US00RE48589E

(19) United States
(12) Reissued Patent
Garrod et al.

(10) Patent Number: US RE48,589 E
(45) Date of Reissued Patent: Jun. 8, 2021

(54) **SHARING AND DECONFLICTING DATA CHANGES IN A MULTIMASTER DATABASE SYSTEM**

(71) Applicant: **Palantir Technologies, Inc.**, Palo Alto, CA (US)

(72) Inventors: **John Kenneth Garrod**, Palo Alto, CA (US); **John Antonio Carrino**, Palo Alto, CA (US); **Katherine Brainard**, East Orange, NJ (US); **Jacob Scott**, Berkeley, CA (US); **Allen Chang**, Mountain View, CA (US)

(73) Assignee: **Palantir Technologies Inc.**, Palo Alto, CA (US)

(21) Appl. No.: **14/830,420**

(22) Filed: **Aug. 19, 2015**

Related U.S. Patent Documents

Reissue of:
(64) Patent No.: **8,515,912**
Issued: **Aug. 20, 2013**
Appl. No.: **12/836,801**
Filed: **Jul. 15, 2010**

(51) **Int. Cl.**
G06F 16/23 (2019.01)

(52) **U.S. Cl.**
CPC ***G06F 16/2329*** (2019.01)

(58) **Field of Classification Search**
CPC G06F 17/30578; G06F 16/2329
USPC 707/610
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,881,179 A 11/1989 Vincent
5,241,625 A 8/1993 Epard et al.
5,548,749 A 8/1996 Kroenke et al.
5,708,828 A 1/1998 Coleman
5,765,171 A 6/1998 Gehani et al. G06F 16/27
5,774,717 A 6/1998 Porcaro
5,806,074 A 9/1998 Souder et al.
5,845,300 A 12/1998 Comer
5,870,761 A 2/1999 Demers et al.

(Continued)

FOREIGN PATENT DOCUMENTS

AU 2011279270 9/2015
AU 2013251186 11/2015

(Continued)

OTHER PUBLICATIONS

U.S. Appl. No. 14/518,757, filed Oct. 20, 2014, Office Action, dated Dec. 1, 2015.

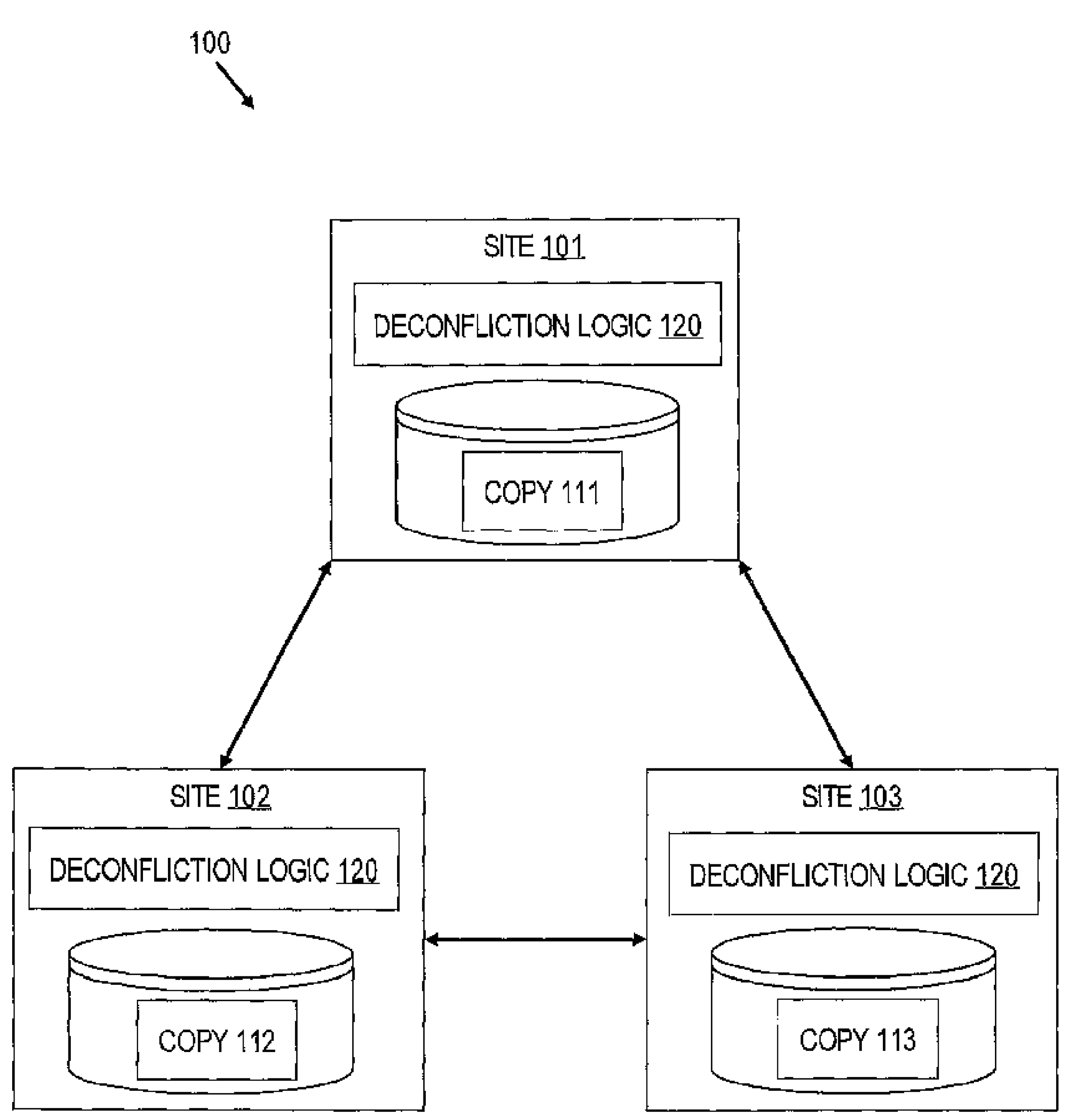
(Continued)

Primary Examiner — Fred O Ferris, III
(74) *Attorney, Agent, or Firm* — Christine E. Orich; Hickman Becker Bingham Ledesma LLP

(57) **ABSTRACT**

A computer-based method and system for sharing and deconflicting data changes amongst a plurality of replication sites. In a particular embodiment, data changes at sites to data objects are tracked by each site on a per-data object basis using per-data object version vectors. In another particular embodiment, data changes at sites to links connecting two data objects are tracked by each site on a per-link set basis using per-link set version vectors. In another particular embodiment, per-object version vectors are used to detect a conflict resulting from concurrent changes at two or more sites in which one of the concurrent changes includes an object resolution change.

8 Claims, 7 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

5,943,676	A	8/1999	Boothby
5,978,475	A	11/1999	Schneier et al.
5,999,911	A	12/1999	Berg et al.
6,065,026	A	5/2000	Cornelia et al.
6,098,078	A *	8/2000	Gehani et al. G06F 16/27 707/610
6,101,479	A	8/2000	Shaw
6,190,053	B1	2/2001	Stahlecker et al.
6,202,085	B1	3/2001	Benson et al.
6,216,140	B1 *	4/2001	Kramer 715/234
6,232,971	B1	5/2001	Haynes
6,237,138	B1	5/2001	Hameluck et al.
6,240,414	B1	5/2001	Beizer et al.
6,243,706	B1	6/2001	Moreau et al.
6,279,018	B1	8/2001	Kudrolli et al.
6,289,338	B1	9/2001	Stoffel et al.
6,317,754	B1 *	11/2001	Peng 707/610
6,370,538	B1	4/2002	Lamping et al.
6,374,252	B1	4/2002	Althoff et al.
6,430,305	B1	8/2002	Decker
6,463,404	B1	10/2002	Appleby
6,523,019	B1	2/2003	Borthwick
6,523,172	B1	2/2003	Martinez-Guerra et al.
6,539,381	B1 *	3/2003	Prasad et al. G06F 16/27
6,539,538	B1	3/2003	Brewster et al.
6,560,620	B1 *	5/2003	Ching 715/229
6,640,231	B1	10/2003	Andersen et al.
6,642,945	B1	11/2003	Sharpe
6,665,683	B1	12/2003	Meltzer
6,725,240	B1	4/2004	Asad et al.
6,748,481	B1	6/2004	Parry et al.
6,807,569	B1	10/2004	Bhimani et al.
6,816,941	B1	11/2004	Carlson et al.
6,850,317	B2	2/2005	Mullins et al.
6,877,137	B1	4/2005	Rivette et al.
6,944,777	B1	9/2005	Belani et al.
6,944,821	B1	9/2005	Bates et al.
6,967,589	B1	11/2005	Peters
6,978,419	B1	12/2005	Kantrowitz
7,017,046	B2	3/2006	Doyle et al.
7,027,974	B1	4/2006	Busch et al.
7,072,911	B1	7/2006	Doman
7,086,028	B1	8/2006	Davis et al.
7,089,541	B2	8/2006	Ungar
7,167,877	B2	1/2007	Balogh et al.
7,174,377	B2	2/2007	Bernard et al.
7,194,680	B1	3/2007	Roy et al.
7,213,030	B1	5/2007	Jenkins
7,225,468	B2	5/2007	Waisman et al.
7,237,192	B1	6/2007	Stephenson et al.
7,240,330	B2	7/2007	Fairweather
7,302,708	B2	11/2007	Kovarik
7,392,254	B1	6/2008	Jenkins
7,437,664	B2 *	10/2008	Borson 715/234
7,441,182	B2	10/2008	Beilinson et al.
7,441,219	B2	10/2008	Perry et al.
7,523,146	B2	4/2009	Holt et al.
7,530,105	B2	5/2009	Gilbert et al.
7,533,069	B2	5/2009	Fairweather
7,596,285	B2	9/2009	Brown et al.
7,627,812	B2	12/2009	Chamberlain et al.
7,634,717	B2	12/2009	Chamberlain et al.
7,664,829	B2	2/2010	Yamamoto et al.
7,676,788	B1 *	3/2010	Ousterhout et al. 717/106
7,685,083	B2	3/2010	Fairweather
7,707,178	B2	4/2010	Prahlad et al.
7,716,140	B1	5/2010	Nielsen et al.
7,730,396	B2	6/2010	Chidlovskii et al.
7,765,489	B1	7/2010	Shah
7,770,032	B2	8/2010	Nesta et al.
7,770,100	B2	8/2010	Chamberlain et al.
7,801,871	B2	9/2010	Gosnell
7,805,408	B2	9/2010	Padgett et al.
7,818,297	B2	10/2010	Peleg et al.
7,877,421	B2	1/2011	Berger et al.
7,880,921	B2	2/2011	Dattilo et al.
7,941,336	B1	5/2011	Robin-Jan
7,953,710	B2 *	5/2011	Novik G06F 17/30578 707/662
7,958,147	B1	6/2011	Turner et al.
7,962,495	B2	6/2011	Jain et al.
7,962,848	B2	6/2011	Bertram
7,966,199	B1	6/2011	Frasher
8,001,465	B2	8/2011	Kudrolli et al.
8,001,482	B2	8/2011	Bhattiprolu et al.
8,010,507	B2	8/2011	Poston et al.
8,015,151	B2	9/2011	Lier et al.
8,073,857	B2	12/2011	Sreekanth
8,117,022	B2	2/2012	Linker
8,132,149	B2	3/2012	Shenfield et al.
8,190,893	B2	5/2012	Benson et al.
8,191,005	B2	5/2012	Baier et al.
8,196,184	B2	6/2012	Amirov et al.
8,225,201	B2	7/2012	Michael
8,239,668	B1	8/2012	Chen et al.
8,271,948	B2	9/2012	Talozi et al.
8,290,838	B1	10/2012	Thakur et al.
8,290,990	B2	10/2012	Drath et al.
8,301,904	B1	10/2012	Gryaznov
8,302,855	B2	11/2012	Ma et al.
8,312,367	B2	11/2012	Foster
8,312,546	B2	11/2012	Alme
8,316,060	B1	11/2012	Snyder et al.
8,380,659	B2	2/2013	Zunger
8,392,556	B2	3/2013	Goulet et al.
8,442,940	B1	5/2013	Faletti et al.
8,489,623	B2	7/2013	Jain et al.
8,515,912	B2	8/2013	Garrod
8,527,461	B2	9/2013	Ducott, III et al.
8,527,949	B1	9/2013	Pleis et al.
8,560,494	B1	10/2013	Downing
8,620,641	B2	12/2013	Farnsworth et al.
8,646,080	B2	2/2014	Williamson et al.
8,682,696	B1	3/2014	Shanmugam
8,688,573	B1	4/2014	Ruknoic et al.
8,688,749	B1	4/2014	Ducott, III et al.
8,689,182	B2	4/2014	Leithead et al.
8,726,379	B1	5/2014	Stiansen et al.
8,732,574	B2	5/2014	Burr et al.
8,782,004	B2	7/2014	Ducott
8,799,313	B2	8/2014	Satlow
8,807,948	B2	8/2014	Luo et al.
8,838,538	B1	9/2014	Landau et al.
8,886,601	B1	11/2014	Landau et al.
8,903,717	B2	12/2014	Elliot
8,930,874	B2	1/2015	Duff et al.
8,930,897	B2	1/2015	Nassar
8,938,686	B1	1/2015	Erenrich et al.
8,984,390	B2	3/2015	Aymeloglu et al.
9,009,827	B1	4/2015	Albertson et al.
9,021,260	B1	4/2015	Falk et al.
9,058,315	B2	6/2015	Burr et al.
9,165,100	B2	10/2015	Begur et al.
9,189,492	B2	11/2015	Ducott
9,286,373	B2	3/2016	Elliot et al.
9,501,552	B2	11/2016	McGrew
9,569,070	B1	2/2017	Ma et al.
2001/0021936	A1	9/2001	Bertram
2002/0032677	A1	3/2002	Morgenthaler et al.
2002/0095360	A1	7/2002	Joao
2002/0103705	A1	8/2002	Brady
2002/0112157	A1	8/2002	Doyle et al.
2002/0196229	A1	12/2002	Chen et al.
2003/0036927	A1	2/2003	Bowen
2003/0055825	A1	3/2003	Chen et al.
2003/0061132	A1	3/2003	Mason et al.
2003/0084017	A1	5/2003	Ordille
2003/0088654	A1	5/2003	Good et al.
2003/0093755	A1	5/2003	O'Carroll
2003/0126102	A1	7/2003	Borthwick
2003/0028560	A1	9/2003	Kudrolli et al.
2003/0172053	A1	9/2003	Fairweather
2003/0177112	A1	9/2003	Gardner
2003/0182313	A1	9/2003	Federwisch et al.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2004/0034570 A1 2/2004 Davis
2004/0044648 A1 3/2004 Anfindsen et al.
2004/0044992 A1 3/2004 Muller et al.
2004/0078451 A1 4/2004 Dietz et al.
2004/0083466 A1 4/2004 Dapp et al.
2004/0103124 A1* 5/2004 Kupkova 707/203
2004/0103147 A1 5/2004 Flesher et al.
2004/0111390 A1* 6/2004 Saito et al. 707/1
2004/0153418 A1 8/2004 Hanweck
2004/0205492 A1 10/2004 Newsome
2004/0221223 A1 11/2004 Yu et al.
2004/0236688 A1 11/2004 Bozeman
2004/0236711 A1 11/2004 Nixon et al.
2004/0250124 A1 12/2004 Chesla et al.
2004/0250576 A1 12/2004 Flanders
2005/0010472 A1 1/2005 Quatse et al.
2005/0028094 A1 2/2005 Allyn
2005/0034107 A1 2/2005 Kendall et al.
2005/0039116 A1 2/2005 Slack-Smith
2005/0039119 A1 2/2005 Parks et al.
2005/0044187 A1 2/2005 Jhaveri et al.
2005/0050537 A1 3/2005 Thompson et al.
2005/0091186 A1 4/2005 Elish
2005/0091420 A1 4/2005 Snover et al.
2005/0097061 A1 5/2005 Shapiro et al.
2005/0108063 A1 5/2005 Madill et al.
2005/0125715 A1 6/2005 Franco et al.
2005/0183005 A1 8/2005 Denoue et al.
2005/0193024 A1 9/2005 Beyer et al.
2005/0229256 A2 10/2005 Banzhof
2005/0267865 A1 12/2005 Bird et al.
2006/0026561 A1 2/2006 Bauman et al.
2006/0031779 A1 2/2006 Theurer et al.
2006/0036568 A1 2/2006 Moore et al.
2006/0045470 A1 3/2006 Poslinski et al.
2006/0053097 A1 3/2006 King et al.
2006/0053170 A1 3/2006 Hill et al.
2006/0059423 A1 3/2006 Lehmann et al.
2006/0069912 A1 3/2006 Zheng et al.
2006/0074866 A1 4/2006 Chamberlain et al.
2006/0080139 A1 4/2006 Mainzer
2006/0106879 A1 5/2006 Zondervan et al.
2006/0129746 A1 6/2006 Porter
2006/0136513 A1 6/2006 Ngo et al.
2006/0143075 A1 6/2006 Carr et al.
2006/0155654 A1 7/2006 Plessis et al.
2006/0155945 A1 7/2006 McGarvey
2006/0178915 A1 8/2006 Chao
2006/0190497 A1 8/2006 Inturi et al.
2006/0206866 A1 9/2006 Eldrige et al.
2006/0218637 A1 9/2006 Thomas et al.
2006/0224579 A1 10/2006 Zheng
2006/0242204 A1 10/2006 Karas et al.
2006/0265377 A1 11/2006 Raman et al.
2006/0265417 A1 11/2006 Amato et al.
2006/0265747 A1 11/2006 Judge
2006/0271526 A1 11/2006 Charnock et al.
2006/0277460 A1 12/2006 Forstall et al.
2007/0000999 A1 1/2007 Kubo et al.
2007/0005707 A1 1/2007 Teodosiu et al.
2007/0018986 A1 1/2007 Hauser
2007/0026373 A1 2/2007 Suriyanarayanan et al.
2007/0043686 A1 2/2007 Teng et al.
2007/0061752 A1 3/2007 Cory
2007/0074169 A1 3/2007 Chess et al.
2007/0078872 A1 4/2007 Cohen
2007/0112714 A1 5/2007 Fairweather
2007/0112887 A1 5/2007 Liu et al.
2007/0113164 A1 5/2007 Hansen et al.
2007/0130217 A1 6/2007 Linyard et al.
2007/0136095 A1 6/2007 Weinstein
2007/0168516 A1 7/2007 Liu et al.
2007/0168871 A1 7/2007 Jenkins
2007/0174760 A1 7/2007 Chamberlain et al.
2007/0180075 A1 8/2007 Chasman et al.
2007/0185850 A1 8/2007 Walters et al.
2007/0220067 A1 9/2007 Suriyanarayanan et al.
2007/0220328 A1 9/2007 Liu et al.
2007/0233756 A1 10/2007 D'Souza et al.
2007/0245339 A1 10/2007 Bauman et al.
2007/0284433 A1 12/2007 Domenica et al.
2007/0294766 A1 12/2007 Mir et al.
2007/0299697 A1 12/2007 Friedlander et al.
2007/0299887 A1 12/2007 Novik et al.
2008/0005188 A1* 1/2008 Li G06F 17/30174
2008/0016155 A1 1/2008 Khalatian
2008/0027981 A1 1/2008 Wahl
2008/0033753 A1* 2/2008 Canda et al. 705/2
2008/0086718 A1* 4/2008 Bostick et al. 717/120
2008/0091693 A1 4/2008 Murthy
2008/0109714 A1 5/2008 Kumar et al.
2008/0140387 A1 6/2008 Linker
2008/0141117 A1 6/2008 King et al.
2008/0148398 A1 6/2008 Mezack et al.
2008/0168135 A1 7/2008 Redlich et al.
2008/0172607 A1 7/2008 Baer
2008/0177782 A1 7/2008 Poston et al.
2008/0186904 A1 8/2008 Koyama et al.
2008/0189240 A1 8/2008 Mullins et al.
2008/0201580 A1 8/2008 Savitzky et al.
2008/0228467 A1 9/2008 Womack et al.
2008/0229422 A1 9/2008 Hudis et al.
2008/0235575 A1 9/2008 Weiss
2008/0243951 A1 10/2008 Webman et al.
2008/0249820 A1 10/2008 Pathria
2008/0276167 A1 11/2008 Michael
2008/0281580 A1 11/2008 Zabokritski
2008/0288475 A1 11/2008 Kim et al.
2008/0313132 A1 12/2008 Hao et al.
2008/0313243 A1 12/2008 Poston et al.
2008/0320299 A1 12/2008 Wobber et al.
2009/0024946 A1 1/2009 Gotz
2009/0024962 A1 1/2009 Gotz
2009/0031401 A1 1/2009 Cudich et al.
2009/0043801 A1 2/2009 LeClair
2009/0089651 A1 4/2009 Herberger et al.
2009/0103442 A1 4/2009 Douville
2009/0106178 A1 4/2009 Chu
2009/0112678 A1 4/2009 Luzardo
2009/0112745 A1 4/2009 Stefanescu
2009/0150868 A1 6/2009 Chakra et al.
2009/0164934 A1 6/2009 Bhattiprolu et al.
2009/0172821 A1 7/2009 Daira et al.
2009/0177962 A1 7/2009 Gusmorino et al.
2009/0187546 A1 7/2009 Whyte et al.
2009/0199090 A1* 8/2009 Poston et al. 715/255
2009/0199106 A1 8/2009 Jonsson et al.
2009/0216562 A1 8/2009 Faulkner et al.
2009/0228507 A1 9/2009 Jain et al.
2009/0228701 A1 9/2009 Lin
2009/0248757 A1 10/2009 Havewala et al.
2009/0249178 A1 10/2009 Ambrosino et al.
2009/0249244 A1 10/2009 Robinson et al.
2009/0254970 A1 10/2009 Agarwal et al.
2009/0271343 A1 10/2009 Vaiciulis et al.
2009/0281839 A1 11/2009 Lynn et al.
2009/0282068 A1 11/2009 Shockro et al.
2009/0287470 A1 11/2009 Farnsworth et al.
2009/0307049 A1 12/2009 Elliott et al.
2009/0313463 A1 12/2009 Pang et al.
2009/0319891 A1 12/2009 MacKinlay
2009/0328222 A1 12/2009 Helman et al.
2010/0004857 A1 1/2010 Pereira et al.
2010/0011000 A1 1/2010 Chakra et al.
2010/0011282 A1 1/2010 Dollard et al.
2010/0057622 A1 3/2010 Faith et al.
2010/0070842 A1 3/2010 Aymeloglu et al.
2010/0070844 A1 3/2010 Aymeloglu et al.
2010/0076813 A1 3/2010 Ghosh et al.
2010/0077481 A1 3/2010 Polyakov et al.
2010/0098318 A1 4/2010 Anderson
2010/0100963 A1 4/2010 Mahaffey
2010/0122152 A1 5/2010 Chamberlain et al.
2010/0145909 A1 6/2010 Ngo

