lindsay et al.	N/A	N/A
2012/0236858	12/2011	Armstrong	370/390	H04L 12/4625
2012/0250512	12/2011	Jagadeeswaran et al.	N/A	N/A
2012/0287821	12/2011	Godfrey et al.	N/A	N/A
2012/0297083	12/2011	Ferguson et al.	N/A	N/A
2012/0300669	12/2011	Zahavi	N/A	N/A
2012/0314707	12/2011	Epps et al.	N/A	N/A
2013/0010636	12/2012	Regula	N/A	N/A
2013/0039169	12/2012	Schlansker et al.	N/A	N/A
2013/0060944	12/2012	Archer et al.	N/A	N/A
2013/0103777	12/2012	Kagan et al.	N/A	N/A
2013/0121178	12/2012	Mainaud et al.	N/A	N/A
2013/0136090	12/2012	Liu et al.	N/A	N/A
2013/0182704	12/2012	Jacobs et al.	N/A	N/A
2013/0194927	12/2012	Yamaguchi et al.	N/A	N/A
2013/0203422	12/2012	Masputra et al.	N/A	N/A
2013/0205002	12/2012	Wang et al.	N/A	N/A
2013/0208593	12/2012	Nandagopal	N/A	N/A
2013/0246552	12/2012	Underwood et al.	N/A	N/A
2013/0290673	12/2012	Archer	712/30	G06F 15/17318
2013/0301645	12/2012	Bogdanski et al.	N/A	N/A
2013/0304988	12/2012	Totolos et al.	N/A	N/A
2013/0311525	12/2012	Neerincx et al.	N/A	N/A
2013/0329577	12/2012	Suzuki et al.	N/A	N/A
2013/0336164	12/2012	Yang et al.	N/A	N/A
2014/0003427	12/2013	Nishi	370/390	H04L 12/18

2014/0019661	12/2013	Hormuth et al.	N/A	N/A
2014/0032695	12/2013	Michels et al.	N/A	N/A
2014/0036680	12/2013	Lih et al.	N/A	N/A
2014/0064082	12/2013	Yeung et al.	N/A	N/A
2014/0095753	12/2013	Crupnicoff et al.	N/A	N/A
2014/0098675	12/2013	Frost et al.	N/A	N/A
2014/0119367	12/2013	Han et al.	N/A	N/A
2014/0122560	12/2013	Ramey et al.	N/A	N/A
2014/0129664	12/2013	McDaniel et al.	N/A	N/A
2014/0133292	12/2013	Yamatsu et al.	N/A	N/A
2014/0136646	12/2013	Tamir et al.	N/A	N/A
2014/0169173	12/2013	Naouri et al.	N/A	N/A
2014/0185621	12/2013	Decusatis et al.	N/A	N/A
2014/0189174	12/2013	Ajanovic et al.	N/A	N/A
2014/0207881	12/2013	Nussle et al.	N/A	N/A
2014/0211804	12/2013	Makikeni et al.	N/A	N/A
2014/0226488	12/2013	Shamis et al.	N/A	N/A
2014/0241164	12/2013	Cociglio et al.	N/A	N/A
2014/0258438	12/2013	Ayoub	N/A	N/A
2014/0301390	12/2013	Scott et al.	N/A	N/A
2014/0307554	12/2013	Basso et al.	N/A	N/A
2014/0325013	12/2013	Tamir et al.	N/A	N/A
2014/0328172	12/2013	Kumar et al.	N/A	N/A
2014/0347997	12/2013	Bergamasco et al.	N/A	N/A
2014/0362698	12/2013	Arad	N/A	N/A
2014/0369360	12/2013	Carlstrom	N/A	N/A
2014/0379847	12/2013	Williams	N/A	N/A
2015/0003247	12/2014	Mejia et al.	N/A	N/A
2015/0006849	12/2014	Xu et al.	N/A	N/A
2015/0009823	12/2014	Ganga et al.	N/A	N/A
2015/0026361	12/2014	Matthews et al.	N/A	N/A
2015/0029848	12/2014	Jain	N/A	N/A
2015/0055476	12/2014	Decusatis et al.	N/A	N/A
2015/0055661	12/2014	Boucher et al.	N/A	N/A
2015/0067095	12/2014	Gopal et al.	N/A	N/A
2015/0089495	12/2014	Persson et al.	N/A	N/A
2015/0103667	12/2014	Elias et al.	N/A	N/A
2015/0124826	12/2014	Edsall et al.	N/A	N/A
2015/0146527	12/2014	Kishore et al.	N/A	N/A
2015/0154004	12/2014	Aggarwal	N/A	N/A
2015/0161064	12/2014	Pope	N/A	N/A
2015/0180782	12/2014	Rimmer et al.	N/A	N/A
2015/0186318	12/2014	Kim et al.	N/A	N/A
2015/0193262	12/2014	Archer et al.	N/A	N/A
2015/0195388	12/2014	Snyder et al.	N/A	N/A
2015/0208145	12/2014	Parker et al.	N/A	N/A
2015/0220449	12/2014	Stark et al.	N/A	N/A
2015/0237180	12/2014	Swartzentruber et al.	N/A	N/A
2015/0244617	12/2014	Nakil et al.	N/A	N/A

2015/0244804	12/2014	Warfield et al.	N/A	N/A
2015/0261434	12/2014	Kagan et al.	N/A	N/A
2015/0263955	12/2014	Talaski et al.	N/A	N/A
2015/0263994	12/2014	Haramaty et al.	N/A	N/A
2015/0288626	12/2014	Aybay	N/A	N/A
2015/0365337	12/2014	Pannell	N/A	N/A
2015/0370586	12/2014	Cooper et al.	N/A	N/A
2016/0006664	12/2015	Sabato et al.	N/A	N/A
2016/0012002	12/2015	Arimilli et al.	N/A	N/A
2016/0028613	12/2015	Haramaty et al.	N/A	N/A
2016/0065455	12/2015	Wang et al.	N/A	N/A
2016/0094450	12/2015	Ghanwani et al.	N/A	N/A
2016/0134518	12/2015	Callon et al.	N/A	N/A
2016/0134535	12/2015	Callon	N/A	N/A
2016/0134559	12/2015	Abel et al.	N/A	N/A
2016/0134573	12/2015	Gagliardi et al.	N/A	N/A
2016/0142318	12/2015	Beecroft	N/A	N/A
2016/0154756	12/2015	Dodson et al.	N/A	N/A
2016/0182383	12/2015	Pedersen	N/A	N/A
2016/0205023	12/2015	Janardhanan	N/A	N/A
2016/0226797	12/2015	Aravinthan et al.	N/A	N/A
2016/0254991	12/2015	Eckert et al.	N/A	N/A
2016/0259394	12/2015	Ragavan	N/A	N/A
2016/0283422	12/2015	Crupnicoff et al.	N/A	N/A
2016/0285545	12/2015	Schmidtke et al.	N/A	N/A
2016/0285677	12/2015	Kashyap et al.	N/A	N/A
2016/0294694	12/2015	Parker et al.	N/A	N/A
2016/0294926	12/2015	Zur et al.	N/A	N/A
2016/0301610	12/2015	Amit et al.	N/A	N/A
2016/0344620	12/2015	Santos et al.	N/A	N/A
2016/0381189	12/2015	Caulfield et al.	N/A	N/A
2017/0024263	12/2016	Verplanken	N/A	N/A
2017/0039063	12/2016	Gopal et al.	N/A	N/A
2017/0041239	12/2016	Goldenberg et al.	N/A	N/A
2017/0048144	12/2016	Liu	N/A	N/A
2017/0054633	12/2016	Underwood et al.	N/A	N/A
2017/0091108	12/2016	Arellano et al.	N/A	N/A
2017/0097840	12/2016	Bridgers	N/A	N/A
2017/0103108	12/2016	Datta et al.	N/A	N/A
2017/0118090	12/2016	Pettit et al.	N/A	N/A
2017/0118098	12/2016	Littlejohn et al.	N/A	N/A
2017/0153852	12/2016	Ma et al.	N/A	N/A
2017/0177541	12/2016	Berman et al.	N/A	N/A
2017/0220500	12/2016	Tong	N/A	N/A
2017/0237654	12/2016	Turner et al.	N/A	N/A
2017/0237671	12/2016	Rimmer et al.	N/A	N/A
2017/0242753	12/2016	Sherlock et al.	N/A	N/A
2017/0250914	12/2016	Caulfield et al.	N/A	N/A
2017/0251394	12/2016	Johansson et al.	N/A	N/A
2017/0270051	12/2016	Chen et al.	N/A	N/A