(56) **References Cited**

U.S. PATENT DOCUMENTS

2010/0169137	A1	7/2010	Jastrebski et al.
2010/0204983	A1	8/2010	Chung et al.
2010/0223260	A1	9/2010	Wu
2010/0235915	A1	9/2010	Memon et al.
2010/0238174	A1	9/2010	Haub et al.
2010/0262688	A1	10/2010	Hussain et al.
2010/0262901	A1	10/2010	DiSalvo
2010/0280851	A1	11/2010	Merkin
2010/0306285	A1	12/2010	Shah et al.
2010/0306722	A1	12/2010	LeHoty et al.
2010/0313119	A1	12/2010	Baldwin et al.
2010/0313239	A1	12/2010	Chakra et al.
2010/0330801	A1	12/2010	Rouh
2011/0010342	A1	1/2011	Chen et al.
2011/0047540	A1	2/2011	Williams et al.
2011/0060910	A1	3/2011	Gormish et al.
2011/0074788	A1	3/2011	Regan et al.
2011/0093327	A1	4/2011	Fordyce et al.
2011/0099133	A1	4/2011	Chang et al.
2011/0107196	A1	5/2011	Foster
2011/0145187	A1	6/2011	Himmelsbach et al.
2011/0161409	A1	6/2011	Nair
2011/0173093	A1	7/2011	Psota et al.
2011/0179048	A1	7/2011	Satlow
2011/0208565	A1	8/2011	Ross et al.
2011/0219450	A1	9/2011	McDougal et al.
2011/0225482	A1	9/2011	Chan et al.
2011/0246229	A1	10/2011	Pacha
2011/0258216	A1	10/2011	Supakkul et al.
2012/0004894	A1	1/2012	Butler
2012/0005159	A1	1/2012	Wang et al.
2012/0016849	A1	1/2012	Garrod et al.
2012/0022945	A1	1/2012	Falkenborg et al.
2012/0023075	A1	1/2012	Pulfer et al.
2012/0036106	A1	2/2012	Desai et al.
2012/0059853	A1	3/2012	Jagota
2012/0065987	A1	3/2012	Farooq et al.
2012/0084117	A1	4/2012	Tavares et al.
2012/0084184	A1	4/2012	Raleigh
2012/0110633	A1	5/2012	An et al.
2012/0110674	A1	5/2012	Belani et al.
2012/0136839	A1	5/2012	Eberlein et al.
2012/0188252	A1	7/2012	Law
2012/0191446	A1	7/2012	Binsztok et al.
2012/0197657	A1	8/2012	Prodanovic
2012/0197660	A1	8/2012	Prodanovic
2012/0210294	A1	8/2012	Gores
2012/0215784	A1	8/2012	King et al.
2012/0221553	A1	8/2012	Wittmer et al.
2012/0226590	A1	9/2012	Love et al.
2012/0254129	A1	10/2012	Wheeler et al.
2012/0266245	A1	10/2012	McDougal et al.
2012/0284670	A1	11/2012	Kashik et al.
2012/0304150	A1	11/2012	Leithead et al.
2012/0304244	A1	11/2012	Xie et al.
2012/0323829	A1	12/2012	Stokes et al.
2013/0006655	A1	1/2013	Van Arkel et al.
2013/0006668	A1	1/2013	Van Arkel et al.
2013/0016106	A1	1/2013	Yip et al.
2013/0067017	A1	3/2013	Carriere et al.
2013/0086482	A1	4/2013	Parsons
2013/0091084	A1	4/2013	Lee
2013/0097482	A1	4/2013	Marantz et al.
2013/0124193	A1	5/2013	Holmberg
2013/0124567	A1	5/2013	Balinsky et al.
2013/0139268	A1	5/2013	An et al.
2013/0151305	A1	6/2013	Akinola et al.
2013/0151453	A1	6/2013	Bhanot et al.
2013/0166480	A1	6/2013	Popescu et al.
2013/0173540	A1	7/2013	Qian et al.
2013/0191336	A1	7/2013	Ducott et al.
2013/0191338	A1	7/2013	Ducott, III et al.
2013/0251233	A1	9/2013	Yang et al.
2013/0262527	A1	10/2013	Hunter et al.
2013/0262528	A1	10/2013	Foit
2013/0263019	A1	10/2013	Castellanos et al.
2013/0275446	A1	10/2013	Jain et al.
2013/0276799	A1	10/2013	Davidson
2013/0288719	A1	10/2013	Alonzo
2013/0346444	A1	12/2013	Makkar et al.
2014/0040182	A1	2/2014	Gilder et al.
2014/0040714	A1	2/2014	Siegel et al.
2014/0059683	A1	2/2014	Ashley
2014/0081652	A1	3/2014	Klindworth
2014/0089339	A1	3/2014	Siddiqui et al.
2014/0114972	A1	4/2014	Ducott et al.
2014/0123279	A1	5/2014	Bishop et al.
2014/0129518	A1	5/2014	Ducott et al.
2014/0129936	A1	5/2014	Richards et al.
2014/0143009	A1	5/2014	Brice et al.
2014/0149130	A1	5/2014	Getchius
2014/0208281	A1	7/2014	Ming
2014/0222793	A1	8/2014	Sadkin et al.
2014/0244284	A1	8/2014	Smith
2014/0358829	A1	12/2014	Hurwitz
2014/0366132	A1	12/2014	Stiansen et al.
2015/0026622	A1	1/2015	Roaldson et al.
2015/0046481	A1	2/2015	Elliot
2015/0073954	A1	3/2015	Braff
2015/0074050	A1	3/2015	Landau et al.
2015/0089353	A1	3/2015	Folkening
2015/0100559	A1	4/2015	Nassar
2015/0100907	A1	4/2015	Erenrich et al.
2015/0106379	A1	4/2015	Elliot et al.
2015/0142766	A1	5/2015	Jain et al.
2015/0186483	A1	7/2015	Tappan et al.
2015/0212663	A1	7/2015	Papale et al.
2015/0235334	A1	8/2015	Wang et al.
2015/0254220	A1	9/2015	Burr et al.
2015/0261847	A1	9/2015	Ducott et al.
2016/0019252	A1	1/2016	Ducott
2016/0062555	A1	3/2016	Ward et al.
2016/0098176	A1	4/2016	Cervelli et al.
2016/0110369	A1	4/2016	Cervelli et al.
2016/0162519	A1	6/2016	Stowe et al.
2017/0068716	A1	3/2017	Richards et al.

FOREIGN PATENT DOCUMENTS

CA	2666364	1/2015
CA	2806954	9/2017
CN	101729531	6/2010
CN	103281301	9/2013
CN	102054015	5/2014
DE	102014204840	9/2014
DE	102014215621	2/2015
EP	0816968	1/1996
EP	1647908	4/2006
EP	1 672 527	6/2006
EP	2778913	9/2014
EP	2778914	9/2014
EP	2778986	9/2014
EP	2911078	8/2015
EP	2993595	3/2016
EP	3002691	4/2016
EP	3009943	4/2016
EP	3032441	6/2016
GB	2366498	3/2002
GB	2513007	10/2014
GB	2518745	4/2015
NL	2013306	2/2015
WO	WO 01/025906	4/2001
WO	WO 2001/088750	11/2001
WO	WO 2003/060751	7/2003
WO	WO 2007/133206	11/2007
WO	WO 2008/064207	5/2008
WO	WO 2008/113059	9/2008
WO	WO 2010/030913	3/2010
WO	WO 2010/030914	3/2010
WO	WO 2011/071833	6/2011
WO	WO 2011/161565	12/2011

(56) **References Cited**

FOREIGN PATENT DOCUMENTS

WO WO 2012/009397 1/2012
WO WO 2012/119008 9/2012

OTHER PUBLICATIONS

U.S. Appl. No. 14/675,716, filed Mar. 31, 2015, Final Office Action, dated Dec. 24, 2015.
U.S. Appl. No. 14/076,385, filed Nov. 11, 2013, Final Office Action, dated Jan. 25, 2016.
U.S. Appl. No. 13/657,684, filed Oct. 22, 2012, Office Action, dated Aug. 25, 2014.
U.S. Appl. No. 14/156,208, filed Jan. 15, 2014, Office Action, dated Mar. 9, 2015.
U.S. Appl. No. 14/156,208, filed Jan. 15, 2014, Notice of Allowance, dated Feb. 12, 2016.
U.S. Appl. No. 14/156,208, filed Jan. 15, 2014, Interview Summary, dated Sep. 17, 215.
U.S. Appl. No. 14/334,232, filed Jul. 17, 2014, Notice of Allowance, dated Nov. 10, 2015.
U.S. Appl. No. 12/836,801, filed Jul. 15, 2010, Notice of Allowance, dated Apr. 16, 2013.
U.S. Appl. No. 13/076,804, filed Mar. 31, 2011, Notice of Allowance, dated Aug. 26, 2013.
U.S. Appl. No. 13/076,804, field Mar. 31, 2011, Advisory Action, dated Jun. 20, 2013.
U.S. Appl. No. 13/355,726, filed Jan. 23, 2012, Notice of Allowance, dated Apr. 28, 2014.
U.S. Appl. No. 13/355,726, filed Jan. 23, 2012, Office Action, dated Mar. 25, 2014.
U.S. Appl. No. 13/686,750, filed Nov. 27, 2012, Office Action, dated Mar. 13, 2013.
U.S. Appl. No. 14/156,208, filed Jan. 15, 2015, Office Action, dated Mar. 9, 2015.
U.S. Appl. No. 14/286,485, filed May 23, 2014, Notice of Allowance, dated Jul. 29, 2015.
U.S. Appl. No. 14/286,485, filed May 23, 2014, Pre-Interview Office Action, dated Mar. 12, 2015.
U.S. Appl. No. 13/076,804, filed Mar. 31, 2011, Final Office Action, dated Apr. 12, 2013.
U.S. Appl. No. 14/156,208, filed Jan. 15, 2015, Final Office Action, dated Aug. 11, 2015.
U.S. Appl. No. 14/473,860, filed Aug. 9, 2014, Notice of Allowance, dated Jan. 5, 2015.
U.S. Appl. No. 13/657,684, filed Oct. 22, 2012, Notice of Allowance, dated Mar. 2, 2015.
U.S. Appl. No. 13/657,684, filed Oct. 22, 2012, Office Action, dated Aug. 28, 2014.
U.S. Appl. No. 14/076,385, filed Nov. 11, 2013, Final Office Action, dated Jan. 22, 2015.
U.S. Appl. No. 14/518,757, filed Oct. 20, 2014, First Office Action Interview, dated Apr. 2, 2015.
U.S. Appl. No. 14/518,757, filed Oct. 20, 2014, Final Office Action, dated Jul. 20, 2015.
U.S. Appl. No. 14/286,485, filed May 23, 2014, First Office Action Interview, dated Apr. 30, 2015.
U.S. Appl. No. 14/076,385, filed Nov. 11, 2013, Office Action, dated Jun. 2, 2015.
U.S. Appl. No. 14/334,232, filed Jul. 17, 2015, Office Action, dated Jul. 10, 2015.
Dell Latitude D600 2003, Dell Inc., http://www.dell.com/downloads/global/products/latit/en/spec_latit_d600_en.pdf.
Dou et al., "Ontology Translaation on the Semantic Web 2005," Springer-Verlag, Journal on Data Semantics II Lecture Notes in Computer Science, vol. 3350, pp. 35-37.
Fidge, Colin J., "Timestamps in Message-Passing Systems," K. Raymond (Ed.) Proc. of the 11th Australian Computer Science Conference (ACSC 1988), pp. 56-66.
Holliday, JoAnne, "Replicated Database Recovery using Multicast Communication," IEEE 2002, pp. 11.
Lamport, "Time, Clocks and the Ordering of Events in a Distributed System," Communications of the ACM, Jul. 1978, vol. 21, No. 7, pp. 558-565.
Loeliger, Jon, "Version Control with Git," O'Reilly, May 2009, pp. 330.
Mattern, F. "Virtual Time and Global States of Distributed Systems," Cosnard, M., Proc. Workshop on Parallel and Distributed Algorithms, Chateau de Bonas, France:Elsevier, 1989, pp. 215-226.
O'Sullivan, Bryan, "Making Sense of Revision Control Systems," Communications of the ACM, Sep. 2009, vol. 52, No. 9, pp. 57-62.
OWL Web Ontology Language Reference Feb 04, W3C, http://www.w3.org/TR/owl-ref/.
Parker, Jr. et al., "Detection of Mutual Inconsistency in Distributed Systems," IEEE Transactions in Software Engineering, May 1983, vol. SE-9, No. 3, pp. 241-247.
Claims for European Patent Application No. 13152370.6 dated Jun. 2013, 5 pages.
Claims for Australian Patent Application No. 2012238282 dated Jan. 2014, 5 pages.
Claims for Australian Patent Application No. 2012238282 dated Jun. 2014, 4 pages.
Claims for International Patent Application No. PCT/US2011/043794 dated Jan. 2013, 6 pages.
Claims for International Patent Application No. PCT/US2011/043794 dated Feb. 2012, 6 pages.
Claims for Canadian Patent Application No. 2666364 dated Oct. 2013, 7 pages.
Official Communication for European Patent Application No. 13152370.6 dated Jun. 3, 2013.
Official Communication for Canadian Patent Application No. 2666364 dated Oct. 3, 2013.
International Search Report & Written Opinion for Patent Application No. PCT/US2011/043794 dated Feb. 24, 2012.
Official Communication for Australian Patent Application No. 2012238282 dated Jun. 6, 2014.
Official Communication for Australian Patent Application No. 2012238282 dated Jan. 30, 2014.
Written Opinion and Search Report for International Patent Application No. PCT/US2011/043794 dated Jan. 24, 2013.
Symantec Corporation, "E-Security Begins with Sound Security Policies," Announcement Policies, Jun. 14, 2001.
Official Communication for European Patent Application No. 15156004.2 dated Aug. 24, 2015.
Official Communication for European Patent Application No. 15155845.9 dated Oct. 6, 2015.
Official Communication for Canadian Patent Application No. 2806954 dated Jan. 15, 2016.
Official Communication for European Patent Application No. 14159175.0 dated Feb. 4, 2016.
Abbey, Kristen, "Review of Google Docs," May 1, 2007, pp. 2.
Klemmer et al., "Where Do Web Sites Come From? Capturing and Interacting with Design History," Association for Computing Machinery, CHI 2002, Apr. 20-25, 2002, Minneapolis, MN, pp. 8.
Altmanninger et al., "A Categorization for Conflicts in Model Versioning," Elektrotechnik & Informationstechnik (2011), 128/11-12: 421-426.
Official Communication for European Patent Application No. 15190307.7 dated Feb. 19, 2016.
Official Communication for European Patent Application No. 15188106.7 dated Feb. 3, 2016.
Official Communication for Australian Patent Application No. 2014201506 dated Feb. 27, 2015.
Palantir, "Extracting and Transforming Data with Kite," Palantir Technologies, Inc., Copyright 2010, pp. 38.
Official Communication for Netherlands Patent Application No. 2012438 dated Sep. 21, 2015.
SnagIt, "SnagIt Online Help Guide," <http://download.techsmith.com/snagit/docs/onlinehelp/enu/snagit_help.pdf>, TechSmith Corp., Version 8.1, printed Feb. 7, 2007, pp. 284.
"GrabUp—What a Timesaver!" <http://atlchris.com/191/grabup/>, Aug. 11, 2008, pp. 3.

(56) **References Cited**

OTHER PUBLICATIONS

Palermo, Christopher J., "Memorandum," [Disclosure relating U.S. Appl. No. 13/916,447, filed Jun. 12, 2013, and related applications], Jan. 31, 2014 in 3 pages.
Microsoft, "Registering an Application to a URI Scheme," <http://msdn.microsoft.com/en-us/library/aa767914.aspx>, printed Apr. 4, 2009 in 4 pages.
Official Communication for New Zealand Patent Application No. 622497 dated Jun. 19, 2014.
Delicious, <http://delicious.com/> as printed May 15, 2014 in 1 page.
Kwout, <http://web.archive.org/web/20080905132448/http://www.kwout.com/> Sep. 5, 2008, pp. 2.
Schroder, Stan, "15 Ways To Create Website Screenshots," <http://mashable.com/2007/08/24/web-screenshots/>, Aug. 24, 2007, pp. 2.
Glaab et al., "EnrichNet: Network-Based Gene Set Enrichment Analysis," Bioinformatics 28.18 (2012): pp. i451-i457.
Official Communication for New Zealand Patent Application No. 622404 dated Mar. 20, 2014.
Conner, Nancy, "Google Apps: The Missing Manual," May 1, 2008, pp. 15.
FireEye, <http://www.fireeye.com/> Printed Jun. 30, 2014 in 2 pages.
Official Communication for New Zealand Patent Application No. 622473 dated Jun. 19, 2014.
Geiger, Jonathan G., "Data Quality Management, the Most Critical Initiative You Can Implement", Data Warehousing, Management and Quality, Paper 098-29, SUGI 29, Intelligent Solutions, Inc., Bounder, CO, pp. 14, accessed Oct. 3, 2013.
Baker et al., "The Development of a Common Enumeration of Vulnerabilities and Exposures," Presented at the Second International Workshop on Recent Advances in Intrusion Detection, Sep. 7-9, 1999, pp. 35.
Johnson, Maggie, "Introduction to YACC and Bison".
Microsoft Windows, "Microsoft Windows Version 2002 Print Out 2," 2002, pp. 1-6.
SnagIt, "SnagIt 8.1.0 Print Out," Software release date Jun. 15, 2006, pp. 6.
Gu et al., "Record Linkage: Current Practice and Future Directions," Jan. 15, 2004, pp. 32.
Palantir, "Kite," https://docs.palantir.com/gotham/3.11.1.0/adminreference/datasources.11 printed Aug. 30, 2013 in 2 pages.
Official Communication for Netherlands Patent Application No. 2011729 dated Aug. 13, 2015.
Official Communication for Great Britain Patent Application No. 1413935.6 dated Jan. 27, 2015.
Wang et al., "Research on a Clustering Data De-Duplication Mechanism Based on Bloom Filter," IEEE 2010, 5 pages.
Hur et al., "SciMiner: web-based literature mining tool for target identification and functional enrichment analysis," Bioinformatics 25.6 (2009): pp. 838-840.
Nitro, "Trick: How to Capture a Screenshot As PDF, Annotate, Then Share It," <http://blog.nitropdf.com/2008/03/04/trick-how-to-capture-a-screenshot-as-pdf-annotate-it-then-share/>, Mar. 4, 2008, pp. 2.
Nivas, Tuli, "Test Harness and Script Design Principles for Automated Testing of non-GUI or Web Based Applications," Performance Lab, Jun. 2011, pp. 30-37.
Official Communication for Israel Patent Application No. 198253 dated Nov. 24, 2014.
Hua et al., "A Multi-attribute Data Structure with Parallel Bloom Filters for Network Services", HiPC 2006, LNCS 4297, pp. 277-288, 2006.
Lee et al., "A Data Mining and CIDF Based Approach for Detecting Novel and Distributed Intrusions," Lecture Notes in Computer Science, vol. 1907 Nov. 11, 2000, pp. 49-65.
Morrison et al., "Converting Users to Testers: An Alternative Approach to Load Test Script Creation, Parameterization and Data Corellation," CCSC: Southeastern Conference, JCSC 28, 2, Dec. 2012, pp. 188-196.
Waters et al., "Building an Encrypted and Searchable Audit Log," Published Jan. 9, 2004, 11 pages, http://www.parc.com/content/attachments/building_encrypted_searchable_5059_parc.pdf.
Schneier et al., "Cryptographic Support for Secure Logs on Untrusted Machines," The Seventh USENIX Security Symposium Proceedings, USENIX Press, Jan. 1998, pp. 53-62, https://www.schneier.com/paper-secure-logs.pdf.
Official Communication for European Patent Application No. 12181585.6 dated Sep. 4, 2015.
Galliford, Miles, "SnagIt Versus Free Screen Capture Software: Critical Tools for Website Owners," <http://www.subhub.com/articles/free-screen-capture-software>, Mar. 27, 2008, pp. 11.
Crosby et al., "Efficient Data Structures for Tamper-Evident Logging," Department of Computer Science, Rice University, 2009, pp. 17.
Ferreira et al., "A Scheme for Analyzing Electronic Payment Systems," Basil 1997.
Official Communication for Canadian Patent Application No. 2831660 dated Jun. 9, 2015.
Online Tech Tips, "Clip2Net—Share files, folders and screenshots easily," <http://www.online-tech-tips.com/free-software-downloads/share-files-folders-screenshots/>, Apr. 2, 2008, pp. 5.
Official Communication for Australian Patent Application No. 2014201507 dated Feb. 27, 2015.
FireEye—Products and Solutions Overview, <http://www.fireeye.com/products-and-solutions> Printed Jun. 30, 2014 in 3 pages.
Official Communication for New Zealand Patent Application No. 622414 dated Mar. 24, 2014.
Official Communication for Netherlands Patent Application No. 2013306 dated Apr. 24, 2015.
VirusTotal—About, <http://www.virustotal.com/en/about/> Printed Jun. 30, 2014 in 8 pages.
Palantir, "Kite Data-Integration Process Overview," Palantir Technologies, Inc., Copyright 2010, pp. 48.
Official Communication for European Patent Application No. 14158958.0 dated Apr. 16, 2015.
Chaudhuri et al., "An Overview of Business Intelligence Technology," Communications of the ACM, Aug. 2011, vol. 54, No. 8.
Official Communication for New Zealand Patent Application No. 628161 dated Aug. 25, 2014.
Official Communication for European Patent Application No. 14189344.6 dated Feb. 29, 2016.
Palantir, "The Repository Element," https://docs.palantir.com/gotham/3.11.1.0/dataguide/kite_config_file.04 printed Aug. 30, 2013 in 2 pages.
Official Communication for New Zealand Patent Application No. 622513 dated Apr. 3, 2014.
Official Communication for Great Britain Patent Application No. 1404486.1 dated Aug. 27, 2014.
Palantir, "Kite Operations," Palantir Technologies, Inc., Copyright 2010, p. 1.
Official Communication for European Patent Application No. 14158977.0 dated Apr. 16, 2015.
Official Communication for Great Britain Patent Application No. 1404479.6 dated Aug. 12, 2014.
Official Communication for Great Britain Patent Application No. 1404499.4 dated Aug. 20, 2014.
JetScreenshot.com, "Share Screenshots via Internet in Seconds," <http://web.archive.org/web/20130807164204/http://www.jetscreenshot.com/>, Aug. 7, 2013, pp. 1.
Kokossi et al., "D7-Dynamic Ontoloty Management System (Design)," Information Societies Technology Programme, Jan. 10, 2002, pp. 1-27.
"Remove a Published Document or Blog Post," Sharing and Collaborating on Blog Post.
Bluttman et al., "Excel Formulas and Functions for Dummies," 2005, Wiley Publishing, Inc., pp. 280, 284-286.
Official Communication for Great Britain Patent Application No. 1404489.5 dated Aug. 27, 2014.
Schneier et al., "Automatic Event Stream Notarization Using Digital Signatures," Security Protocols, International Workshop Apr. 1996 Proceedings, Springer-Verlag, 1997, pp. 155-169, https://schneier.com/paper-event-stream.pdf.

(56) **References Cited**

OTHER PUBLICATIONS

Niepert et al., "A Dynamic Ontology for a Dynamic Reference Work", Joint Conference on Digital Libraries, Jun. 17_22, 2007, Vancouver, British Columbia, Canada, pp. 1-10.
Zheng et al., "Goeast: a web-based software toolkit for Gene Ontology enrichment analysis," Nucleic acids research 36.suppl 2 (2008): pp. W385-W363.
Palantir, https://docs.palantir.com/gotham/3.11.1.0/dataguide/baggage/KiteSchema.xsd printed Apr. 4, 2014 in 4 pages.
Warren, Christina, "TUAW Faceoff: Screenshot apps on the firing line," <http://www.tuaw.com/2008/05/05/tuaw-faceoff-screenshot-apps-on-the-firing-line/>, May 5, 2008, pp. 11.
Official Communication for New Zealand Patent Application No. 622473 dated Mar. 27, 2014.
Palantir, "Write a Kite Configuration File in Eclipse," Palantir Technologies, Inc., Copyright 2010, pp. 2.
Official Communication for Australian Patent Application No. 2013251186 dated Mar. 12, 2015.
Official Communication for New Zealand Patent Application No. 622497 dated Mar. 26, 2014.
Official Communication for European Patent Application No. 14158977.0 dated Jun. 10, 2014.
Anonymous, "BackTult _ JD Edwards One World Version Control System," printed Jul. 23, 2007 in 1 page.
Ma et al., "A New Approach to Secure Logging," ACM Transactions on Storage, vol. 5, No. 1, Article 2, Published Mar. 2009, 21 pages.
Wollrath et al., "A Distributed Object Model for the Java System," Conference on Object-Oriented Technologies and Systems, Jun. 17-21, 1996, pp. 219-231.
Miklau et al., "Securing History: Privacy and Accountability in Database Systems," 3 rd Biennial Conference on Innovative Data Systems Research (CIDR), Jan. 7-10, 2007, Asilomar, California, pp. 387-396.
Notice of Acceptance for Australian Patent Application No. 2013251186 dated Nov. 6, 2015.
Microsoft, "Using the Clipboard," <http://msdn.microsoft.com/en-us/library/ms649016.aspx>, printed Jun. 8, 2009 in 20 pages.
Official Communication for Canadian Patent Application No. 2666364 dated Jun. 4, 2012.
SnagIt, "SnagIt 8.1.0 Print Out 2," Software release date Jun. 15, 2006, pp. 1-3.
Official Communication for New Zealand Patent Application No. 622389 dated Mar. 20, 2014.
O'Reilly.com, http://oreilly.com/digitalmedia/2006/01/01/mac-os-x-screenshot-secrets.html published Jan. 1, 2006 in 10 pages.
Official Communication for European Patent Application No. 14159629.6 dated Jul. 31, 2014.
"A Tour of Pinboard," <http://pinboard.in/tour> as printed May 15, 2014 in 6 page.
Kahan et al., "Annotea: an Open RDF Infrastructure for Shared Web Annotations", Computer Networks, Elsevier Science Publishers B.V., vol. 39, No. 5, dated Aug. 5, 2002.
Official Communication for New Zealand Patent Application No. 622484 dated Apr. 2, 2014.
European Claims application No. 11807426.9-1951, dated Nov. 2016, 7 pages.
European Patent Office, "Search Report" in application No. 11807426.9-1951, dated Nov. 15, 2016, 8 pages.
Parker et al., "Detection of Mutual Inconsistency in Distributed System", IEEE, vol. SE-9, No. 3, dated May 1, 1983, 8 pages.
Saito et al., "Optimistic Replication" Technical Report, dated Sep. 2003, 52, pages.
Parker Jr. et al., "Detection of Mutual Inconsistency in Distributed Systems", IEEE vol. SE-9, No. 3, dated May 1983, 8 pages.
Official Communication for Canadian Patent Application No. 2,826,905 dated Oct. 17, 2016.
Anonymous, "Record Linkage—Wikipedia", dated Apr. 26, 2011, https://en.wikipedia.org/w/index.php?title=Record_linkage&oldid=426069016, 5 pages.
European Patent Office, Search Opinion, Application No. EP-13 152 370.6, dated Jun. 3, 2013, 8 pages.
European Claims in application No. EP-13 152 370.6, dated Jun. 2013, 5 pages.
European Patent Office, "Search Report" in application No. 11 807 426.9-1217, dated May 16, 2018, 7 pages.
European Claims in application No. 11 807 426.9-1217, dated May 2018, 7 pages.
IP Australia, AU Patent Examination Report, Application No. 2012/238282, dated Jun. 6, 2014.
IP Australia, AU Patent Examination Report, Application No. 2012/2838282, dated Jan. 30, 2014.
O'Sullivan B, et al., "Making Sense of Revision-Control Systems", Communications of the ACM, vol. 52, No. 9, dated Sep. 2009, pp. 57-62.
Ries et al., "Locking Granularity Revisited", ACM Transactions on Database Systems, ACM, New York, NY, US vol. 4, No. 2, dated Jun. 1, 1979, 18 pages.
Reiher, Peter et al., "Resolving File Conflicts in the Ficus File System", USENIX, The Advanced Computing Systems Association, dated Aug. 17, 1995, pp. 1-13.
Ratner, David, "Selective Replication: Fine-Grain Control of Replicated Files", dated 1995, 97 pages.
Notification of Transmittal of the International Search Report and the Written Opinion of the International Searching Authority, or the Declaration received in Application No. PCT/US11/43794 dated Feb. 24, 2012 (9 pages).
Current Claims of PCT Application No. PCT/US11/43794 dated Feb. 2012 (6 pages).
The International Bureau of WIPO Switzerland, "Written Opinion and Search Report", in application No. PCT/US2011/043794 dated Jan. 24, 2013, 5 pages.
Current Claims in application No. PCT/US2011/043794 dated Jan. 2013, 6 pages.
U.S. Appl. No. 13/076,804, filed Mar. 31, 2011, Final Office Action.
U.S. Appl. No. 13/686,750, filed Nov. 27, 2012, Notice of Allowance.
European Search Report, EP Application No. 13152370.6-1951, dated Jun. 3, 2013, 8 pages.
Claims from EP Application No. 13152370.6, dated Jun. 2013, 5 pages.
D. Scott Parker, Jr. et al., "Detection of Mutual Inconsistency in Distributed Systems" IEEE Transactions in Software Engineering, XP 000654801, May 1993, 8 pages.