2017/0272331	12/2016	Lissack	N/A	N/A
2017/0272370	12/2016	Ganga et al.	N/A	N/A
2017/0286316	12/2016	Doshi et al.	N/A	N/A
2017/0289066	12/2016	Haramaty et al.	N/A	N/A
2017/0295098	12/2016	Watkins et al.	N/A	N/A
2017/0324664	12/2016	Xu et al.	N/A	N/A
2017/0371778	12/2016	McKelvie et al.	N/A	N/A
2018/0004705	12/2017	Menachem et al.	N/A	N/A
2018/0019948	12/2017	Patwardhan et al.	N/A	N/A
2018/0026878	12/2017	Zahavi et al.	N/A	N/A
2018/0077064	12/2017	Wang	N/A	N/A
2018/0083868	12/2017	Cheng	N/A	N/A
2018/0097645	12/2017	Rajagopalan et al.	N/A	N/A
2018/0097912	12/2017	Chumbalkar et al.	N/A	N/A
2018/0113618	12/2017	Chan et al.	N/A	N/A
2018/0115469	12/2017	Erickson et al.	N/A	N/A
2018/0131602	12/2017	Civanlar et al.	N/A	N/A
2018/0131678	12/2017	Agarwal et al.	N/A	N/A
2018/0150374	12/2017	Ratcliff	N/A	N/A
2018/0152317	12/2017	Chang et al.	N/A	N/A
2018/0152357	12/2017	Natham et al.	N/A	N/A
2018/0173557	12/2017	Nakil et al.	N/A	N/A
2018/0183724	12/2017	Callard et al.	N/A	N/A
2018/0191609	12/2017	Caulfield et al.	N/A	N/A
2018/0198736	12/2017	Labonte et al.	N/A	N/A
2018/0212876	12/2017	Bacthu et al.	N/A	N/A
2018/0212902	12/2017	Steinmacher-Burow	N/A	N/A
2018/0219804	12/2017	Graham et al.	N/A	N/A
2018/0225238	12/2017	Karguth et al.	N/A	N/A
2018/0234343	12/2017	Zdornov et al.	N/A	N/A
2018/0254945	12/2017	Bogdanski et al.	N/A	N/A
2018/0260324	12/2017	Marathe et al.	N/A	N/A
2018/0278540	12/2017	Shalev et al.	N/A	N/A
2018/0287928	12/2017	Levi et al.	N/A	N/A
2018/0323898	12/2017	Dods	N/A	N/A
2018/0335974	12/2017	Simionescu et al.	N/A	N/A
2018/0341494	12/2017	Sood et al.	N/A	N/A
2019/0007349	12/2018	Wang et al.	N/A	N/A
2019/0018808	12/2018	Beard et al.	N/A	N/A
2019/0036771	12/2018	Sharpless et al.	N/A	N/A
2019/0042337	12/2018	Dinan et al.	N/A	N/A
2019/0042518	12/2018	Marolia	N/A	N/A
2019/0044809	12/2018	Willis et al.	N/A	N/A
2019/0044827	12/2018	Ganapathi et al.	N/A	N/A
2019/0044863	12/2018	Mula et al.	N/A	N/A
2019/0044872	12/2018	Ganapathi et al.	N/A	N/A
2019/0044875	12/2018	Murty et al.	N/A	N/A
2019/0052327	12/2018	Motozuka et al.	N/A	N/A
2019/0058663	12/2018	Song	N/A	N/A

2019/0068501	12/2018	Schneider et al.	N/A	N/A
2019/0081903	12/2018	Kobayashi et al.	N/A	N/A
2019/0095134	12/2018	Li	N/A	N/A
2019/0104057	12/2018	Goel et al.	N/A	N/A
2019/0104206	12/2018	Goel et al.	N/A	N/A
2019/0108106	12/2018	Aggarwal et al.	N/A	N/A
2019/0108332	12/2018	Glew et al.	N/A	N/A
2019/0109791	12/2018	Mehra et al.	N/A	N/A
2019/0121781	12/2018	Kasichainula	N/A	N/A
2019/0140979	12/2018	Levi et al.	N/A	N/A
2019/0146477	12/2018	Cella et al.	N/A	N/A
2019/0171612	12/2018	Shahar et al.	N/A	N/A
2019/0196982	12/2018	Rozas et al.	N/A	N/A
2019/0199646	12/2018	Singh et al.	N/A	N/A
2019/0253354	12/2018	Caulfield et al.	N/A	N/A
2019/0280978	12/2018	Schmatz et al.	N/A	N/A
2019/0294575	12/2018	Dennison et al.	N/A	N/A
2019/0306134	12/2018	Shanbhogue et al.	N/A	N/A
2019/0332314	12/2018	Zhang et al.	N/A	N/A
2019/0334624	12/2018	Bernard	N/A	N/A
2019/0356611	12/2018	Das et al.	N/A	N/A
2019/0361728	12/2018	Kumar et al.	N/A	N/A
2019/0379610	12/2018	Srinivasan et al.	N/A	N/A
2020/0036644	12/2019	Belogolovy et al.	N/A	N/A
2020/0084150	12/2019	Burstein et al.	N/A	N/A
2020/0145725	12/2019	Eberle et al.	N/A	N/A
2020/0177505	12/2019	Li	N/A	N/A
2020/0177521	12/2019	Blumrich et al.	N/A	N/A
2020/0259755	12/2019	Wang et al.	N/A	N/A
2020/0272579	12/2019	Humphrey et al.	N/A	N/A
2020/0274832	12/2019	Humphrey et al.	N/A	N/A
2020/0334195	12/2019	Chen et al.	N/A	N/A
2020/0349098	12/2019	Caulfield et al.	N/A	N/A
2021/0081410	12/2020	Chavan et al.	N/A	N/A
2021/0152494	12/2020	Johnsen et al.	N/A	N/A
2021/0263779	12/2020	Haghighat et al.	N/A	N/A
2021/0334206	12/2020	Colgrove et al.	N/A	N/A
2021/0377156	12/2020	Michael et al.	N/A	N/A
2021/0409351	12/2020	Das et al.	N/A	N/A
2022/0131768	12/2021	Ganapathi et al.	N/A	N/A
2022/0166705	12/2021	Froese	N/A	N/A
2022/0200900	12/2021	Roweth	N/A	N/A
2022/0210058	12/2021	Bataineh et al.	N/A	N/A
2022/0217078	12/2021	Ford et al.	N/A	N/A
2022/0217101	12/2021	Yefet et al.	N/A	N/A
2022/0245072	12/2021	Roweth et al.	N/A	N/A
2022/0278941	12/2021	Shalev et al.	N/A	N/A
2022/0309025	12/2021	Chen et al.	N/A	N/A
2023/0035420	12/2022	Sankaran et al.	N/A	N/A
2023/0046221	12/2022	Pismenny et al.	N/A	N/A

FOREIGN PATENT DOCUMENTS

Patent No.	Application Date	Country	CPC
101729609	12/2009	CN	N/A
102932203	12/2012	CN	N/A
110324249	12/2018	CN	N/A
110601888	12/2018	CN	N/A
0275135	12/1987	EP	N/A
2187576	12/2009	EP	N/A
2219329	12/2009	EP	N/A
2947832	12/2014	EP	N/A
3445006	12/2018	EP	N/A
2003-244196	12/2002	JP	N/A
3459653	12/2002	JP	N/A
10-2012-0062864	12/2011	KR	N/A
10-2012-0082739	12/2011	KR	N/A
10-2014-0100529	12/2013	KR	N/A
10-2015-0026939	12/2014	KR	N/A
10-2015-0104056	12/2014	KR	N/A
10-2017-0110106	12/2016	KR	N/A
10-1850749	12/2017	KR	N/A
2001/069851	12/2000	WO	N/A
02/47329	12/2001	WO	N/A
2003/019861	12/2002	WO	N/A
2004/001615	12/2002	WO	N/A
2005/094487	12/2004	WO	N/A
2007/034184	12/2006	WO	N/A
2009/010461	12/2008	WO	N/A
2009/018232	12/2008	WO	N/A
2014/092780	12/2013	WO	N/A
2014/137382	12/2013	WO	N/A
2014/141005	12/2013	WO	N/A
2018/004977	12/2017	WO	N/A
2018/046703	12/2017	WO	N/A
2019/072072	12/2018	WO	N/A

OTHER PUBLICATIONS

Chang, F., et al.; “PVW: Designing Virtual World Server Infrastructure”; 2010; 8 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US20/24260, mailed on Jul. 7, 2020, 11 pages. cited by applicant
Mamidala, A.R., et al.; “Efficient Barrier and Allreduce on Infiniband clusters using multicast and adaptive algorithms”; Sep. 20-23, 2004; 10 pages. cited by applicant
Roth, P. C., et al.; “MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools1”; Nov. 15-21, 2003; 16 pages. cited by applicant
International Search Report and Written Opinion received for PCT Patent Application No. PCT/US20/24342, mailed on Oct. 27, 2020, 10 pages. cited by applicant
International Search Report and Written Opinion received for PCT Patent Application No. PCT/US2020/024192, mailed on Oct. 23, 2020, 9 pages. cited by applicant