* cited by examiner

FIG. 1

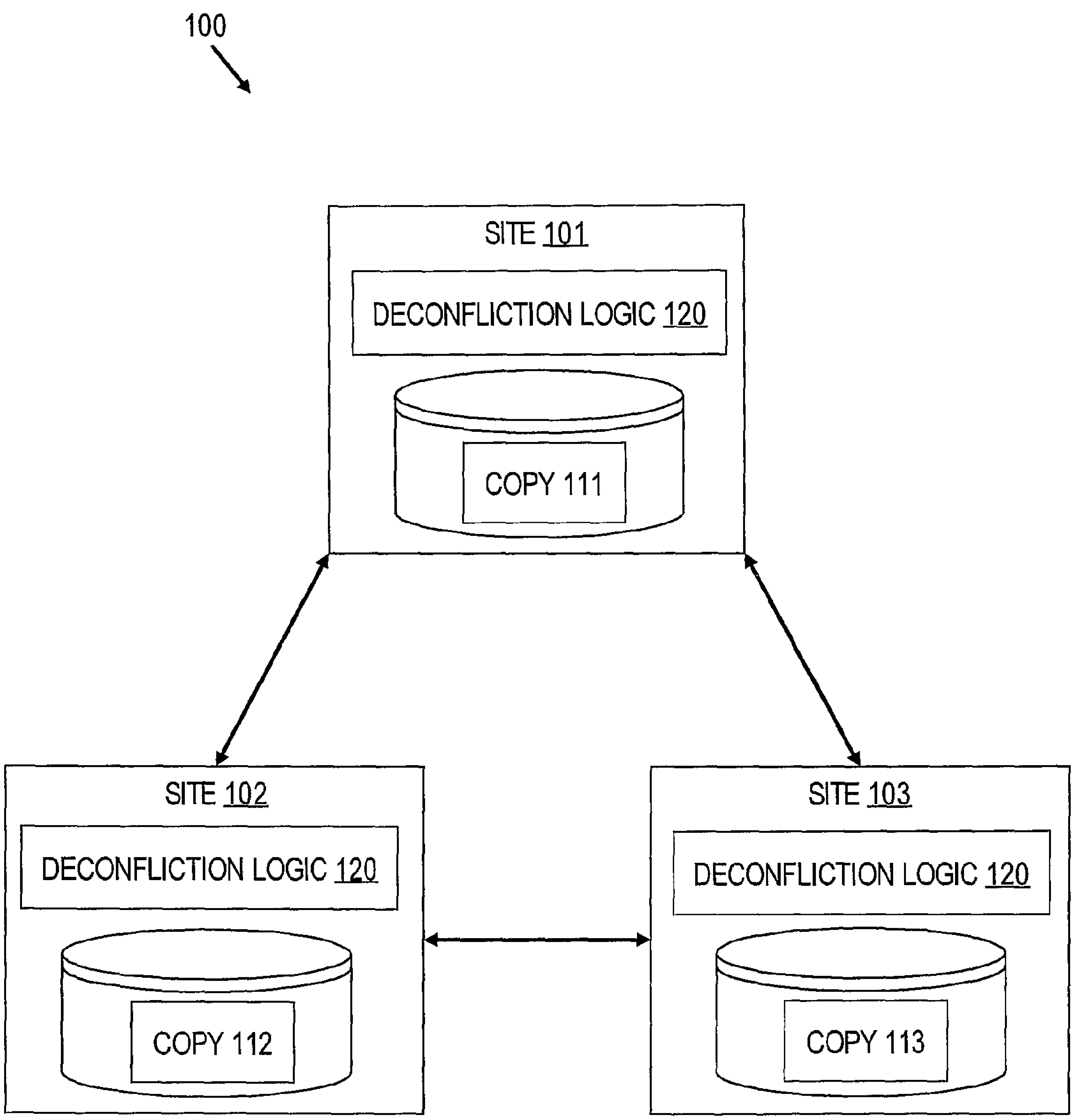

100
SITE 101
DECONFLICTION LOGIC 120
COPY 111
SITE 102
DECONFLICTION LOGIC 120
COPY 112
SITE 103
DECONFLICTION LOGIC 120
COPY 113

FIG. 2

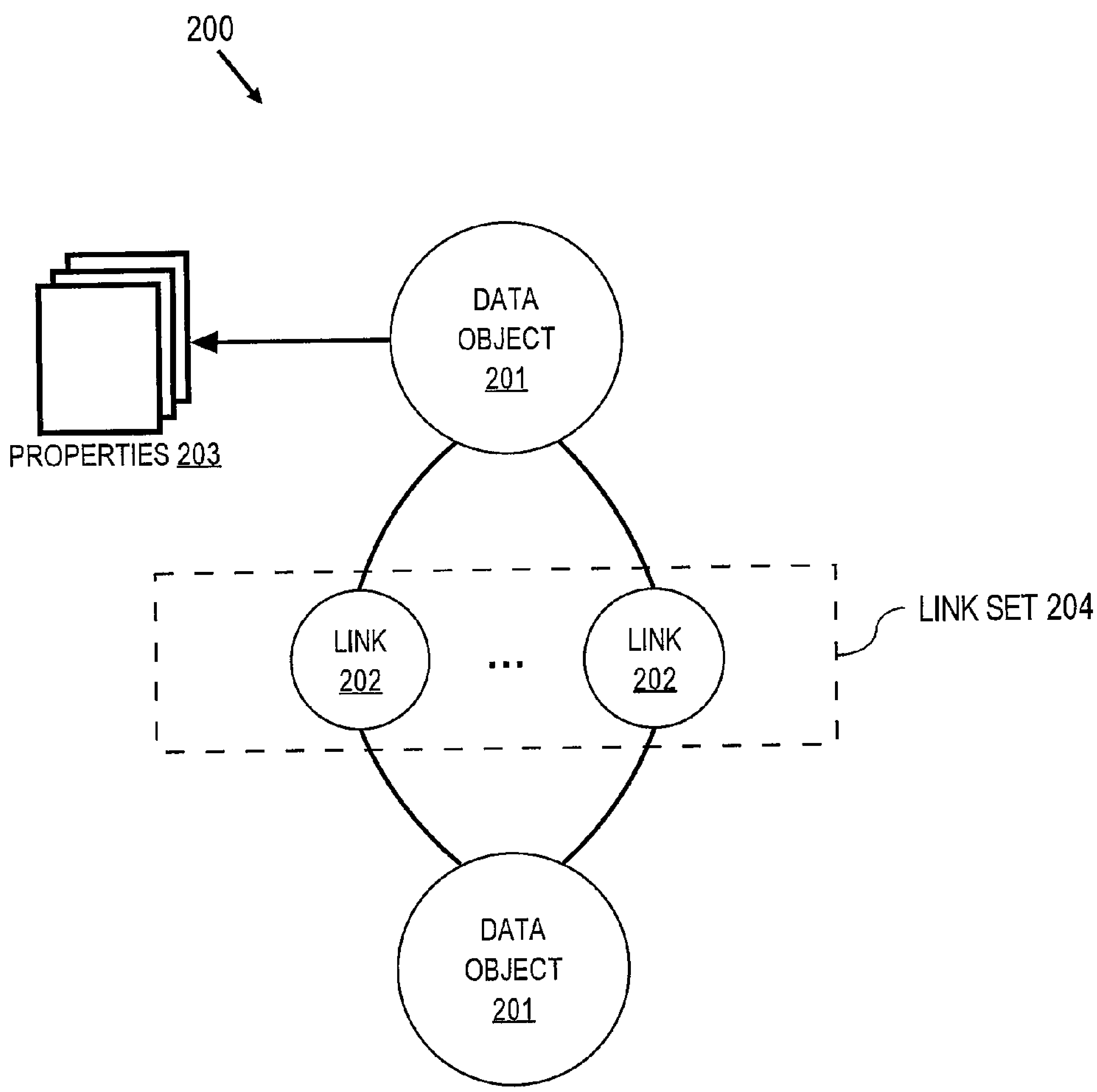

200
DATA
OBJECT
201
PROPERTIES 203
LINK
202
...
LINK
202
LINK SET 204
DATA
OBJECT
201

FIG. 3

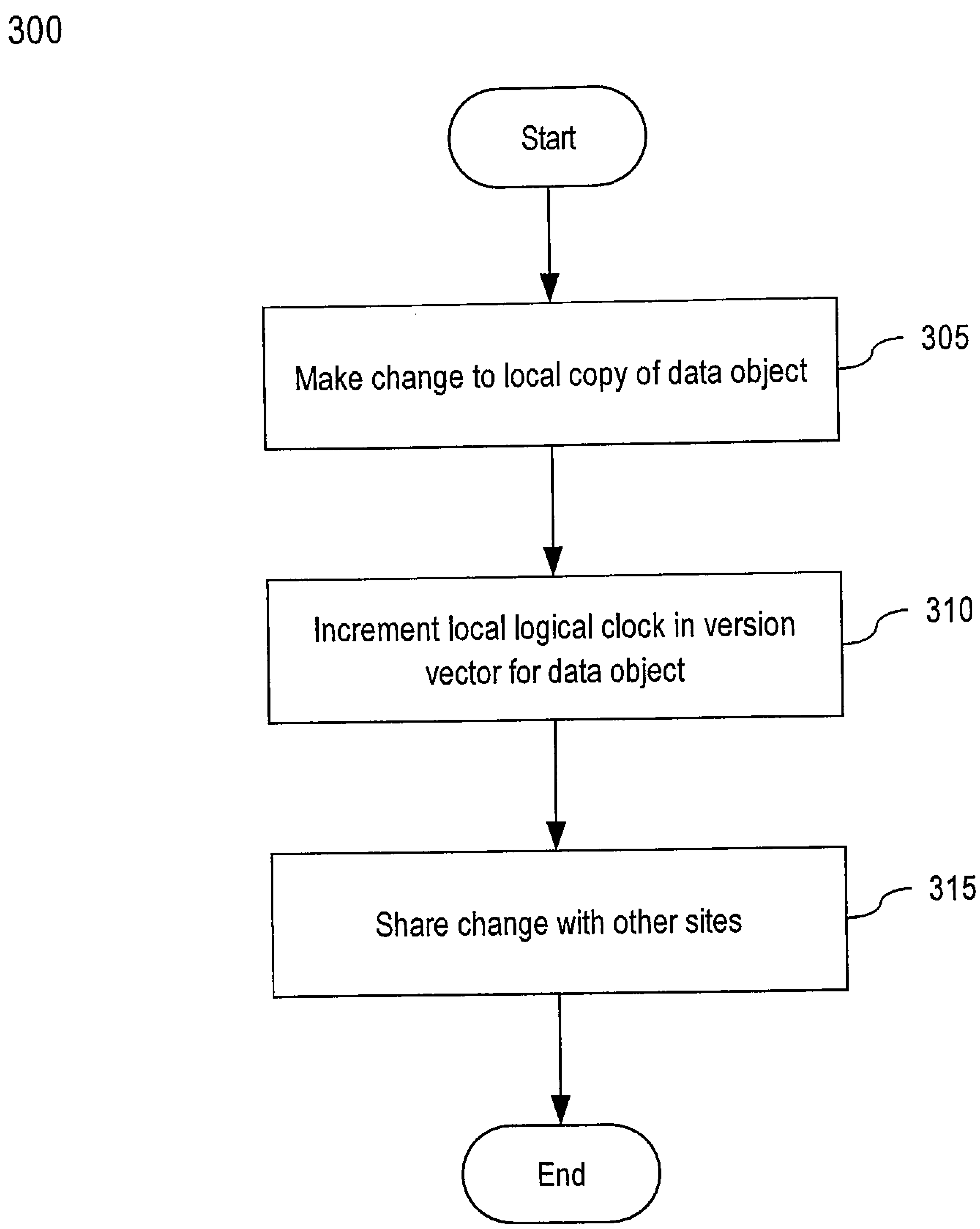

300
Start
Make change to local copy of data object
305
Increment local logical clock in version vector for data object
310
Share change with other sites
315
End

FIG. 4

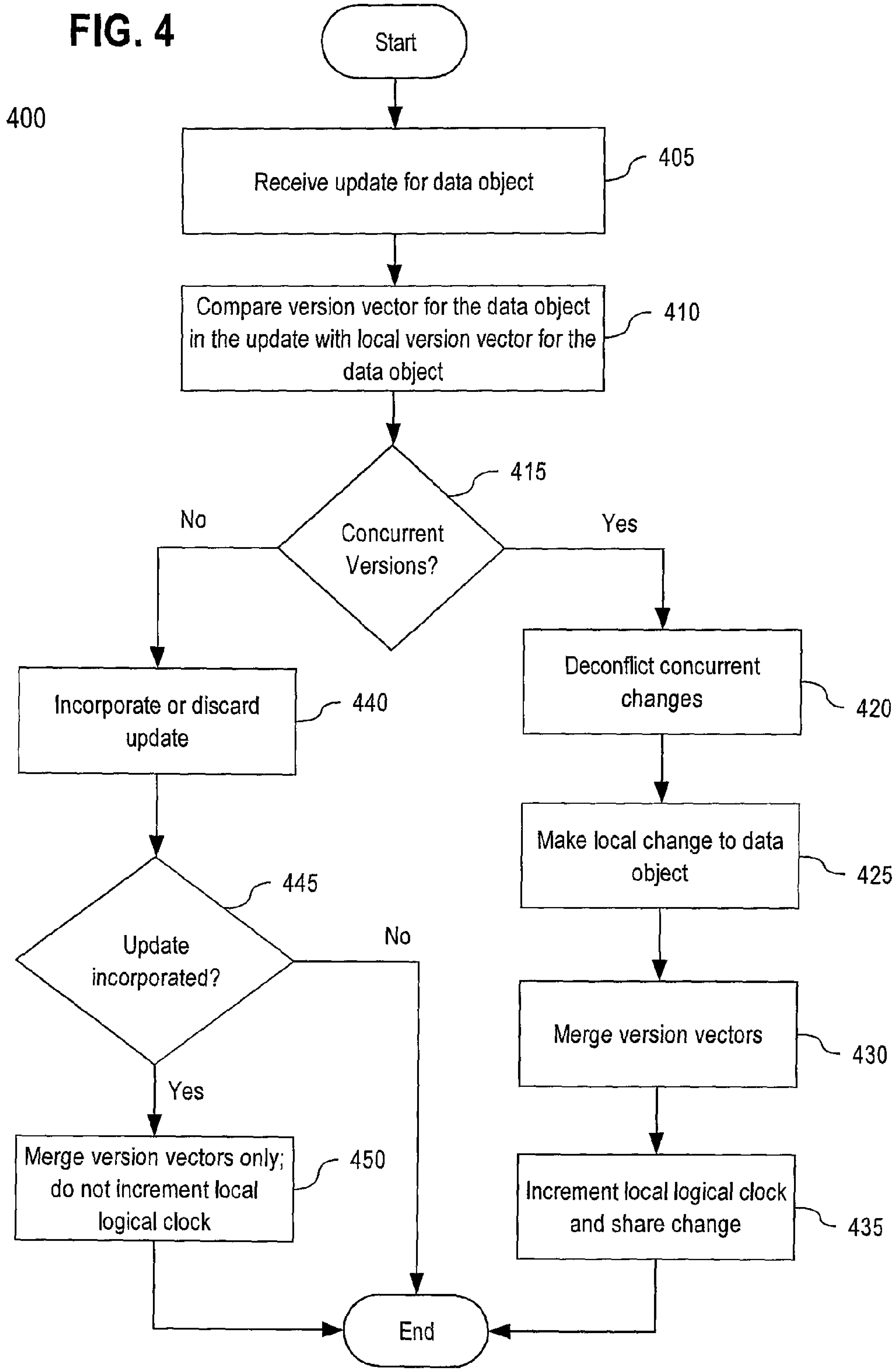

400
Start
Receive update for data object
405
Compare version vector for the data object in the update with local version vector for the data object
410
Concurrent Versions?
415
No
Yes
Deconflict concurrent changes
420
Make local change to data object
425
Merge version vectors
430
Increment local logical clock and share change
435
Incorporate or discard update
440
Update incorporated?
445
No
Yes
Merge version vectors only; do not increment local logical clock
450
End

FIG. 5

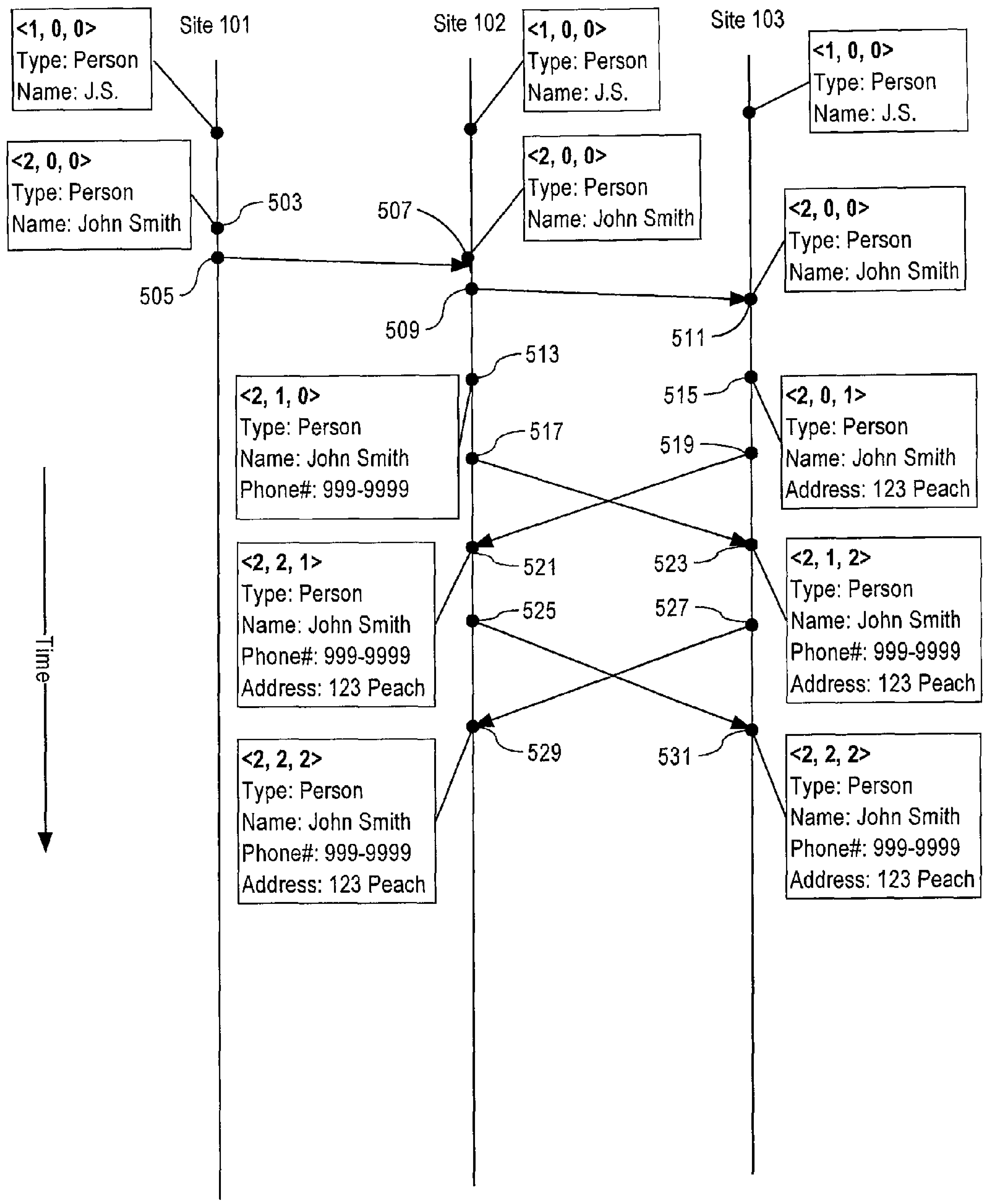
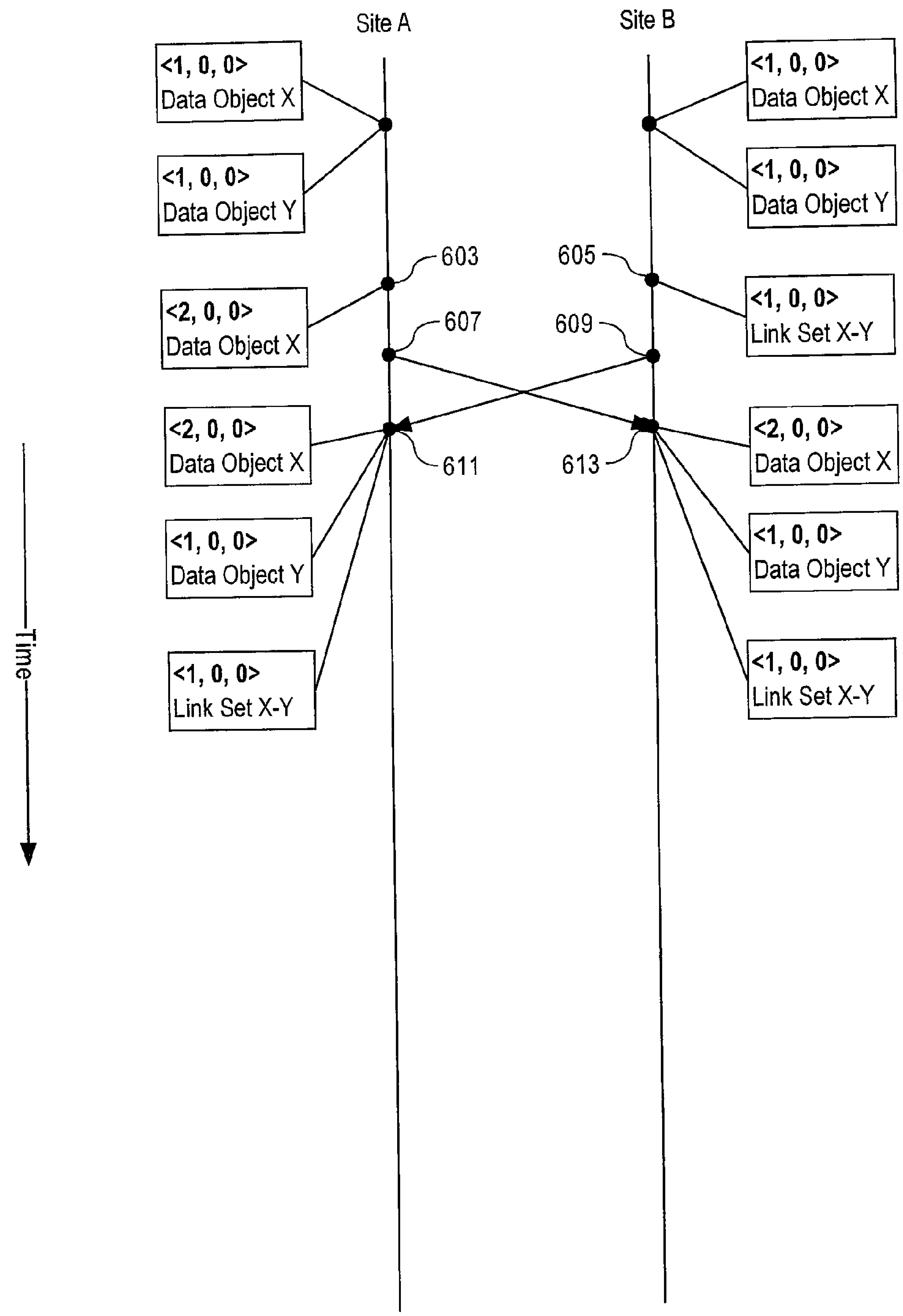

Site 101
Site 102
Site 103
<1, 0, 0>
Type: Person
Name: J.S.
<1, 0, 0>
Type: Person
Name: J.S.
<1, 0, 0>
Type: Person
Name: J.S.
<2, 0, 0>
Type: Person
Name: John Smith
503
507
<2, 0, 0>
Type: Person
Name: John Smith
<2, 0, 0>
Type: Person
Name: John Smith
505
509
511
513
515
<2, 1, 0>
Type: Person
Name: John Smith
Phone#: 999-9999
517
519
<2, 0, 1>
Type: Person
Name: John Smith
Address: 123 Peach
Time
521
523
<2, 2, 1>
Type: Person
Name: John Smith
Phone#: 999-9999
Address: 123 Peach
525
527
<2, 1, 2>
Type: Person
Name: John Smith
Phone#: 999-9999
Address: 123 Peach
529
531
<2, 2, 2>
Type: Person
Name: John Smith
Phone#: 999-9999
Address: 123 Peach
<2, 2, 2>
Type: Person
Name: John Smith
Phone#: 999-9999
Address: 123 Peach

FIG. 6


Site A
Site B
<1, 0, 0> Data Object X
<1, 0, 0> Data Object Y
<1, 0, 0> Data Object X
<1, 0, 0> Data Object Y
603
605
<2, 0, 0> Data Object X
<1, 0, 0> Link Set X-Y
607
609
<2, 0, 0> Data Object X
611
613
<2, 0, 0> Data Object X
<1, 0, 0> Data Object Y
<1, 0, 0> Data Object Y
<1, 0, 0> Link Set X-Y
<1, 0, 0> Link Set X-Y
Time

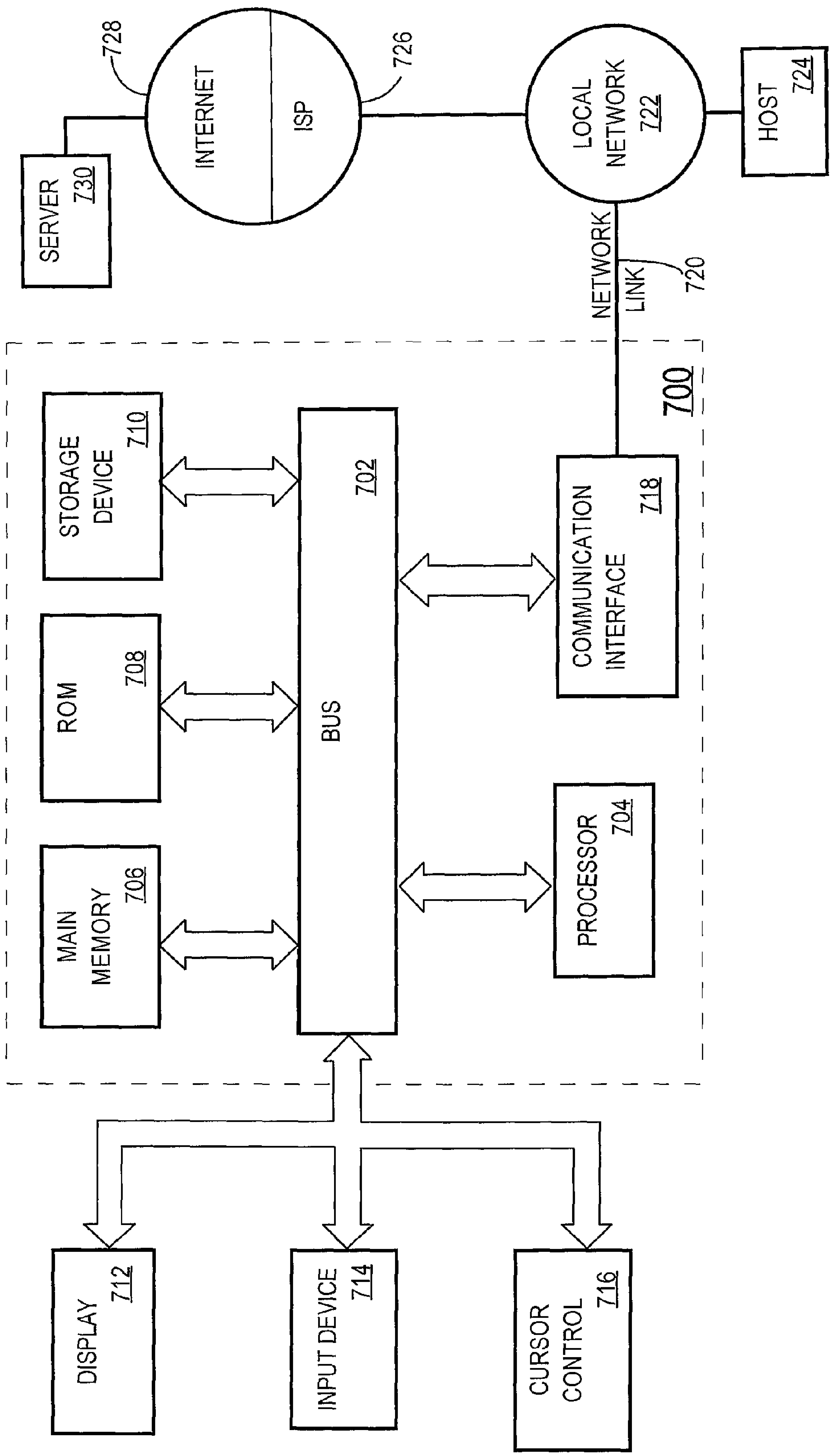

SERVER 730
INTERNET
ISP
728
726
LOCAL NETWORK 722
HOST 724
NETWORK LINK
720
STORAGE DEVICE 710
ROM 708
MAIN MEMORY 706
BUS
702
COMMUNICATION INTERFACE 718
PROCESSOR 704
700
DISPLAY 712
INPUT DEVICE 714
CURSOR CONTROL 716

FIG. 7

SHARING AND DECONFLICTING DATA CHANGES IN A MULTIMASTER DATABASE SYSTEM

Matter enclosed in heavy brackets [ ] appears in the original patent but forms no part of this reissue specification; matter printed in italics indicates the additions made by reissue; a claim printed with strikethrough indicates that the claim was canceled, disclaimed, or held invalid by a prior post-patent action or proceeding.

TECHNICAL FIELD

The present disclosure generally relates to distributed computing systems and, in particular, to sharing and deconfliction of data changes in a multi-master database system.

BACKGROUND

In a multimaster database system, data is stored in a group of databases, data changes may be made to any member of the group, and data changes made to one member are propagated to the rest of the group. Multimaster database systems typically employ either a "synchronous" or an "asynchronous" replication scheme for propagating a change made to one database to the rest of the databases in the group.

In synchronous multimaster replication, each change is applied to all databases in the group immediately or to none of the databases if one or more of the databases in the group cannot accept the change. For example, one of the databases may be offline or unavailable. Synchronous multimaster replication is typically achieved using a two-phase commit protocol.