International Search Report and Written Opinion received for PCT Patent Application No. PCT/US2020/024221, mailed on Oct. 26, 2020, 9 pages. cited by applicant
International Search Report cited in PCT/US2020/024170 mailed Dec. 16, 2020; 3 pages. cited by applicant
Maabi, S., et al.; “ERFAN: Efficient reconfigurable fault-tolerant deflection routing algorithm for 3-D Network-on-Chip”; Sep. 6-9, 2016. cited by applicant
Maglione-Mathey, G., et al.; “Scalable Deadlock-Free Deterministic Minimal-Path Routing Engine for InfiniBand-Based Dragonfly Networks”; Aug. 21, 2017; 15 pages. cited by applicant
Mammeri, Z; “Reinforcement Learning Based Routing in Networks: Review and Classification of Approaches”; Apr. 29, 2019; 35 pages. cited by applicant
Mollah; M. A., et al.; “High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation: 8th International Workshop”; Nov. 13, 2017. cited by applicant
Open Networking Foundation; “OpenFlow Switch Specification”; Mar. 26, 2015; 283 pages. cited by applicant
Prakash, P., et al.; “The TCP Outcast Problem: Exposing Unfairness in Data Center Networks”; 2011; 15 pages. cited by applicant
Ramakrishnan, K., et al.; “The Addition of Explicit Congestion Notification (ECN) to IP”; Sep. 2001; 63 pages. cited by applicant
Silveira, J., et al.; “Preprocessing of Scenarios for Fast and Efficient Routing Reconfiguration in Fault-Tolerant NoCs”; Mar. 4-6, 2015. cited by applicant
Tsunekawa, K.; “Fair bandwidth allocation among LSPs for AF class accommodating TCP and UDP traffic in a Diffserv-capable MPLS network”; Nov. 17, 2005; 9 pages. cited by applicant
Underwood, K.D., et al.; “A hardware acceleration unit for MPI queue processing”; Apr. 18, 2005; 10 pages. cited by applicant
Wu, J.; “Fault-tolerant adaptive and minimal routing in mesh-connected multicomputers using extended safety levels”; Feb. 2000; 11 pages. cited by applicant
Xiang, D., et al.; “Fault-Tolerant Adaptive Routing in Dragonfly Networks”; Apr. 12, 2017; 15 pages. cited by applicant
Xiang, D., et al.; “Deadlock-Free Broadcast Routing in Dragonfly Networks without Virtual Channels”, submission to IEEE transactions on Parallel and Distributed Systems, 2015, 15 pages. cited by applicant
Ramakrishnan et al., RFC 3168, “The addition of Explicit Congestion Notification (ECN) to IP”, Sep. 2001 (Year: 2001). cited by applicant
Extended European Search Report and Search Opinion received for EP Application No. 20809930.9, mailed on Mar. 2, 2023, 9 pages. cited by applicant
Extended European Search Report and Search Opinion received for EP Application No. 20810784.7, mailed on Mar. 9, 2023, 7 pages. cited by applicant
Awerbuch, B., et al.; “An On-Demand Secure Routing Protocol Resilient to Byzantine Failures”; Sep. 2002; 10 pages. cited by applicant
Belayneh L.W., et al.; “Method and Apparatus for Routing Data in an Inter-Nodal Communications Lattice of a Massively Parallel Computer System by Semi-Randomly Varying Routing Policies for Different Packets”; 2019; 3 pages. cited by applicant
Bhatele, A., et al.; “Analyzing Network Health and Congestion in Dragonfly-based Supercomputers”; May 23-27, 2016; 10 pages. cited by applicant
Blumrich, M.A., et al.; “Exploiting Idle Resources in a High-Radix Switch for Supplemental Storage”; Nov. 2018; 13 pages. cited by applicant
Chang, F., et al.; “PVW: Designing Vir PVW: Designing Virtual World Ser orld Server Infr er Infrastructur astructure”; 2010; 8 pages. cited by applicant
Chen, F., et al.; “Requirements for RoCEv3 Congestion Management”; Mar. 21, 2019; 8 pages. cited by applicant

Cisco Packet Tracer; “packet-tracer;—ping”; https://www.cisco.com/c/en/us/td/docs/security/asa/asa-command-reference/I-R/cmdref2/p1.html; 2017. cited by applicant
Cisco; “Understanding Rapid Spanning Tree Protocol (802.1w)”; Aug. 1, 2017; 13 pages. cited by applicant
Eardley, Ed, P; “Pre-Congestion Notification (PCN) Architecture”; Jun. 2009; 54 pages. cited by applicant
Escudero-Sahuquillo, J., et al.; “Combining Congested-Flow Isolation and Injection Throttling in HPC Interconnection Networks”; Sep. 13-16, 2011; 3 pages. cited by applicant
Hong, Y.; “Mitigating the Cost, Performance, and Power Overheads Induced by Load Variations in Multicore Cloud Servers”; Fall 2013; 132 pages. cited by applicant
Huawei; “The Lossless Network for Data Centers”; Nov. 7, 2017; 15 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024248, mailed on Jul. 8, 2020, 11 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US20/024332, mailed on Jul. 8, 2020, 13 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US20/24243, mailed on July 9. 2020, 10 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US20/24253, mailed on July 6. 2020, 12 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US20/24256, mailed on July 7. 2020, 11 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US20/24257, mailed on Jul. 7, 2020, 10 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US20/24258, mailed on July 7. 2020, 9 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US20/24259, mailed on Jul. 9, 2020, 13 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US20/24268, mailed on July 9. 2020, 11 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US20/24269, mailed on July 9. 2020, 11 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US20/24339, mailed on Jul. 8, 2020, 11 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024125, mailed on Jul. 10, 2020, 5 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024129, mailed on Jul. 10, 2020, 11 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024237, mailed on Jul. 14, 2020, 5 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024239, mailed on Jul. 14, 2020, 11 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024241, mailed on Jul. 14, 2020, 13 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024242, mailed on Jul. 6, 2020, 11 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024244, mailed on Jul. 13, 2020, 10 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024245, mailed on Jul. 14, 2020, 11 pages. cited by applicant

International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024246, mailed on Jul. 14, 2020, 10 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024250, mailed on Jul. 14, 2020, 12 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024254, mailed on Jul. 13, 2020, 10 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024262, mailed on Jul. 13, 2020, 10 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024266, mailed on Jul. 9, 2020, 10 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024270, mailed on Jul. 10, 2020, 13 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024271, mailed on Jul. 9, 2020, 10 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024272, mailed on Jul. 9, 2020, 10 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024276, mailed on Jul. 13, 2020, 9 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024304, mailed on Jul. 15, 2020, 11 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024311, mailed on Jul. 17, 2020, 8 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024321, mailed on Jul. 9, 2020, 9 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024324, mailed on Jul. 14, 2020, 10 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/024327, mailed on Jul. 10, 2020, 15 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/24158, mailed on Jul. 6, 2020, 18 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/24251, mailed on Jul. 6, 2020, 11 pages. cited by applicant
International Search Report and Written Opinion received for PCT Application No. PCT/US2020/24267, mailed on Jul. 6, 2020, 9 pages. cited by applicant
International Search Report and Written Opinion received for PCT Patent Application No. PCT/US20/24303, mailed on Oct. 21, 2020, 9 pages. cited by applicant
International Search Report and Written Opinion received for PCT Patent Application No. PCT/US20/24340, mailed on Oct. 26, 2020, 9 pages. cited by applicant

---

Primary Examiner: Mattis; Jason E

Assistant Examiner: Ma; Robert

Attorney, Agent or Firm: Dickinson Wright

Background/Summary

CROSS REFERENCE TO RELATED APPLICATIONS (1) This application is a 371 National Stage Entry of PCT/US2020/024260, filed on Mar. 23, 2020, which claims the benefit of and priority to U.S. Provisional Patent Application No. 62/852,203, filed on May 23, 2019; U.S. Provisional Patent Application No. 62/852,273, filed on May 23, 2019; and U.S. Provisional Patent Application No. 62/852,289, filed on May 23, 2019; the contents of which are incorporated herein by reference in their entirety.

BACKGROUND

Field

(1) This is generally related to the technical field of networking. More specifically, this disclosure is related to systems and methods for facilitating dynamic allocation of reduction engines in a network.

Related Art

(2) As network-enabled devices and applications become progressively more ubiquitous, various types of traffic as well as the ever-increasing network load continue to demand more performance from the underlying network architecture. For example, applications such as high-performance computing (HPC), media streaming, and Internet of Things (IOT) can generate different types of traffic with distinctive characteristics. As a result, in addition to conventional network performance metrics such as bandwidth and delay, network architects continue to face challenges such as scalability, versatility, and efficiency.

SUMMARY

(3) A switch equipped with a reduction engine capable of being dynamically allocated in a network is provided. During operation, the reduction engine can be dynamically armed based on a multicast frame. As a result, the network can facilitate an efficient and scalable environment for high performance computing.

Description

BRIEF DESCRIPTION OF THE FIGURES

(1) FIG. **1** shows an exemplary network.
(2) FIG. **2** shows an exemplary multicast tree for a reduction process.
(3) FIG. **3**A shows a flow chart of an exemplary reduction process.
(4) FIG. **3**B shows a flow chart of an exemplary reduction operation by a reduction engine.
(5) FIG. **4** shows an example where one leaf endpoint is late joining the reduction process.
(6) FIG. **5**A shows an example where one leaf endpoint fails to supply a contribution because of an error.
(7) FIG. **5**B shows a flow chart of an exemplary timer-based reduction process.
(8) FIG. **6** shows an example where a reduction engine on a leaf switch is unavailable.
(9) FIG. **7** shows exemplary reduction operations.
(10) FIG. **8** shows a set of MINMAXLOC operands that can be used in a reduction process.
(11) FIG. **9** shows rounding modes that can be used in a reduction process.
(12) FIG. **10** shows a Portals-formatted reduction frame.
(13) FIG. **11** shows a reduction header.
(14) FIG. **12** shows the endianness of operands that can be used for MINMAXLOC reproducible sum operators in a reduction process.
(15) FIG. **13** shows exemplary reduction result codes
(16) FIG. **14** shows an example where a Portals packet can be prepended with an Ethernet header.
(17) FIG. **15** shows an exemplary switching system that facilitates a reduction engine.
(18) In the figures, like reference numerals refer to the same figure elements.

DETAILED DESCRIPTION

(19) Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other embodiments and

applications without departing from the spirit and scope of the present disclosure. Thus, the present invention is not limited to the embodiments shown.

(20) Embodiments of the present invention solve the problem of accommodating a large number of computing endpoints in a network by providing a reduction engine that can be dynamically configured, which allows traffic resulting from large-scale computing to be reduced in a timely, flexible, and scalable manner. Allocation and management of any shared resource within the body of a network can be difficult, especially if errors occur while the unit is processing data. The systems and methods described herein can significantly simplify the allocation, deallocation, and error handling of a dynamically allocated switch/router resource.

(21) In general, a network can support thousands of users. If a function is provided, and expected to be used by any user, it is important to manage the resource efficiently. One approach can be to use a system call to a network-based server that has been authorized to manage the function. Although this may appear to be a simple solution, in practice the management could become complicated, especially if the function provided is widely distributed throughout the network and thousands of users may be trying to gain access. The time it takes to setup an operation for a single use can run into many seconds, and a similar amount of time may be required to release the function after use. Any error condition may also require significant support in both software development and real time analysis during the error condition.

(22) The type of function being used may only be active for a few microseconds or even a few nanoseconds. Even if the function is repeatedly used by the same application, the setup and teardown cost could dwarf the amount of time the function is being useful. For computing features with such an enormous overhead, using software to gain and release access to such functions may not be able to accommodate a large number of users.

(23) Embodiments of the present invention can provide reasonably fair access to all users of the network while reducing the setup cost tiny and ensure the resource can be released quickly on a successful completion of the operation. It can also ensure the resource is released reasonably quickly (e.g., a few milliseconds) when an error occurs, without the need for any management software intervention.

(24) Specifically, a reduction engine can be provided within a switch. The reduction engine can take packets from a number of endpoints and combine them to generate a single packet that can be returned to a node. The reduction engine can also perform a synchronization function, often referred to as a barrier, or can perform some mathematical function that combines or sorts the values provided by the endpoints into a single value. By placing the reduction engine within the body of the network, the latency, i.e., the time it takes to complete the operation, can be reduced by an order of magnitude because typically a single round of communication across the network is usually sufficient to complete the whole reduction.

(25) The reduction process can use a multicast session, issued from a reduction root node's edge port and sent to all the edge leaf ports, to setup or arm each of the reduction engines within the network. Each port of the switches within the network can have an instance of the reduction engine. The arming multicast (i.e., the multicast setup packet) can start at the root node's ingress edge port. When the setup packet is received, it can arm the local reduction engine associated with the corresponding port. The packet can then be multicast to a number of output ports where it is forwarded to the output's link partner ingress port, which can reside on another switch. The downstream switch can further multicast the setup packet to a set of output ports while at the same time arming an instance of a reduction engine associated with the ingress port.

(26) The above process can repeat, arming reduction engines along the multicast data path until the setup packet arrives at the egress edge ports of the leaf switches where it is passed to the compute node. At this point, all the reduction engines can be armed and ready to receive the reduction packets, which can travel upstream along the multicast tree and be reduced to a single packet that represents a reduced result.

(27) After a computation operation, the leaf nodes can be ready to inject their result packets back into the network. They can do so in a way that causes the packet to retrace on the reverse path taken by the original multicast packet. This ensures the result packets can each be intercepted by the now armed reduction engines where the reduction function can take place.
(28) The multicast tree can be traversed by the result packets in the reverse direction through the network. This reduction process does not require any software intervention, other than setting up the initial multicast tree. Modern switching devices typically can accommodate a large number of separate multicast trees, which allows many reduction configurations to be simultaneously configured in a network. As a result, the setup and teardown cost can be significantly amortized and parallelized, which allows the reduction function to be scaled to a large number of users.
(29) It is also possible to add a count value to each reduction packet to represent the number of inputs used to construct the reduction result held within the packet. This allows the acceleration provided by a reduction engine to be skipped, if necessary, without affecting the reduction function. This is possible because the receiving node is then given enough information to complete the operation itself.
(30) A timeout mechanism can also be added to the reduction engines to ensure the reduction engine resource can eventually become free, even in the presence of errors. If an error occurs, or if the reduction is not able to complete because one of the inputs to the reduction function is not present or is delayed for some reason, the timeout can ensure that the resource is released with the available input information that can be used for the reduction computation, up to that point. The root node of the reduction can receive this partial result and recognize that this result is not complete. The root node can optionally wait for the missing result to arrive without blocking the shared reduction resource.
(31) FIG. **1** shows an exemplary network. In this example, a network **100** of switches, which can also be referred to as a “switch fabric,” can include switches **102**, **104**, **106**, **108**, and **110**. Each switch can have a unique address or ID within switch fabric **100**. Various types of devices and networks can be coupled to a switch fabric. For example, a storage array **112** can be coupled to switch fabric **100** via switch **110**; an InfiniBand (IB) based HPC network **114** can be coupled to switch fabric **100** via switch **108**; a number of end hosts, such as host **116**, can be coupled to switch fabric **100** via switch **104**; and an IP/Ethernet network **118** can be coupled to switch fabric **100** via switch **102**. In general, a switch can have edge ports and fabric ports. An edge port can couple to a device that is external to the fabric. A fabric port can couple to another switch within the fabric via a fabric link. Typically, traffic can be injected into switch fabric **100** via an ingress port of an edge switch, and leave switch fabric **100** via an egress port of another (or the same) edge switch. An ingress link can couple a network interface controller (NIC) of an edge device (for example, an HPC end host) to an ingress edge port of an edge switch. Switch fabric **100** can then transport the traffic to an egress edge switch, which in turn can deliver the traffic to a destination edge device via another NIC.
(32) In one embodiment, each port of a switch can include a reduction engine that is used to accelerate reduction operations. Reductions can be performed using a multicast tree. Each reduction engine in the multicast tree can be armed by a reduction arm frame sent by a root switch through the multicast tree. After receiving the reduction arm frame, leaf nodes of the multicast tree can send reduction data frames containing their contributions up the multicast tree to the root node. Each reduction engine in the tree can intercept the reduction data frames and perform reduction on them. When a reduction engine receives the expected number of contributions or times out, it can forward the reduced result up the multicast tree. The root node may receive a single, fully reduced data frame, or, if any reduction engine times out, it may receive multiple, partially reduced data frames. In either case, the root node can complete the reduction, incorporating its own contribution. The final result of the reduction can then be sent down the multicast tree to leaf nodes. The result frame can carry another round of reduction arming instruction, which can then re-arm the reduction