In contrast, in "asynchronous" multimaster replication, a change made to a database is immediately accepted by the database but propagation of the change to other databases in the group may be deferred. Because propagation of changes may be deferred, if one or more of the databases in the group are unavailable, the available databases can still accept changes, queuing the changes locally until they can be propagated. For this reason, multimaster database systems employing an asynchronous replication strategy are considered to be more highly available than multimaster database systems employing a synchronous replication strategy. However, asynchronous replication raises the possibility of conflicts that occur as a result of concurrent database changes.

A conflict can arise in a multimaster database system when the same data is changed in two different databases before either one of those changes can be propagated to the other. For example, assume that in database A data representing a particular person's eye color is changed to "brown", and after that change but before that change can be propagated to database B data in database B representing the particular person's eye color is changed to "green". Without additional information, it is unclear which change is the "correct" change that should be adopted by all databases in the system.

Multimaster database systems employing an asynchronous replication scheme typically provide mechanisms for "deconflicting" conflicts. As used herein, the term "deconflict", refers generally to detecting and resolving a conflict such that a resolution of the conflict is eventually adopted by all databases in the system. In some cases, the multimaster database system may be able to deconflict automatically without requiring user intervention. In other cases, user intervention may be required to determine which of the concurrent changes should be adopted.

In multimaster database systems employing asynchronous replication, when conflicts are detected has an enormous effect on the integrity of database data. For example, some database systems may support "object resolution". Object resolution involves a user or an automated computing process determining that two or more separate data objects actually represent the same real-world entity and invoking a function of the database system so that the separate data objects are resolved into a single data object. For example, assume there are two separate data objects, one having a name property value of "John Smith", the other having a name property value of "J. S.". A user may decide that these two data objects both represent the same real-world person. Accordingly, in a database system that supports object resolution, the user may invoke a function of the database system so that the two separate data objects are resolved to a single data object having a name property value of "John Smith" or "J.S." as selected by the user resolving the objects together.

In multimaster database systems employing asynchronous replication, it would be desirable to detect as a conflict concurrent changes that include an object resolution change. For example, assume that in database A, User 1 changed the hair color property of a data object representing a person named "J.S." from "brown" to "blonde". Further assume that before the hair color change made by User 1 can be propagated from database A to database B that User 2 changes database B by resolving together the data object representing "J.S." with another data object representing a person named "John Smith". It would be desirable for the multimaster database system to detect these two concurrent changes as a conflict as User 2 may not have decided to resolve "J.S." and "John Smith" together if User 2 had known that John Smith's hair color was changed by User 1. Similarly, User 1 may not have decided to change the hair color of "J.S." had User 1 known that User 2 resolved "J.S." and "John Smith" together.

What is a needed then is a multimaster database system employing asynchronous replication that detects conflicts resulting from concurrent changes in a manner that is in line with user expectations and that handles the deconfliction and propagation of such changes appropriately. Embodiments of the present invention fulfill these and other needs.

The approaches described in this section are approaches that could be pursued, but not necessarily approaches that have been previously conceived or pursued. Therefore, unless otherwise indicated, it should not be assumed that any of the approaches described in this section qualify as prior art merely by virtue of their inclusion in this section.

BRIEF DESCRIPTION OF DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

FIG. **1** illustrates a multimaster database system for use in sharing and deconflicting data changes amongst a plurality of replication sites according to an embodiment of the invention.

FIG. **2** illustrates an object-centric conceptual data model according to an embodiment of the invention.

FIG. **3** illustrates a method for sharing a data change to a data object in a multimaster database system using per-object version vectors, according to an embodiment of the invention.

FIG. **4** illustrates a method for detecting and deconflicting a conflict involving concurrent changes to a data object using per-object version vectors, according to an embodiment of the invention.

FIG. **5** illustrates an example of detecting and deconflicting a conflict involving concurrent changes to a data object using per-object version vectors according to an embodiment of the invention.

FIG. **6** illustrates an example of sharing data changes using per-link set version vectors according to an embodiment of the invention.

FIG. **7** illustrates a computer system with which an embodiment may be implemented.

DETAILED DESCRIPTION

Introduction

Referring to the figures, exemplary embodiments of the invention will now be described. The exemplary embodiments are primarily described with reference to block diagrams or flowcharts. As to the flowcharts, each block within the flowcharts represents both a method step and an apparatus element for performing the method step. Depending upon the implementation, the corresponding apparatus element may be configured in hardware, software, firmware, or combinations thereof.

Further, in the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, that the present invention may be practiced without these specific details. In other instances, block diagrams include well-known structures and devices in order to avoid unnecessarily obscuring the present invention.

Overview

According to one or more embodiments of the present invention, a multimaster database system and computer-based method therein provide sharing and deconfliction of data changes amongst a plurality of replication sites.

In a particular embodiment, data changes at sites to data objects are tracked by each site on a per-data object basis using per-data object version vectors. The method includes a first computing device at a first site making a change to a data object. The first computing device shares the change to the data object with one or more other sites. A second computing device at a second site receives an update reflecting the change to the data object made by the first computing device at the first site. The update includes an identification of the data object, data reflecting the change to the data object, and a version vector for the data object at the first site. The second computing device obtains a version vector for the data object at the second site and compares the version vector of the data object at the first site to the version vector of the data object at the second site to determine whether the two version vectors are identical, ordered, or concurrent. Based on this comparison, the second site either attempts to automatically deconflict the two versions of the data object if, according to their version vectors, they are concurrent, or automatically incorporates the received update into the second site's copy of the data object if, according to their version vectors, the version of the data object at the second site is ordered before the version received in the update.

In another particular embodiment, data changes at sites to links connecting two data objects are tracked on a per-link set basis using per-link set version vectors. The method includes a first computing device at a first site making a change to a set of links connecting two data objects. The first computing device shares the change to the link set with one or more other sites. A second computing device at a second site receives an update reflecting the change to the link set made by the first computing device at the first site. The update includes an identification of the link set and a version vector for the link set at the first site. The second computing device obtains a version vector for the link set at the second site and compares the version vector for link set at the first site to the version vector of the link set at the second site to determine whether the two version vectors are identical, ordered, or concurrent. Based on this comparison, the second site either attempts to automatically deconflict the two versions of the link set if, according to their version vectors, they are concurrent, or automatically incorporates the received update into the second site's copy of the link set if, according to their version vectors, the version of the link set at the second site is ordered before the version received in the update.

In another particular embodiment, per-object version vectors are used to detect a conflict resulting from concurrent changes at two or more sites in which at least one of the concurrent changes includes an object resolution change. The method includes a first computing device at a first site of the plurality of sites resolving two or more data objects together via an object resolution feature of a database system or database application. The first computing device shares the resolution change with one or more other sites of the plurality of sites. A second computing device receives an update reflecting the resolution change made by the first computing device at the first site. The update includes an identification of each of the two or more data objects that were resolved together, and, for each of the two or more data objects, a version vector of the data object at the first site. The second computing device obtains, for each of the two or more data objects, a version vector of the data object at the second site. The second computing device compares, for each of the two or more data objects, the version vector of the data object at the first site to the version vector of the data object at the second site to determine whether the two versions are identical, ordered, or concurrent. In response to the second computing device determining that the version vector of at least one data object of the two or more data objects at the first site is concurrent with the version vector of the at least one data object at the second site, the second computing device determines that the resolution change made by the first computing device at the first site conflicts with the version of the at least one data object at the second site.

Other embodiments include, without limitation, a computer-readable non-transitory medium that includes processor-executable instructions that enable a processing unit to implement one or more aspects of the disclosed methods as well as a system configured to implement one or more aspects of the disclosed methods.

Multimaster Database System with Deconfliction Engine

FIG. **1** illustrates a multimaster database system **100** for use in sharing and deconflicting data changes amongst a plurality of replication sites according to an embodiment of the invention. In one embodiment, sites **101**, **102**, and **103** are coupled through one or more data networks such as the Internet, one or more wide area networks (WANs), one or more local area networks (LANs), one or more network communication buses, or some combination thereof. It is not necessary that a highly or continuously available data network exist between replication sites and the data network(s) connecting any two sites may only be periodically available. In another embodiment, one or more of the sites are not connected to any other site in the system and data is transported to and from these sites manually using portable media or a portable media device as such as a Compact Disc (CD), a Digital Versatile Disc (DVD), Universal Serial Bus (USB) flash device, etc.

Each site **101**, **102**, and **103** may comprise one or more networked computing devices such as one or more workstation computers, server computers, laptop computers, mobile computing devices, or combinations thereof connected to each other via one or more data networks. Further, while only three sites are shown in FIG. **1**, multimaster database system **100** may comprise many hundreds or even many thousands of geographically distributed sites.

According to one embodiment, each site **101**, **102**, and **103** each have copies **111**, **112**, and **113** of the same body of data. The body of data may be, for example, one or more tables in a relational database. However, embodiments of the invention are not limited to relational databases and any type of database capable of supporting the conceptual data model described herein may be used. Non-limiting examples of types of databases capable of supporting the conceptual data model described herein include relational databases, hierarchical databases, and object-oriented databases.

With respect to that particular body of data, site **101** may be configured to asynchronously propagate to site **102** changes made to copy **111**, and asynchronously propagate to site **103** changes made to copy **111**. Similarly, site **102** may be configured to asynchronously propagate to site **101** changes made to copy **112**, and asynchronously propagate to site **103** changes made to copy **212**. Site **103** may be configured to asynchronously propagate to both sites **101** and **102** changes made to copy **113**. However, it is not necessary that each site be configured to propagate to every other site changes made to its copy of the body of data. In other words, a full-meshed multimaster site topology is not required to implement embodiments of the invention and partially-meshed or cascading multimaster topologies may be used.

As system **100** employs an asynchronous replication scheme, each copy **111**, **112**, and **113** of the body of data is loosely consistent with the other copies. That is, each copy may diverge from time to time such that at any given moment one copy's view of the body of data may be different from another copy's view of the body of data. In the absence of new changes, the copies are expected to eventually become consistent with one another. Thus, as well as being loosely consistent with one another, the copies **112**, **112**, **113**, etc. can also be said to be eventually consistent.

Each site **101**, **102**, and **103** has deconfliction logic **120** for receiving remote changes to the body of data from other sites, detecting conflicts, deconflicting detected conflicts either automatically or with user assistance, and sharing local changes to the body of data with other sites. Deconfliction logic **120** may be implemented as one or more computer software programs, one or more field programmable logics, hard-wired logic, or a combination thereof. In one embodiment, deconfliction logic **120** is a software component of a database management system such as those commercially available from the Oracle Corporation of Redwood Shores, Calif. and the Microsoft Corporation of Redmond Wash. In another embodiment, deconfliction logic **120** is software component of a web-based, server-based or desktop application that uses a database management system for performing the deconfliction techniques described herein. In yet another embodiment, deconfliction logic **120** is implemented in part by a web-based, server-based or desktop application and in part by a database management system.

As used herein, the term "change", unless otherwise apparent from the surrounding text, refers to an addition, edit, or deletion to a copy of the body of data at a site. A change can be initiated by a user or a computing process. In addition, a change can also be initiated by deconfliction logic **120** in response to receiving notification of a previous change made at a site different from the site receiving the notification.

As used herein, the term "update", unless otherwise apparent from the surrounding text, refers to information about a change that is sent from the site that made the change to another site. Each change may result in an update being received by every other site so that the other sites can incorporate the change into their respective copies of the body of data. Reception of an update at a site may raise a conflict with the receiving site's copy of the body of data. Techniques implemented by deconfliction logic **120** for detecting and deconflicting conflicts in various scenarios are described in greater detail below.

Object-Centric Data Model

In one embodiment, the body of data, of which each site **101**, **102**, and **103** maintains a copy of, is conceptually structured according to an object-centric data model. It should be understood that this conceptual data model is independent of any particular database data model that may be used for storing a copy of the body of data at a site. For example, each object of the conceptual data model may correspond to one or more rows in a relational database or an entry in Lightweight Directory Access Protocol (LDAP) database.

FIG. **2** illustrates an object-centric conceptual data model **200** according to an embodiment. Model **200** is centered on the notion of a data object **201**. At the highest level of abstraction, data object **201** is a container for information representing things in the world. For example, data object **201** can represent an entity such as a person, a place, an organization, or other noun. Data object **201** can represent an event that happens at a point in time or for a duration. Data object **201** can represent a document or other unstructured data source such as an e-mail message, a news report, or a written paper or article. At a minimum, each data object **201** is associated with a unique identifier that uniquely identifies the data object within system **100**. Each data object **201** may also have a type (e.g., Person, Event, or Document) and a display name which may be the value of a particular property of the data object.

Each data object **201** may have one or more properties **203**. Properties **203** are attributes of the data object **201** that represent individual data items. At a minimum, each property **203** of a data object **201** has a type and a value. Different types of data objects may have different types of properties. For example, a Person data object might have an Eye Color property and an Event object might have a Date property. In one embodiment, the set of data object types and the set of property types for each type of data object supported by the system **100** are defined according to a pre-defined or user-defined ontology or other hierarchical structuring of knowledge through sub-categorization of object types and property types according to their relevant and/or cognitive qualities. In addition, data model **200** may support property multiplicity. In particular, a data object **201** may be allowed to have more than one property **203** of the same type. For example, a Person data object might have multiple Address properties or multiple Name properties.

Each link **202** represents a connection between two data objects **201**. In one embodiment, the connection is either through a relationship, an event, or through matching properties.

A relationship connection may be asymmetrical or symmetrical. For example, Person data object A may be connected to Person data object B by a Child Of relationship (where Person data object B has an asymmetric Parent Of relationship to Person data object A), a Kin Of symmetric relationship to Person data object C, and an asymmetric Member Of relationship to Organization data object X. The type of relationship between two data objects may vary depending on the types of the data objects. For example, Person data object A may have an Appear In relationship with Document data object Y or have a Participate In relationship with Event data object E.

As an example of an event connection, two Person data objects may be connected by an Airline Flight data object representing a particular airline flight if they traveled together on that flight, or by a Meeting data object representing a particular meeting if they both attended that meeting. In one embodiment, when two data objects are connected by an event, they are also connected by relationships, in which each object has a specific relationship to the event, such as, for example, an Appears In relationship.

As an example of a matching properties connection, two Person data objects representing a brother and a sister, may both have an Address property that indicates where they live. If the brother and the sister live in the same home, then their Address properties likely contain similar, if not identical information. In one embodiment, a link between two data objects may be established based on similar or matching properties of the data objects.

The above are just some examples of the types of connections that may be represented by a link and other types of connections may be represented. Thus, it should be understood that embodiments of the invention are not limited to any particular types of connections between data objects. For example, a document might contain two different tagged entities. A link between two data objects may represent a connection between these two entities through their co-occurrence within the same document.

Each data object **201** can have multiple links with another data object **201** to form a link set **204**. For example, two Person data objects representing a husband and a wife could be linked through a Spouse Of relationship, a matching property (Address), and an event (Wedding).

In one embodiment, data model **200** supports object resolution. As mentioned above, object resolution includes a user or an automated computing process determining that two or more separate data objects **201** actually represent the same real-world entity and invoking a function of the system **100** at a site **101**, **102**, **103**, etc. so that the separate data objects appear to users of the system **100** as if they were a single data object. In one embodiment, when one data object **201** is resolved together with another data object **201** the properties and links of one data object are copied to the other data object and then deleted from the data object from which they were copied. However, both data objects are still retained by the system. As well as facilitating the ability to un-resolve data objects that were previously resolved together, retaining data objects after resolving them together facilitates detection and deconfliction of conflicts as described in greater detail below.

Per-Data Object Version Vectors

A version vector is known mechanism for tracking changes in distributed systems. However, version vectors are typically employed on a per-site basis. That is, with typical implementations of version vectors in distributed systems, each site uses a single version vector to track all changes made to the copy of the database maintained by that site.

In accordance with an embodiment of the invention, in order to track and to deconflict changes to the body of data, each site **101**, **102**, **103**, etc. maintains version vectors on a per-data object basis. By doing so, conflicts involving changes to properties of data objects and conflicts involving object resolution changes can be appropriately detected and deconflicted as explained in greater detail below.

In one embodiment, each site maintains one version vector for each data object managed by the system. Thus, for a system having m sites managing n data objects, each site will maintain n version vectors for a total of m*n version vectors maintained by all m sites. Each version vector may contain up to m elements, one for each of the m sites. Each element of a version vector holds a value representing a logical clock for the associated data object at the site corresponding to the element. In a practical embodiment, to conserve data storage space, data maintained at a site representing a version vector may not represent all m elements, but instead some subset of the m elements. For example, elements of a version vector that have a default value may not be represented.

Each site has, in each version vector that the site maintains, its own logical clock value as one of the elements. This logical clock value represents the version of the associated data object at the site maintaining the version vector. Each other element in the version vector represents the site's best guess based on the updates the site has received of the version of the associated data object at the site corresponding to the other element.

In one embodiment, each element of a version vector is set to some initial value (e.g., zero). When a site changes one or more properties of a data object in a database transaction against the site's copy of the body of data, the site increments its own logical clock in the version vector associated with the data object by a fixed value (e.g., one). When sharing the change with other sites as an update, the site includes in the update data representing the change to the data object and data representing the site's version vector for the data object after the increment. A site receiving the update can compare the version vector in the update with its own version vector for the data object to determine whether the version of the data object at the receiving site and the version of the data object in the update are: (1) identical, (2) ordered, or (3) concurrent.

Known techniques for comparing two version vectors to determine whether the two versions are identical, ordered, or concurrent can be used. In one embodiment, comparing two version vectors includes comparing each element in one version vector with the corresponding element in the other version vector. Correspondence between elements is determined based on the site the elements correspond to. In particular, the element for a site in one version vector is compared against the element for the same site in the other version vector. Two versions are identical if each element in one version vector equals the corresponding element in the other version vector. The two versions are ordered if one version "happened before" the other. Version vector A happened before version vector B if each element in version vector B is greater than or equal to the corresponding element in version vector A and at least one element in the version vector B is greater than the corresponding element in version vector A. Similarly, version vector B happened before version vector A if each element in version vector A is greater than or equal to the corresponding element in version vector B and at least one element in the version vector A is greater than the corresponding element in version vector B. Two versions are concurrent if they are neither identical nor ordered.

Sharing Changes to Data Objects Using Per-Object Version Vectors

FIG. **3** illustrates a method **300** for sharing a data change to a data object in a multimaster database system using per-object version vectors, according to an embodiment of the invention. As shown, the method **300** begins at step **305** where a site makes a change to a local copy of a data object stored in the site's copy of the body of data. For example, a user may use a database application at the site to add, delete, or edit one or more properties of the data object.

In one embodiment, as part of changing a data object at a site, each change results in a new version of the data object at the site. At step **310**, the site's local logical clock in the version vector for the data object is incremented by a fixed value (e.g., one) to reflect the new version of the data object at the site where the change was made. The other elements in the version vector are not incremented.

In one embodiment, each change to a data object at a site is shared with every other site in the system. Depending on the topology of the multimaster system (e.g., full-meshed or partially meshed), a site making a change may communicate with every other site to share the change, or just some subset of them that are responsible for communicating the change with other sites. At step **315**, the change made at step **305** is shared with at least one other site in the system. Sharing the change includes sending, to the at least one other site, data that represents the change and data that represents the version vector for the changed data object after the increment at step **310**.

In one embodiment, data that represents the change includes an identifier of the data object and a materialized representation of the data object including all properties of the data object. In another embodiment, data that represents the change includes an identifier of the data object but just the properties of the data object affected by the change. Data that represents the version vector for the changed data object need not include a representation of each element of the version vector and in a practical embodiment, data representing only a subset of all possible elements of the version vector is shared with the at least one other site.

Detecting and Deconflicting Conflicts Involving Changes to Data Objects Using Per-Object Version Vectors

FIG. **4** illustrates a method **400** for detecting and deconflicting a conflict involving concurrent changes to a data object using per-object version vectors, according to an embodiment of the invention. As shown, the method **400** begins at step **405** where a site receives an update for a data object from another site. The update includes data that represents a change to the data object including an identifier of the changed data object and data that represents the version vector for the changed data object. The version vector represents the version of the data object at the site that made the change immediately after the change was made. For clarity of explanation, the version vector for the changed data object received in the update will be referred to as the changing site's version vector for the data object.

At step **410**, the site receiving the update obtains locally its version vector for the data object based on the identifier of the data object included in the update and compares its version vector with the changing site's version vector to determine whether the two versions are identical, ordered, or concurrent. As mentioned above, this comparison includes comparing the changing site's version vector with the receiving site's version vector on an element by element basis.

At step **415**, a determination is made whether the changing site's version for the data object received in the update and the receiving site's version vector for the data object are concurrent. If the two versions are concurrent, then a conflict has been detected. That is, the version of the data object at the receiving site reflects a change to the data object made without knowledge of the change received in the update and the version of the data object received in the update reflects a change to the data object made without knowledge of the change that the receiving site is aware of. If a conflict is detected, then the method **400** proceeds to step **420** where the concurrent changes resulting in the conflict is either automatically or manually deconflicted.

At step **420**, an initial determination is made whether the conflict can be automatically deconflicted. In one embodiment, determining whether a conflict can be automatically deconflicted is based on a set of heuristics and/or deconfliction rules. The set of heuristics and/or deconfliction rules may be user defined. For example, in one embodiment, determining whether a conflict can be automatically deconflicted includes determining whether the concurrent changes involve changes to non-overlapping properties or non-overlapping property types of the data object. For example, if the change received in the update is to a Phone Number property of a particular Person data object and the change the receiving site is aware of is to an Address property of the particular data object, then the system may automatically determine that both changes can accepted. In one embodiment, non-overlapping properties are detected at the receiving site by performing a property by property comparison between the changing site's version of the data object received in the update and the receiving site's version of the data object.

If the conflict cannot be automatically deconflicted, then the receiving site holds the update in a pending update queue for the data object until it can be deconflicted with the **[**aide**]** *aid* of user input. For example, the receiving site may not be able to automatically deconflict a conflict if the concurrent changes involve changes to the same property of a data object. For example, if the change received in the update is to a Phone Number property of a particular Person data object and the change to the data object the receiving site is aware of is also to the Phone Number property of the particular data object, then the receiving site may not be able to automatically resolve the conflict. While an update to a data object remains in the receiving site's pending update queue for the data object, the receiving site can continue to make changes to the data object and accept and apply updates to the data object received from other sites until the user either discards or accepts the update.

In one embodiment, to help a user make an informed deconfliction decision when manually deconflicting a conflict involving concurrent changes to a data object, the deconflicting site determines the greatest common ancestor at the deconflicting site of (a) the version of the data object in the pending update queue at the deconflicting site (pending version) and (b) the current version of the data object at the deconflicting site (current version). The greatest common ancestor of these two versions is determined as the most recent version of the data object at the deconflicting site that is ordered before (i.e., happened before) both (a) the pending version of the data object and (b) the current version of the data object according to their respective version vectors. An application at the deconflicting site uses the greatest common ancestor information to present to a user the differences between both: (1) the greatest common ancestor version of the data object and the pending version and (2) the greatest common ancestor version and the current version. For example, the application may present a graphical user interface that provides a visual indication of the property-wise differences so that a user can understand the nature of the concurrent changes and indicate which version of the data object is correct. Based on presentation of the differences (1) and (2), the user can determine which one of the two versions for the data object is the correct version for the data object and provide an indication through the application of the selected version.

At step **425**, the deconfliction of the concurrent changes in step **420** results in a change to the receiving site's local copy of the data object. The change to the data object reflects the result of the deconfliction. For example, if it was determined in step **420** that the concurrent changes involved non-overlapping properties, then the change made to the data object at step **425** might involve modifying the receiving site's local copy of the data object to incorporate the changed non-overlapping properties received in the update.

After the change is made to the receiving site's local copy of the data object, at step **430**, the changing site's version vector for the data object is merged together with the receiving site's version vector for the data object. Merging the two version vectors includes merging each element in the changing site's version vector for the data object with the corresponding element in the receiving site's version vector for the data object. Merging two elements includes choosing the numerically greater of the two elements as the value of the element in the new version vector. What is produced by this merging at step **430** is a new version vector that is ordered after both the receiving site's version vector for the data object and the changing site's version vector for the data object. Stated otherwise, the receiving site's version vector for the data object and the changing site's version vector now both happened before the new version vector. After the two version vectors are merged, the receiving site's version vector for the data object is replaced with the new version vector which then becomes the version vector for the data object at the receiving site.

Step **435** is similar to a combination of steps **310** and **315** of method **300**. At step **435**, the receiving site's logical clock in the version vector for the data object is incremented by a fixed value (e.g., one) to reflect the change made at step **425** as a result of the deconfliction at step **420**. The other elements in the version vector are not incremented. In addition, at step **430**, the change(s) to the receiving site's copy of the data object are shared with other site(s) in the system.

If, at step **415**, the receiving site determines that the changing site's version vector for the data object and the receiving site's version vector for the data object are either identical or ordered (i.e., not concurrent), then, at step **440**, the receiving site either incorporates the update into the receiving site's local copy of the data object or discards the update. In one embodiment, the receiving site incorporates the update into the receiving site's local copy of the data object if the receiving site's version vector for the data object is ordered before (i.e., happened before) the changing site's version vector for the data object. Incorporating the update into the receiving site's local copy of the data object includes overwriting data object information in the receiving site's local copy with the superseding changes for the data object included in the update. In one embodiment, the receiving site discards the update if the receiving site's version vector for the data object is identical to the changing site's version vector for the data object. The receiving site may also discard the update if the changing site's version vector for the data object is ordered before (i.e., happened before) the receiving site's version vector for the data object. In this latter case, the update represents an old change that was already incorporated into and been superseded by the receiving site's version of the data object.

If, at step **435**, the update was incorporated into the receiving site's local copy of the data object, then, at step **450**, the changing site's version vector for the data object is merged together with the receiving site's version vector for the data object to produce a new version vector for the data object at the receiving site. Step **450** is similar to step **430**. However, unlike the case where the received update to the data object is in conflict with the receiving site's version of the data object, the new version vector for the data object at the receiving site is not incremented after merging the receiving site's version vector for the data object and the changing site's version vector for the data object.

Method **300** and method **400** of FIGS. **3** and **4** will now be further explained by example with reference to FIG. **5**. FIG. **5** illustrates an example of sharing and deconflicting data changes in multimaster system **100**. Logical time proceeds downward from the top of the figure to the bottom as events occur at the sites **101**, **102**, and **103**. As shown, each site **101**, **102**, and **103** initially has identical copies of the same data object. The data object has two attributes: a Type attribute and a Name attribute. The Type attribute is set to the value "Person" and the Name attribute is set to the value "J.S." in each copy of the data object at each site. In addition, each site **101**, **102**, and **103** maintains a version vector for the data object. Initially, the version vectors are identical (i.e., <1, 0, 0>) reflecting that each site has the same version of the data object. Each version vector has three elements, one for each site **101**, **102**, and **103**. In the example depicted in FIG. **5**, the first (leftmost) element of each version vector corresponds to site **101**, the second (middle) element of each version vector corresponds to site **102**, and the third (rightmost) element of each version vector corresponds to site **103**.

At event **503** at site **101**, a local change is made to site **101**'s copy of the data object. In particular, the Name property is changed from "J.S." to "John Smith". In accordance with step **310** of method **300**, site **101**'s logical clock for the data object is incremented by a fixed value. In the example, site **101**'s logical clock in the version vector for the data object is incremented from 1 to 2.

In accordance with step **315** of method **300**, at event **505**, site **101** shares the change to its copy of the data object with site **102**. In particular, an update is sent from site **101** to site **102**. In one embodiment, the update includes an identifier of the data object, data representing the change made, and data representing site **101**'s version vector for the data object (e.g., <2, 0, 0>).

At event **507**, the update sent from site **101** is received at site **102**. In accordance with step **410** of method **400**, the version vector for the data object received in the update <2, 0, 0> is compared against site **102**'s current version vector for the data object <1, 0, 0>. Such comparison reveals that sites **102**'s version vector happened before (is ordered before) site **101**'s version vector. Thus, the update received at site **102** reflecting the change made at site **101** does not conflict with site **102**'s version of the data object. In accordance with step **440** of method **400**, site **102** incorporates the change received in the update into its local copy of the data object with the change received in the update superseding any differing properties of site **102**'s copy of the data object. In particular, the value of the Name property in site **102**'s copy of the data object is changed from "J.S." to "John Smith". In accordance with step **450** of method **400**, Site **101**'s version vector for the data object received in the update is merged with site **102**'s version vector to produce a new version vector for the data object at site **102** of <2, 0, 0>.

At event **509**, site **101**'s update is propagated by site **102** to site **103**. In one embodiment, site **102** is configured to perform such propagation as part of a partially-meshed or cascading multimaster replication topology. In an alternative embodiment, instead of relying on site **102** to propagate the update, site **101** communicates the update to both site **102** and site **103** as part of a fully meshed multimaster replication topology. At event **511**, site **103** receives the update and incorporates the update into its local copy of the data object and merges version vectors by performing steps similar to those performed by site **102** at event **507**.

Event **513** and event **515** represent concurrent changes to the data object. In particular, at site **102** a Phone # property is added to the data object. At site **103**, an Address property is added to the data object. In accordance with step **310** of method **300**, site **102** and site **103** both increment their logical clock for the data object. At event **517**, site **102** sends an update to site **103** reflecting the addition of the Phone # property. At event **519**, site **103** sends an update to site **102** reflecting the addition of the Address property. Although not shown in FIG. **5**, sites **102** and **103** may also communicate updates to other sites in the system (e.g., site **101**). At event **521**, site **102** receives the update sent from site **103** and detects the conflict. In particular, the version vector received in the update from site **103** (i.e., <2, 0, 1>) is not identical to, nor ordered before or after, the version vector for the object at site **102** (i.e., <2, 1, 0>). In accordance with step **420** of method **400**, site **102** attempts to automatically deconflict the conflict based on a pre-specified set of heuristics and/or deconfliction rules. In the example of FIG. **5**, site **102** compares its copy of the data object with the version of the data object received in the update and determines that the concurrent changes involve changes to non-overlapping properties. Thus, at event **512**, site **102** determines that the conflict can be automatically deconflicting and updates its local copy of the data object accordingly. In particular, the Address property received in the update is added to site **102**'s local copy of the data object. Further, in accordance with step **430** of method **400**, site **102**'s version vector for the data object is merged with site **103**'s version vector for the data object received in the update and the resulting version vector becomes the new version vector for the data object at site **102**. Then, in accordance with step **435** of method **400**, site **102** increments its logical clock in the version vector for the data object by one to produce a **[**newe**]** *new* version vector for the data object at site **102** of <2, 2, 1>.

At event **523**, site **103** performs a process similar to what site **102** performs at event **521**.

Avoiding Needless Repetitive Updates

After event **521** at site **102** and after event **523** at site **103**, site **102** and site **103** both have identical copies of the data object. However, site **102** and site **103** have different version vectors for the data object. In the example, site **102** has a version vector for the data object of <2, 2, 1> and site **103** has a version vector for the data object of <2, 1, 2>. In accordance with step **435** of method **400**, site **102** and site **103** may send an update to each other reflecting their respective automatic deconfliction operations performed at events **521** and **523** respectively. When received by the other site, these updates will be detected as conflict. For example, site **102**'s version vector <2, 2, 1> is not identical, nor ordered before or after, site **103**'s version vector <2, 1, 2>. If no corrective action is taken, site **102** and site **103** will repeatedly and needlessly deconflict, increment their logical clocks for the data object, and send updates to each other even though both sites have identical copies of the data object.

In one embodiment, to avoid needless repetitive updates, at step **420** of method **400**, after a conflict has been detected, a comparison is made between the version of the data object received in the update and the receiving site's version of the data object. If the two versions are identical, then only a merge of the two version vectors is performed (step **430**). The receiving site's local copy of the data object is not changed and the receiving site's logical clock in the version vector for the data object is not incremented (i.e., steps **425** and **435** are not performed). In one embodiment, this comparison includes a property by property comparison between the two versions of the data object.

For example, returning to FIG. **5**, at event **529**, site **102** receives an update from site **103** indicating that site **103** added the Phone # property to its copy of the data object and including its current version vector for the data object of <2, 1, 2>. Upon receiving this update, site **102** detects a conflict because its version vector <2, 2, 1> is not identical to, nor ordered before or after, site **103**'s version vector <2, 1, 2>. Site **102** compares its version of the data object with the version of the data object received in the update from site **103**. Upon determining that the versions are identical (i.e., both versions have the same properties with the same values), site **102** merges the two version vectors to produce a new version vector for the data object at site **102** of <2, 2, 2>. Site **103** performs a similar process at event **531** to arrive at the same version vector <2, 2, 2>. Now that both version vectors are identical, a conflict may **[**not**]** *no* longer be detected and updates relating to the previous deconfliction no longer propagated by the sites.

Per-Link Set Version Vectors

In one embodiment, links connecting two data objects are versioned separately and independently from the data objects connected by the links. In particular, the set of links connecting two objects is associated with its own version vector separate from the versions vectors associated with the two objects. Each site maintains a version vector for each link set. Changes to a link set at a site including adding a link to the set or removing a link from the set result in the site incrementing its local logical clock for the link set and the site sharing the change to the link set with other sites. The versions vectors associated with copies of a link set at the sites can be used to detect and deconflict conflicts involving concurrent changes to two different copies of the same link set in a manner similar that described above for how per-object version vectors can be used to detect and deconflict conflicts involving concurrent changes to two different copies of the same data object.