engines at the same time.
(33) The reduction engine can reduce latency in critical network operations including reduce, all-reduce, and barrier. Reduction operations can be performed over a spanning tree embedded within the network. FIG. **2** shows an exemplary multicast tree for a reduction process. In this example, a multicast tree for the reduction process can include a root endpoint **202**, a root switch **204**, a number of leaf switches such as leaf switch **206**, and a number of leaf endpoints such as endpoint **208**. Root switch **204** is responsible for initiating the multicast tree for the reduction process. Each switch can include a reduction engine that can be armed when the multicast session is setup. The leaf endpoints can inject frames, which can be combined as they flow up the tree, with the result being delivered to a process running at the root of the tree. As described below, the root process may need to complete the reduction in software. This is the ready phase of a reduction. The result of a reduction can be then multicast back down the tree to processes at the leaf endpoints and the reduction engines can be re-armed, ready for the next round reduction. This is the multicast phase of a reduction process.
(34) The multicast phase of a reduction process can provide synchronization for a barrier operation, during which no data is required and a null reduction operation is used. Each node can join the reduction tree and wait for the result. When the root node receives the result, it can then issue a multicast down the reduction tree. In one embodiment, no endpoint is allowed to leave the barrier before all endpoints have entered.
(35) Reduction engines can be provided on the output side of each link. They can operate on data held in the reduction buffers. In one embodiment, each reduction engine can support eight active reduction trees. Other numbers of reduction trees can also be supported. The reduction engines can perform on-the-fly combining of data frames. The reduction engines are armed during the multicast phase. They can combine upstream frames for a given amount of time. The reduction engine can be disarmed either when the current operation has been completed, or after a timeout period. In the event of a reduction timing out, any partial results can be forwarded up the tree towards the root. The purpose of the timeout is to ensure that no reduction state remains in the event of error, device failure, or frame loss within the reduction tree.
(36) FIG. **3**A shows a flow chart of an exemplary reduction process. During operation, a root process first initializes the reduction tree (operation **302**). In HPC programming models, initialization can be a collective operation involving a number of processes that are to participate in a reduction. One process, which in this case can be the root process, can communicate with the network management software to create a spanning tree (the multicast tree), which can be represented by a multicast address. The network can use a multicast protocol to establish the multicast tree topology, and store the forwarding information in a data structure, such as a multicast table. This data structure typically stores topological and forwarding information, such as for a given multicast address, what output ports should a multicast packet be forwarded to. The root process can then arm the reduction engines in the spanning tree by sending a frame to the multicast address (operation **304**). Other processes can wait until they receive this frame. Once this frame has reached all the participating processes, the reduction tree is now ready for use.
(37) Subsequently, the participating processes can perform the computation task that results in their contribution to the reduction operation. Processes other than the root process can each construct a reduction frame and send it to the multicast address of the reduction tree (operation **306**). The reduction engines residing in the switches participating in the reduction tree can perform reduction on the received frames and each send a reduced frame upstream toward the rood switch of the reduction tree (operation **308**). The root process can consume the data reduction frames. It can receive the contributions from the leaf nodes in one or more data reduction frames and complete the ready phase by performing the reduction operation to these frames, including its own contribution (operation **310**). Optionally, the root process can then determine whether the computation task is complete (operation **312**). If it complete, the root process can send the result to

the multicast address and release the reduction engines (operation **316**). If the computation task is not complete, the root process subsequently constructs a reduction frame containing the result and sends it to the multicast address, which in turn can re-arm all the reduction engines in the reduction tree (operation **314**). This operation can prepare the reduction engines for the next round of reduction. A similar reduction process can then be repeated until the computation task is complete.
(38) The root node, or more generally a process on the root node, can perform a special role. It first completes the reduction process. As described later, loops are usually not allowed in the multicast tree; hence the root typically does not send its own contribution to itself. Assuming that the reduction engine at the root node of the reduction tree is able to accumulate all of the contributions from the leaf nodes before timing out, the root node can receive a single data reduction frame from the leaf nodes. In this case, the root node can combine this result with its own contribution. On the other hand, if the reduction engine at the root node times out or cannot be allocated, the root node may receive a number of data reduction frames that are to be combined in software. Once the root node has computed the final reduction, it can multicasts this result to the leaf nodes. The root process can also have additional responsibilities in terms of handling errors.
(39) FIG. **3**B shows a flow chart of an exemplary reduction operation by a reduction engine. During operation, a reduction engine residing on a switch can first receive a barrier frame sent to the multicast address (operation **322**). Note that the barrier frame is the initial frame that is used to arm the reduction engines for a reduction tree for the first time. After receiving the barrier frame, the reduction engine can record the parent node (i.e., the switch from which the barrier frame is received), and perform a lookup in the multicast table to determine the downstream node to which the barrier is to be forwarded (operation **324**). In addition, the reduction engine also sets a wait count, which corresponds to the number of contributions from endpoints that are expected to be received based on the multicast table entry.
(40) Subsequently, the reduction engine forwards the barrier frame to the child nodes (operation **326**). As the barrier frame travels down the reduction tree, all the reduction engines participating in this reduction tree are armed. As a result, the reduction engine at the local switch begins to receive reduction frames returned from the child nodes or endpoints (operation **328**). Next, the reduction engine can aggregate the contributions and forward a reduction frame to the parent node (operation **330**). At this point, the reduction engine is now ready for reduction operation. Next, the reduction engine can receive a result frame from the root node, which arms all the reduction engines for a reduction operation which involves actual data.
(41) The examples shown in this description assume that one process per node contributes to the reduction, which is not required. There can be multiple contributions per node. In some embodiments, a local shared memory reduction and a network reduction can both be performed. Furthermore, the reduction engine described herein can support multiple concurrent non-blocking reduction operations on the same reduction tree.
(42) The computational operations supported by a reduction engine can include, but are not limited to: Null (i.e., the barrier operation which does not involve any payload data); MIN, MAX, and SUM operations on integer or floating point data types; MINMAXLOC operation (which returns the locations of minimum and maximum values found in an array) on integer or floating point values and integer indices; Bitwise AND, OR, and XOR operation on integer data types; Reproducible sum operations on floating point data types.
(43) The data types supported by a reduction engine can include, but are not limited to 64-bit integer and 64-bit IEEE 754 floating point.
(44) In one embodiment, the MINMAXLOC operator can follow Message Passing Interface (MPI) conventions for MINLOC and MAXLOC operators when the values being compared are equal. In one embodiment, the lower of the two index values is returned.
(45) For compatibility with a commonly used modern instruction set, rounding modes and exception behavior can follow the definitions in the Advanced RISC Machine (ARM) Architecture

Reference Manual, ARMvS. For example, if any operand of a floating point operation is not a number (NaN), the result can be a quiet NaN with its sign=0.
(46) The reproducible sum and MINMAXLOC operators can use one operand per endpoint. Other reductions can be performed on four 64-bit operands at a time with the same operation being applied to each of the operands.
(47) The sum of a set of IEEE floating point values may depend on the order in which the operands are added. This can be an important issue when a reduction includes operands of widely varying magnitudes. The publication "Efficient Reproducible Floating Point Reduction Operations on Large Scale Systems," available at https://bebop.cs.berkeley.edu/reproblas/docs/talks/SIAM_AN13.pdf describes one technique that can be used to achieve the desired level of precision for a given number of elements.
(48) A deterministic reduction can be performed using a global maximum followed by a global sum using standard floating point arithmetic. A single global sum using integer arithmetic can also be used. With the second approach to reduction, the host software is to perform the same operation when multiple contributions are delivered to the root node.
(49) In general, each reduction engine can support multiple, independent reduction trees, each identified by a globally unique multicast address. Each point in the tree can be initialized with a local wait count value denoted as rt_waitcount. This count value is normally equal to the number of endpoints beneath that stage of the tree (i.e., the number of children nodes of a given node in the tree).
(50) Reduction trees can be initialized by creating an entry in a multicast table which specifies the wait count and the set of output ports. This static state, which varies between locations in the tree, can be initialized by the management software in the same way as a multicast tree.
(51) A single multicast address can be used for each reduction tree. At a parent port, the multicast table entry can specify the set of child ports. At each of the child ports, the multicast table entry can specify the parent port, i.e., the reverse path pointing back towards the root node. In general, loops are not allowed within a reduction tree. Unlike typical multicast entries, where any member of the multicast group is able to multicast to all other members of the multicast group, the multicast entries set up for reduction are one-sided and only the reduction root is able to multicast to all members of the multicast set. When any other member of the reduction tree sends a frame to the multicast address, this frame is only forwarded back to root node of the reduction tree. In addition, the forwarding of this frame typically follows exactly the reverse of the downstream multicast path from the root node. This forwarding mechanism guarantees reduction frames can be correctly intercepted by the reduction engines that have been set up for them.
(52) In one embodiment, one or more fields in a frame's header can be used as a protection key to ensure that all contributors to a given reduction are from the same application or service. For example, a virtual network identifier (VNI) field from the frame header can be used as a protection key. In addition, a frame's reduction header can contain a 32-bit cookie. All frames in a reduction can be required to have the same protection key and cookie as the frame used to arm all the reduction engines in the same reduction tree.
(53) As mentioned above, reduction trees are armed before they can be used. A multicast session can be used to arm the tree. In a global reduction process, the multicast phase that distributes the result of a reduction can re-arm the tree. In one embodiment, a reduction_arm request can include the state that is constant for all points in the reduction tree. Reduction operations on a given tree can be identified by their multicast address, a cookie rt_cookie, and a sequence number rt_seqno. All contributors can be required to provide the same protection key (which can be the VNI value), cookie value, and the correct sequence number. The reduction engine can confirm that these conditions are met. The cookie value can be used to help prevent accidental or malicious interference in the reduction. It may be a random value generated by the root process.
(54) The process of arming a reduction tree can create a dynamic state in the reduction engines for

a given tree. The wait count value can be copied from the multicast table, which indicates the number of output ports for a given multicast address. A timeout value can be determined by comparing the value of the rt_waitcount value with values programmed by the management software. The protection key, cookie value, and the sequence number can be copied from the multicast frame. A local counter rt_count, which tracks the number of received reduction frames, can be initialized to zero.
(55) All contributions to a given reduction can specify the same reduction operation, which can be identified by an rt_op value. The reduction hardware can generate an error if frames with the same sequence number specify different operations.
(56) Partial result frames can include a count of the accumulated number of contributions. The leaf endpoints can inject frames with a count of one. The reduction engine can increment the local counter by the partial counts from each frame as it performs the reduction operation. The reduction operation is complete in a given reduction engine when the local count reaches the wait count. On completion of a reduction or expiry of the timeout, the reduction engine forwards the partial result and frees up the dynamic state for the reduction tree. The static state remains in the multicast table until the multicast table entry is deleted. The reduction tree must be rearmed before it can be used again.
(57) The result of the reduction is completed by a process at the root node. In a global reduction the result can be distributed to the leaf nodes using a multicast down the reduction tree. This operation can also re-arm the tree. In one embodiment, the system can supply both the sequence number for the result being distributed, rt_resno, and the sequence number for the reduction being armed, rt_seqno. A management software can increment the sequence number from one reduction to the next. In normal operation, rt_seqno is typically one higher than rt_resno modulo the size of the counter, which may not be the case in the event of error. For upstream reduction data frames, rt_resno does not need to be set by the management software and hence can be ignored by the hardware. The reduction engine can set rt_resno equal to rt_seqno when sending frames upstream.
(58) In the example shown in FIG. **2**, the reduction tree has a branching ratio of four. Endpoints such as leaf endpoint **208** supply their contributions with a count of one. Each first stage switch in this example, such as switch **206**, has a wait count of four, which corresponds to the four leaf endpoints connected to each leaf switch. These switches can combine frames from their four children. Partial result frames with a contribution count of four are forwarded up the tree to the second stage switch, which in this example is switch **204**. Switch **204** can have a wait count of 16. This second stage switch then applies the reduction operator to four frames (one from each of its children) and forwards a result frame with count 16 to root endpoint **202**. In practice, the multicast tree may not be completely balanced, where some leaf nodes can have different contribution counts than others, and some later-stage switches can have different contribution counts from other switches at the same level.
(59) Note that the reduction engine in each switch can accelerate a component of the reduction, which may not always be necessary for proper functionality. A reduction arm command may be unable to allocate a descriptor because perhaps all of the descriptors are busy, or a reduction descriptor may have timed out before all of the results are received. In either case, data frames for this reduction may fail to find a matching descriptor and then be forwarded along the multicast path. A reduction engine in a switch higher in the multicast tree may reduce these frames or they may reach the root where they can be reduced in software.
(60) In general, a single barrier or reduction operation proceeds with each node other than the root node providing a contribution and then waiting for a result to be returned from the root node. The root node gathers contributions, completes the reduction, and multicasts the result. The sequence number is incremented from one such reduction to the next. It may be desirable to pipeline multiple reductions over the same set of nodes at the same time, for example, to offload progression of non-blocking reductions or to increase bandwidth on multi-element floating point reductions. Software

can perform multiple concurrent reductions on the same tree by distinguishing such reductions using high bits of the multicast address. For example, bits 15-3 of the multicast format Destination (61) Fabric Address (DFA), which in one embodiment is used as an inter-switch address for routing traffic within a switch fabric, can distinguish 8K multicast trees. Bits 20-18 of the same multicast address can then be used to distinguish up to 8 reductions being performed concurrently on the same tree. When a reduction engine forms part of more than one reduction tree, probability of contention can increase when multiple reductions are performed concurrently. As a result, more of the reduction operations may be performed higher in the tree.
(62) In one embodiment, a reduction engine can use a timer to limit the amount of time it spends waiting for contributions. As a result, reduction operations may time out. On expiry of the timer, the partial result of a reduction can be sent up the tree towards the root endpoint. Any further data frames associated with a reduction that has timed out can also be sent up the tree.
(63) FIG. **4** shows an example where one leaf endpoint **402** (shown in gray) is late joining the reduction process. All the rest leaf endpoints send packets with a count of one. One of the first stage switches, switch **404** (shown in gray) has received frames from three of its children when the timeout expires. It can then forward a result frame with a count of three up the tree. Sometime later, endpoint **402** supplies its contribution to the reduction. This frame is forwarded up the tree. In this example, second stage switch **406** accumulates five frames, three with counts of four, one with a count of three (from switch **404**), and the late frame with a count of one from endpoint **402**. Switch **406** subsequently sends its result to root endpoint **408** with a count of 16. In one embodiment, second stage switch **406** can have a longer timeout period to accommodate the late arrival. If all switches had the same timeout values, second stage switch **406** may forward two frames to root endpoint **408**, one with a count of 15 and the late frame from endpoint **402** with a count of one. The root endpoint can then complete the reduction.
(64) In one embodiment, the value for a reduction timeout can be set based on the expected time between reduction operations plus twice the expected variation in arrival time. High timeout values (seconds) do not alter the error free operation, but may delay the arrival of the partial results in the event of error. High timeout values can also cause reduction engine resources to be tied up for longer in the case of a dropped frame or the delayed arrival of a partial result. On the other hand, low timeout values may cause problems with scalability.
(65) The purpose of the reduction engine is to accelerate latency-sensitive operations, e.g., those in which all processes arrive at a reduction or barrier at approximately the same time. Where there is significant load imbalance and one or more endpoints arrive late, the root node can receive multiple frames. The root node can complete the reduction quickly as the last frame arrives. If the timeout is set correctly (or conservatively), the time spent processing these frames can be small in comparison with the time spent waiting.
(66) In some systems, such as Exascale systems, errors such as frame drops are usually expected to occur. The reduction mechanism can be designed to still function well in the presence of errors. Two classes of error can be of importance for reduction operations: link errors that cause frames to be corrupted and device errors that cause frames to be dropped. Where available, link level retry can protect against common link errors; therefore, the dominant error case can be expected to be dropped frames arising from a switch or cable failure. Where forward error correction (FEC) is used without link level retry, a small proportion of link errors can result in frame drops. Most such frame drops can occur on bulk data transfer rather than reductions. However, the likelihood of an error causing reduction frames to be lost can increase with the job size.
(67) The time required to detect errors in reduction operations arising from frame loss can generally be counted in seconds. Under normal operation, this period can be set to be longer than the expected spread of arrival times or the time that nodes might spend in computation while waiting for a non-blocking reduction to complete. After this period, the root process can reasonably be assumed to be blocked waiting for the reduction to complete. However, there are many

examples where processes wait for long periods of time in reduction or barrier operations while other processes complete sequential work. A long time spent in a reduction does not always imply that an error has occurred.
(68) FIG. **5**A shows an example where one leaf endpoint fails to supply a contribution because of an error. In this example, leaf endpoint **502** experiences an error and does not supply its contribution to the reduction process. As a result, intermediate switch **504** times out and forwards a partial result with a count of three. Root switch **506** again times out (because a total count of 16 is not received before switch **506**'s timer expires) and forwards a frame with a count of 15, or perhaps a frame with a count of 12 and a second, subsequent frame with a count of three. In either case, root endpoint **508** then consumes the frame(s) and determines that there has been an error.
(69) FIG. **5**B shows a flow chart of an exemplary timer-based reduction process. In this example, a reduction engine first receives a result frame from the root node, which arms the reduction engine (operation **522**). The reduction engine then becomes armed for the reduction tree identified in the result frame, and forwards the result frame to its child nodes in the reduction tree (operation **524**). Next, the reduction engine determines whether a sufficient number of reduction contributions have been received (operation **526**). This determination is based on the corresponding wait count, which has been set up during the ready phase when the reduction tree is initiated for the first time. If the wait count has been met, the reduction engine then performs the reduction operation on the received contributions, generates its own reduction frame, and forwards the reduction frame toward the root (operation **532**). If the wait count is not satisfied, the reduction engine further determines whether the timer has expired (operation **528**). If the timer has not expired, the reduction engine continues to wait for reduction frames to arrive from child nodes (operation **530**). If the timer has expired, the reduction engine then performs reduction operation on the received reduction frames, if any, and forwards its own reduction frame toward the root node (operation **532**). If there are any late contribution frames that arrive after the timer expires, the reduction engine can also forward them toward the root node (operation **534**).
(70) In one embodiment, reliable reductions can be implemented at the transport layer. The network hardware can be designed to accelerate the common case and to free all hardware resources in the case of error. Software can protect against device failure in the reduction tree by performing the same reduction operation over two independent trees. The probability of uncorrelated double error is small. When the root endpoint receives a result (or sufficient frames to construct the result) from one tree the host software can multicast the result on both trees. The sequence number from the successful operation can be used as the result number.
(71) In the case where the second tree is left with a partial result, potentially several steps back, or stranded on a congested link or a busy queue, delivery of the reduction multicast frame with the rt_arm bit set can advance the sequence number and clear the state. Frames with an old sequence number can be dropped. One or more frames may still be in flight up the tree.
(72) Reductions are typically latency sensitive, and not bandwidth intensive. The additional network load created by performing two simultaneous reductions is usually negligible. An added benefit of performing reductions twice, on distinct trees, is that the first result can be used, potentially reducing the time to complete the operation if the network is congested. The drawback to this approach, however, is that twice as many Reduction resources (e.g., multicast table entries and reduction engines/IDs) are used, potentially reducing the number of simultaneous reductions by a factor of two.
(73) A correlated double error may arise as a result of chassis power failure, but such an error may likely cause node failures as well. In a multi-slice network, software can create reductions trees on different slices. Where there is only a single network slice, trees on the same slice can share a common-chassis switch. The reduction is vulnerable to loss of this switch, but such an error can disconnect the nodes as well.
(74) Allowing reductions to time out ensures that no reduction state is left behind in the event of

error. There does not need to be requirement to issue management requests to search and release hardware resources after a fault. In some embodiments, reduction state can be left in the network in the event of error and fault recovery can be complex.
(75) In some embodiments, a flexible reduction protocol can be provided where a missing reduction engine, or one where all resources have been consumed, does not render the protocol dysfunctional. A missing early leaf reduction computation can be completed by a reduction engine higher in the tree, as shown in FIG. **6**. In this example, the reduction engine on a leaf switch **604** is unavailable for the reduction process. As a result, leaf switch **604** forwards all four contribution frames generated by the four children leaf endpoints to a root switch **606**. Meanwhile, other three leaf switches **601**, **602**, and **603** each send their respective reduction frame (with a count of four) to root switch **606**. Root switch can process all these seven frames and forwards a single reduction frame with a count of 16 to root endpoint **608**. In general, a reduction computation high in the tree can be completed by a process running on the root node. The acceleration offered by the reduction engines in those cases may be different but the calculated result remains the same.
(76) If a reduction tree is armed, but no contributions are provided before all of the active reduction engines time out, then all contributions can be forwarded to the root endpoint. The root process can subsequently perform the entire computation. The time required to perform this computation can be much shorter than the timeout period. This reduction scenario does not require acceleration.
(77) The maximum number of reduction trees that can be supported on a given system can be determined by the product of the number of endpoints that can act as root and the number of active trees supported by a given reduction engine. There is usually limited value in accelerating reductions over small numbers of nodes, because each node can simply send its contribution to the root node. Suppose the goal is to accelerate reductions of sixteen nodes and above. A system might run a thousand or more such jobs, while large systems tend to run a mix of job sizes, which in turn may reduce the number of active reduction trees. A large application might use multiple reduction trees at the same time. In the canonical example, the processes can be arranged in a 2D mesh and reductions are performed over the rows, the columns, and over the entire mesh. For P processes, the number of active reductions can be twice the square root of P plus one. For implementation considerations, the multicast table is relatively expensive in terms of die area. In one embodiment, a switch chip can support 8192 multicast addresses, while other numbers are also possible. Note that non-intersecting trees may use the same multicast address.
(78) In some embodiments, software can decide which reductions to offload. A request can be sent to the network management system to create a reduction tree. If this request fails the network application interface (API) can perform the operation in software. However, running different instances of the same job with software-based reductions or accelerated reductions can potentially lead to performance variation, which is undesirable. The reduction offload strategy can be configured such that running out of reduction trees is unlikely.
(79) In one embodiment, a 6-bit command field is used to indicate different reduction operations, as shown in FIG. **7**. This 6-bit command field can provide scope for expansion if needed. Multiple concurrent reductions may operate on the same tree. Each reduction can use a different reduction ID in the DFA. The wait count can be set to be sufficient for one contribution per node in a maximally sized system. In one embodiment, the wait count can be a 20-bits value.
(80) In addition, four configurable timeout values can be provided. These 24-bit values can be given in units of 1024 clock cycles. At 850 MHz, this provides a range from 1.20 us to 20.2 s. In addition, a 10-bit sequence number can be used, which can avoid being reused while an old reduction frame with the same multicast address and the same cookie value is still in the network. Note that this problem does not occur on the same reduction tree, because in-order delivery prevents an old reduction frame from being delivered after frames transmitted later on the same tree. Therefore, for a late-arriving frame to cause a problem, the process it belongs to needs to exit first so that a newly launched process could request a new multicast tree, which can be built using

the same multicast address. The chance that the old reduction frame survives the reprogramming of the multicast table is low. If it did survive, it still needs to intersect the new tree after the new tree performs 1024 reductions. Note that the 32-bit cookie value can also provide an extra layer of protection.
(81) FIG. **7** shows an exemplary list of reduction operations, which are explained below. The FLT_REP SUM and the MINMAXLOC operations support one reduction at a time per tree. For all other operations, four reductions per tree can be performed in parallel. The floating point sum operations can have four rounding modes and flush-to-zero (FTZ), which can be encoded in three bits as eight different commands. Floating point MIN and MAX have two modes for handling NaNs (mirroring the ARM MIN and MINNUM operations). FTZ only applies to denormalized results. If a result is to be denormalized, it is set to 0 instead and flt_inexact is raised. The operation sometimes known as DAZ may be supported in software at the leaf nodes.
(82) BARRIER: The data returned by BARRIER operations is always 0.
(83) MINMAXLOC: The MINMAXLOC operators are used to support the MINLOC and MAXLOC. Operands 0 and 1 compute MINLOC, and operands 2 and 3 compute MAXLOC. FIG. **8** shows a set of MINMAXLOC operands. Note that when more than one index contains the minimum/maximum value, the lowest such index is recorded in the MINLOC/MAXLOC field.
(84) FLT MIN and FLT MAX: When all inputs of FLT_MIN or FLT_MAX are floating point numbers, the minimum or maximum value is returned. When any operand of FLT_MIN or FLT_MAX is a NaN, NaN is returned. In a given pairwise reduction, if one operand is a signaling NaN and one operand is a quiet NaN, the signaling NaN is selected to be returned. The NaN returned can be turned into a quiet NaN with its sign bit cleared.
(85) FLT MINNUM and FLT MAXNUM: These operations are similar to FLT_MIN and FLT_MAX but handle operands that are NaNs differently. In the absence of a signaling NaN, FLT_MINNUM and FLT_MAXNUM can return the smallest/largest numbers in the reduction. A quiet NaN is only returned when all of the operands in one of these reductions are quiet NaNs.
(86) The behavior of signaling NaNs with regard to these operators can be controlled by a R_TF_RED_CFG_MODE register. In standard IEEE mode, a signaling NaN (SNaN) as an operand of a pairwise reduction always produces a quiet NaN as a result. This produces indeterminate results for the complete reduction. In the recommended associative mode, when one operand is SNaN and the other is a number, the result is the number. Thus, in associative mode, if at least one operand in the reduction is a floating point number, the minimum or maximum floating point operand is returned. In either mode, if any operand is a signaling NaN, flt_invalid is returned.
(87) FLT_MINMAXNUMLOC: This operation computes both FLT_MINNUM and FLT_MAXNUM.
(88) FLT_SUM: This floating point sum operation has a flush-to-zero option and four rounding modes. When flush-to-zero is enabled, if the sum is denormalized, it is set to 0. The sign of the denormalized result is preserved. The four rounding modes match the ARM rounding modes and are shown in FIG. **9**.
(89) FLT_REPSUM: The reproducible floating point sum is accomplished by splitting each floating point operand into up to four integer components, each of which has limited precision. The number of significant bits (W) in each component is selected such that integer overflow cannot occur. The value of W is selected by software and is not observable in the hardware. W is used to compute the integer values IX to load into the reduction operand rt_data. When the reduction is complete, W is used to construct the floating point result. The software may choose to set W to 40; in this case, up to 2{circumflex over ( )}24 operands may be reduced. A floating point number can be represented by up to four W-bit integers as follows:
Σ.sub.j=M.sup.M+4IX[j−M]×2.sup.W×j.
(90) For each floating point operand, the software chooses the largest value of M such that the least significant bit of the operand appears in IX[0]. Software is responsible for loading the four IX

values into rt_data and the value of M in rt_repsum_m, which is an eight-bit signed integer. IX[0], the least significant operand, is loaded into rt_data[0].
(91) When two operands in this format are added by the reduction engine, if one operand's M, M′, is larger than the other, the hardware can discard any IX[j] in the smaller operand where j<M′ because these values may not have significance in the final result. If this occurs during the course of the reduction and any nonzero operands are dropped, repsum_inexact can be returned.
(92) When the reduction is complete, the root process can convert the resulting operands and rt_repsum_m into a floating point number. If there are more operands than are supported by the chosen W, int_overtflow may be returned. In this case, the result is not valid. Note that int_overflow is only reported if the overflow occurs in one of the significant values returned in the result. The rt_repsum_oflow_id identifies the most significant operand to overflow. When a partial result with int_overflow is reduced with another partial result, if (M+rt_repsum_oflow_id) of the overflow partial result is less than M′ of the other partial result, the int_overflow result code is dropped.
(93) In some embodiments, static state for reduction operations can be programmed in the multicast table. This state can vary between devices in the same reduction tree. The state can be created by the management agent when a job starts or a new reduction tree is created. It can use the same mechanism as is used for setting up standard multicast entries. The multicast reduction trees are distinguished from multicast trees by a non-zero wait count. The timeout value for a reduction in a particular reduction engine can be determined by its wait count value and the configuration.
(94) The reduction frame format is the same for reduction_arm and reduction_data frames. They can be distinguished by the rt_arm field which can be set on all frames descending the tree. The 84-byte Portals-formatted reduction frame is shown in FIG. **10**.
(95) The Portals header and command fields are stored once in the reduction state. The 12-byte reduction header is shown in FIG. **11**. The endianness of the operands, which is used for the MINMAXLOC reproducible sum operators, is shown in in FIG. **12**.
(96) Errors or inexact results encountered during the course of a reduction are reported in rt_rc. In general, these events do not prevent the reduction from completing. However, in the event of an opcode mismatch, the reduction cannot be performed. In this case, the Source Fabric Addresses (SFA) of two operands with differing opcodes are returned in rt_data[0]. The reduction result codes are summarized in FIG. **13**.
(97) There is only one result code shared by the four parallel reductions. Result codes are defined in priority order. If a reduction encounters more than one exception condition, the largest is retained. For example, flt_invalid is the highest priority FLT_SUM result code.
(98) In some embodiments, the reduction engine can be utilized from an Ethernet NIC which is generally not equipped to operate with the HPC Ethernet specification and utilize the Portals packet format. To facilitate reduction using reduction engines, the system can use the Soft Portals encapsulation, in which the Portals packet can be prepended with an Ethernet header constructed to be consistent with the configuration for Portals on the port that the NIC is connected to. This is illustrated in FIG. **14**.
(99) Within the Linux operating system, it is possible to construct these packets using a raw socket; this is suitable for functional testing since it requires the process to execute as root or with CAP_NET_RAW. The socket can be opened to receive only the specified ether type that a switch is configured to use in the Ethernet header it prepends to the Portals packets. It should be noted that the VNI field of the Portals packet is the protection mechanism, this can be inserted in a privileged domain and for production usage this can be performed in a kernel module.
(100) FIG. **15** shows an exemplary switching system that facilitates a reduction engine. In this example, a switch **1502** can include a number of communication ports, such as port **1520**. Each port can include a transmitter and a receiver. Switch **1502** can also include a processor **1504**, a storage device **1506**, and a switching logic block **1508**. Switching logic block **1508** can be coupled

to all the communication ports and can further include a crossbar switch **1510** and a reduction engine logic block **1514**.
(101) Crossbar switch **1510** can include one or more crossbar switch chips, which can be configured to forward data packets and control packets among the communication ports. Reduction engine logic block **1514** can be configured to perform various dynamic reduction functions as described above. Also included in switching logic block **1508** is a multicast table **1516**, which can store the reduction tree topology and state information to facilitate the reduction operations performed by reduction engine logic block **1514**. Other types of data structure can also be used to store the topology and state information.
(102) In summary, the present disclosure describes a switch capable of dynamically allocating a reduction engine in a network. The switch is equipped with a reduction engine that can be dynamically allocated based on a multicast frame to perform on-the-fly reduction. As a result, the network can facilitate an efficient and scalable environment for high performance computing.
(103) The methods and processes described above can be performed by hardware logic blocks, modules, logic blocks, or apparatus. The hardware logic blocks, modules, logic blocks, or apparatus can include, but are not limited to, application-specific integrated circuit (ASIC) chips, field-programmable gate arrays (FPGAs), dedicated or shared processors that execute a piece of code at a particular time, and other programmable-logic devices now known or later developed. When the hardware logic blocks, modules, or apparatus are activated, they perform the methods and processes included within them.
(104) The methods and processes described herein can also be embodied as code or data, which can be stored in a storage device or computer-readable storage medium. When a processor reads and executes the stored code or data, the processor can perform these methods and processes.
(105) The foregoing descriptions of embodiments of the present invention have been presented for purposes of illustration and description only. They are not intended to be exhaustive or to limit the present invention to the forms disclosed. Accordingly, many modifications and variations will be apparent to practitioners skilled in the art. Additionally, the above disclosure is not intended to limit the present invention. The scope of the present invention is defined by the appended claims.

Claims

1. A switch, comprising: a number of ports; and a reduction engine coupled to at least one port and to: be armed for a reduction process based on a received frame, wherein the received frame comprises a multicast address; determine a number of output ports based on the multicast address; and send the frame to the determined output ports, thereby allowing additional reduction engines operating on additional switches downstream from the switch to be armed for the reduction process.
2. The switch of claim 1, wherein the frame uniquely identifies the reduction process.
3. The switch of claim 1, wherein the reduction engine is further to set a wait count for the reduction process based on a number of endpoints below the switch based on a multicast tree corresponding to the multicast address.
4. The switch of claim 3, wherein the reduction engine is further to: identify one or more received reduction frames associated with the reduction process; determine that a total number of contributions indicated by the received reduction frames is equal to the wait count; and perform a reduction operation based on the contributions to generate a reduction frame to be forwarded to a parent node.
5. The switch of claim 4, wherein the generated reduction frame comprises a count of total contributions based on the received reduction frames.
6. The switch of claim 1, wherein the reduction engine is further to record a parent node from which the frame is received, thereby facilitating subsequent forwarding of a reduction frame toward

a root node.
7. The switch of claim 1, wherein while determining the output ports the reduction engine is to look up a multicast table based on the multicast address.
8. A method, comprising: arming a reduction engine of a switch for a reduction process based on a received frame, wherein the received frame comprises a multicast address; determining a number of output ports based on the multicast address; and sending the frame to the determined output ports, thereby allowing additional reduction engines operating on additional switches downstream from the switch to be armed for the reduction process.
9. The method of claim 8, wherein the frame uniquely identifies the reduction process.
10. The method of claim 8, further comprising setting a wait count for the reduction process based on a number of endpoints below the switch based on a multicast tree corresponding to the multicast address.
11. The method of claim 10, further comprising: identifying one or more received reduction frames associated with the reduction process; determining that a total number of contributions indicated by the received reduction frames is equal to the wait count; and performing a reduction operation based on the contributions to generate a reduction frame to be forwarded to a parent node.
12. The method of claim 11, wherein the generated reduction frame comprises a count of total contributions based on the received reduction frames.
13. The method of claim 8, further comprising recording a parent node from which the frame is received, thereby facilitating subsequent forwarding of a reduction frame toward a root node.
14. The method of claim 8, wherein determining the output ports comprises looking up a multicast table based on the multicast address.
15. A network system, comprising: a number of interconnected switches, wherein a respective switch comprises: a number of ports; and an reduction engine coupled to at least one port and to: be armed for a reduction process based on a received frame, wherein the received frame comprises a multicast address; determine a number of output ports based on the multicast address; and send the frame to the determined output ports, thereby allowing additional reduction engines to be armed for the reduction process, the additional reduction engines being provided in switches of the number of interconnected switches downstream from the respective switch.
16. The network system of claim 15, wherein the frame uniquely identifies the reduction process.
17. The network system of claim 15, wherein the reduction engine is further to set a wait count for the reduction process based on a number of endpoints below the switch based on a multicast tree corresponding to the multicast address.
18. The network system of claim 17, wherein the reduction engine is further to: identify one or more received reduction frames associated with the reduction process; determine that a total number of contributions indicated by the received reduction frames is equal to the wait count; and perform a reduction operation based on the contributions to generate a reduction frame to be forwarded to a parent node.
19. The network system of claim 18, wherein the generated reduction frame comprises a count of total contributions based on the received reduction frames.
20. The network system of claim 15, wherein the reduction engine is further to record a parent node from which the frame is received, thereby facilitating subsequent forwarding of a reduction frame toward a root node.
21. The network system of claim 15, wherein while determining the output ports the reduction engine is to look up a multicast table based on the multicast address.