In addition, per-link set version vectors allow sites to automatically incorporate a concurrent change that includes a change to a link set and a change to a data object connected to another data object by the link set. For example, assume Site A and Site B have the same version of data object X and the same version of data object Y. Further, assume that Site A's version vector for data object X is identical to Site B's version vector for data object X and that Site A's version vector for data object Y is identical to Site B's version vector for data object Y. If a local change is made to data object X at Site A (e.g., by adding a new property), then Site A increments its local logical clock in the version vector for data object X and sends an update to Site B. Assume that, before Site B receives the update regarding the change to data object X at Site A, a local change is made at Site B linking data object X and data object Y. According to one embodiment, this causes Site B to increment its local logical clock in the version vector for the link set connecting data objects X and Y. However, in this case, Site B does not increment its local logical clock for either data object X or data object Y. Site B then sends an update to Site A reflecting the change to the link set between data objects X and Y. Upon receiving the update from Site B regarding the link set change, Site A incorporates the update such that data object X as modified by the change at Site A is linked to data object Y. Similarly, upon receiving the update from Site A regarding the change to data object X, Site B incorporates the update such that data object X as modified by the change at Site A is linked to data object Y. After the updates have been shared with each other, both Site A and Site B have identical copies of data object X and data object Y and identical copies of the links set connected data objects X and Y.

This example is illustrated in FIG. **6**. As shown, initially Site A and Site B have the same version of data object X and the same version of data object Y. Events **603** and **605** represent concurrent changes. In particular, at event **603**, a local change is made to data object X at Site A. For example, a change is made involving a property of data object X. Concurrently, at event **605**, a local change is made at Site B linking data object X and data object Y For example, if data object X and data object Y each represent a particular person, they may be linked through a Friend Of relationship. At event **607**, Site A shares its change to data object X with Site B and includes its version vector for data object X<2, 0, 0> in its update. At event **609**, Site B shares its change to the X-Y link set and includes its version vector for the X-Y link set <1, 0, 0> in its update. Both sites receive and incorporate each other's updates into their respective copies of the database at events **611** and **613**. Note that in this example there is no detected conflict between the concurrent changes because the set of links connecting data objects X and Y is versioned separately and independently of the data objects X and Y themselves.

Using Per-Object Version Vectors to Detect Object Resolution Conflicts

As mentioned, some database systems may support "object resolution". Object resolution involves a user or an automated computing process determining that two or more separate data objects actually represent the same real-world entity and invoking a function of the database system so that the separate data objects are resolved together into a single data object. For example, assume there are two separate data objects, one having a Name property value of "John Smith", the other having a Name property value of "J. S.". A user may decide that these two data objects both represent the same real-world person. Accordingly, in a database system that supports object resolution, the user may invoke a function of the database system so that the two separate data objects are resolved to a single data object having a name property value of "John Smith" or "J.S." as selected by the user resolving the objects together.

In multimaster database systems employing asynchronous replication, it would be desirable to detect as a conflict concurrent changes that include an object resolution change. For example, assume that in database A, User 1 changed the hair color property of a data object representing a person named "J.S." from "brown" to "blonde". Further assume that before the hair color change made by User 1 can be propagated from database A to database B that User 2 changes database B by resolving together the data object representing "J.S." with another data object representing a person named "John Smith". It would be desirable for the multimaster database system to detect these two concurrent changes as a conflict as User 2 may not have decided to resolve "J.S." and "John Smith" together if User 2 had known that John Smith's hair color was changed by User 1. Similarly, User 1 may not have decided to change the hair color of "J.S." had User 1 known that User 2 resolved "J.S." and "John Smith" together.

In one embodiment, per-object version vectors are used to detect as a conflict a concurrent change involving an object resolution change. In particular, when a site resolves two or more objects together, the site increments each local logical clock at the site in each version vector for each data object resolved together. The resolution of the data objects is then shared as an update with other sites. The update includes the sharing site's resulting version vectors for each of the data objects that were resolved together.

According to one embodiment, a site receiving the update detects a conflict by comparing each version vector for each data object in the object resolution update with its version vector for the corresponding data object. If any of the version vectors are concurrent, then a conflict is detected. The resolution of the objects is incorporated into the receiving site's copy of the database only if each and every version vector received the update is identical to or ordered after the corresponding version vector at the receiving site.

As an example, assume data object X at site **101** of FIG. **1** has version vector <1, 0, 0> and data object Y at site **101** has version vector <1, 0, 0>. When data objects X and Y are resolved together at site **101**, each logical clock for data objects X and Y at site **101** is incremented by a fixed value (e.g., one) giving a version vector at site **101** of <2, 0, 0,> for data object X and a version vector at site **101** of <2, 0, 0> for data object Y. When the object resolution change at site **101** is shared by site **101** with other sites (e.g., site **102** and site **103**), the update includes data indicating the object resolution change (i.e., that data objects X and Y were resolved together) and site **101**'s version vectors for the data objects that were resolved together (e.g., <2, 0, 0> for data object X and <2, 0, 0> for data object Y). Further assume that a change concurrent with the object resolution change made at site **101** is made to data object X at site **102** thereby changing the version vector for data object X at site **102** from <1, 0, 0> to <1, 1, 0>. For example, a property of data object X is modified at site **102**. Upon receiving the update sent from site **101** regarding the object resolution change, site **102** will detect these concurrent changes as a conflict. A conflict will be detected at site **102** because a version vector for at least one data object received in the object resolution update from site **101** is concurrent with the version vector for the data object at site **102**. In particular, the version vector for data object X received in the update <2, 0, 0> is concurrent with the version vector for data object X at site

102 <1, 1, 0>. In response to detecting the conflict, site **102** may attempt to automatically deconflict the conflict according to pre-defined heuristics and/or deconfliction rules, or may require input from a user to deconflict the conflict.

Object Resolution Aware Happens After (RAHA)

In one embodiment, a site receiving an update involving a change to a data object that has been resolved together at the receiving site with one or more other data objects will be applied at the receiving site only if each and every data object resolved together at the receiving site is available in the update. If each and every data object is not available in the update, then the update may be placed in the receiving site's pending update queue. A process at the receiving site periodically scans the pending update queue for updates that, when combined, include each and every data object resolved together at the receiving site. If the scanning process discovers such a combination, then the updates may be applied atomically in combination at the receiving site.

For example, consider the following events that occur in system **100** of FIG. **1**:

(1) Both site **101** and site **102** have copies of data objects X, Y, and Z each at version <1, 0, 0>. Further, data objects X, Y, and Z are resolved together at both site **101** and site **102**.

(2) At site **101**, data object X is unresolved from data objects Y and Z. Each version vector at site **101** is incremented such that each data object X, Y, and Z is now at version <2, 0, 0> at site **101**.

(3) Site **101** sends an update to site **102** that includes data representing data object X at version <2, 0, 0> and data representing the resolution of data objects Y and Z each at version <2, 0, 0>.

(4) Site **102** receives the update from site **101** and places the update in its pending update queue. The update is placed in the pending update queue because neither data object X at version <2, 0, 0> nor the resolution of data objects Y and Z each at version <2, 0, 0> includes all the data objects in the resolution of data objects X, Y, and Z each at version <1, 0, 0> at site **102**.

(5) A scanning process at site **102** scans the pending update queue for updates that, when combined, include each and every of the data objects X, Y, and Z resolved together at site **102**. The scanning process finds the updates received from site **102** in the pending update for data objects X, Y, and Z and applies them to site **102**'s copy of the body data after which both site **101** and site **102** have data object X at version <2, 0, 0> unresolved from resolved data objects Y and Z, each at version <2, 0, 0>.

Per-Site Global Acknowledgement Version Vectors

In one embodiment, to aid in determining what changes should be shared with other sites in the system, each site maintains a single global acknowledgement version vector which the site periodically shares with other sites in the system. A site's global acknowledgement version vector reflects a merging of all version vectors for all changes successfully applied to the site's local copy of the shared body of data. When a sending site shares a change with a receiving site, the receiving system is guaranteed to have successfully already received all changes that are ordered before (i.e., happened before) the receiving site's global acknowledgement version vector. Thus, the sending site need not send those changes to the receiving site that are ordered before (i.e., happened before) the receiving site's global acknowledgement version vector.

In one embodiment, changes in the pending update queue at a site are shared with other sites even though the updates are pending and have not yet been deconflicted. This is done for correctness in systems in which the replication topology is cyclic and/or dynamic. For example, consider system **100** of FIG. **1** in which all three sites **101**, **102**, and **103** are configured to share changes with each other. Further consider the following events that occur in system **100**:

(1) Site **101** sends to site **102** an update for data object A at version <1, 0, 0> and an update for data object B at version <1, 0, 0>.

(2) Concurrent with event (1), site **102** edits object A to version <0, 1, 0>.

(3) Site **102**, upon receiving the update for object B a version <1, 0, 0,> from site **101**, applies the update to its local copy of object B. Site **102**, upon receiving the update for data object A at version <1, 0, 0> from site **101**, places the update in a pending update queue at site **102**.

(4) Site **102** sends to site **103** an update for data object A at version <0, 1, 0> and an update for data object B at version <1, 0, 0>.

(5) Site **103**, up receiving the update for object B at version <1, 0, 0> from site **102**, applies the update to its local copy of object B. Site **103**, upon receiving the update for data object A at version <0, 1, 0> from site **102**, applies the update to its local copy of object A. Site **103**'s global acknowledgement version vector is at <1, 1, 0> as a result of merging the version vector for data object A at version <0, 1, 0> and the version vector for data object B at version <1, 0, 0>.

In this example, if, at event (4), the update for data object A at version <1, 0, 0> in site **102**'s pending update queue is not also shared with site **103**, then site **103** may never receive the update because site **103**'s global acknowledgment version vector indicates that site **103** has already received the update. Thus, according to one embodiment, site **102** at event (4) will also share with site **103** the update in its pending update queue for data object A at version <1, 0, 0>. This is so even though the update has not yet been deconflicted. In one embodiment, the pending update is also stored in site **103**'s pending update queue. In this situation, the conflict can now be deconflicted at either site **102** or site **103**.

Implementing Mechanisms—Hardware Overview

According to one embodiment, the techniques described herein are implemented by one or more special-purpose computing devices. The special-purpose computing devices may be hard-wired to perform the techniques, or may include digital electronic devices such as one or more application-specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs) that are persistently programmed to perform the techniques, or may include one or more general purpose hardware processors programmed to perform the techniques pursuant to program instructions in firmware, memory, other storage, or a combination. Such special-purpose computing devices may also combine custom hard-wired logic, ASICs, or FPGAs with custom programming to accomplish the techniques. The special-purpose computing devices may be desktop computer systems, portable computer systems, handheld devices, networking devices or any other device that incorporates hard-wired and/or program logic to implement the techniques.

For example, FIG. **7** is a block diagram that illustrates a computer system **700** upon which an embodiment of the invention may be implemented. Computer system **700** includes a bus **702** or other communication mechanism for communicating information, and a hardware processor **704** coupled with bus **702** for processing information. Hardware processor **704** may be, for example, a general purpose microprocessor.

Computer system **700** also includes a main memory **706**, such as a random access memory (RAM) or other dynamic storage device, coupled to bus **702** for storing information and instructions to be executed by processor **704**. Main memory **706** also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor **704**. Such instructions, when stored in storage media accessible to processor **704**, render computer system **700** into a special-purpose machine that is customized to perform the operations specified in the instructions.

Computer system **700** further includes a read only memory (ROM) **708** or other static storage device coupled to bus **702** for storing static information and instructions for processor **704**. A storage device **710**, such as a magnetic disk or optical disk, is provided and coupled to bus **702** for storing information and instructions.

Computer system **700** may be coupled via bus **702** to a display **712**, such as a cathode ray tube (CRT), for displaying **40** information to a computer user. An input device **714**, including alphanumeric and other keys, is coupled to bus **702** for communicating information and command selections to processor **704**. Another type of user input device is cursor control **716**, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor **704** and for controlling cursor movement on display **712**. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

Computer system **700** may implement the techniques described herein using customized hard-wired logic, one or more ASICs or FPGAs, firmware and/or program logic which in combination with the computer system causes or programs computer system **700** to be a special-purpose machine. According to one embodiment, the techniques herein are performed by computer system **700** in response to processor **704** executing one or more sequences of one or more instructions contained in main memory **706**. Such instructions may be read into main memory **706** from another storage medium, such as storage device **710**. Execution of the sequences of instructions contained in main memory **706** causes processor **704** to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions.

The term "non-transitory media" as used herein refers to any media that store data and/or instructions that cause a machine to operation in a specific fashion. Such non-transitory media may comprise non-volatile media and/or volatile media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device **710**. Volatile media includes dynamic memory, such as main memory **706**. Common forms of non-transitory media include, for example, a floppy disk, a flexible disk, hard disk, solid state drive, magnetic tape, or any other magnetic data storage medium, a CD-ROM, any other optical data storage medium, any physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, NVRAM, any other memory chip or cartridge.

Non-transitory media is distinct from but may be used in conjunction with transmission media. Transmission media participates in transferring information between non-transitory media. For example, transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus **702**. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

Various forms of media may be involved in carrying one or more sequences of one or more instructions to processor **704** for execution. For example, the instructions may initially be carried on a magnetic disk or solid state drive of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system **700** can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus **702**. Bus **702** carries the data to main memory **706**, from which processor **704** retrieves and executes the instructions. The instructions received by main memory **706** may optionally be stored on storage device **710** either before or after execution by processor **704**.

Computer system **700** also includes a communication interface **718** coupled to bus **702**. Communication interface **718** provides a two-way data communication coupling to a network link **720** that is connected to a local network **722**. For example, communication interface **718** may be an integrated services digital network (ISDN) card, cable modem, satellite modem, or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface **718** may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface **718** sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

Network link **720** typically provides data communication through one or more networks to other data devices. For example, network link **720** may provide a connection through local network **722** to a host computer **724** or to data equipment operated by an Internet Service Provider (ISP) **726**. ISP **726** in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" **728**. Local network **722** and Internet **728** both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link **720** and through communication interface **718**, which carry the digital data to and from computer system **700**, are example forms of transmission media.

Computer system **700** can send messages and receive data, including program code, through the network(s), network link **720** and communication interface **718**. In the Internet example, a server **730** might transmit a requested code for an application program through Internet **728**, ISP **726**, local network **722** and communication interface **718**.

The received code may be executed by processor **704** as it is received, and/or stored in storage device **710**, or other non-volatile storage for later execution.

In the foregoing specification, embodiments of the invention have been described with reference to numerous specific details that may vary from implementation to implementation. Thus, the sole and exclusive indicator of what is the invention, and is intended by the applicants to be the invention, is the set of claims that issue from this application, in the specific form in which such claims issue, including any subsequent correction. Any definitions expressly set forth herein for terms contained in such claims shall govern the meaning of such terms as used in the claims.

Hence, no limitation, element, property, feature, advantage or attribute that is not expressly recited in a claim should limit the scope of such claim in any way. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

[1. In a multimaster database system comprising a plurality of sites, a method for sharing and deconflicting data changes, the method comprising:

at a first site of the plurality of sites, making a first change to a set of one or more links connecting two data objects by adding or removing one or more links from the set of links;

wherein the set of links is associated at the first site with a first version vector for the set of one or more links;

at a second site of the plurality of sites, making a second change to the set of links connecting the two data objects by adding or removing a link from the set of links;

wherein the set of links is associated at the second site with a second version vector for the set of links;

sharing the first change with the second site of the plurality of sites;

receiving, at the second site, an update reflecting the first change to the set of links at the first site;

wherein the update includes:

an identification of the set of links,

data reflecting the first change to the set of links at the first site, and

the first version vector for the set of links;

at the second site, comparing the first version vector for the set of links to the second version vector for the set of links to determine whether the first change to the set of links at the first set and the second change to the set of links at the second site are identical, ordered, or concurrent;

wherein the method is performed by a plurality of computing devices.**]**

[2. The method according to claim **1**, wherein the second change to the set of links is made at the second site before the first change to the set of links is made at the first site; and wherein the method further comprises determining, based on the comparing, that the first and second changes are ordered.**]**

[3. The method according to claim **1**, wherein the second change to the set of links is made at the second site before the first change to the set of links is made at the first site; and wherein the method further comprises determining, based on the comparing, that the first and second changes are concurrent.**]**

[4. In a multimaster database system comprising a plurality of sites, a method for sharing and deconflicting data changes, the method comprising:

at a first site of the plurality of sites:

resolving two or more data objects together to produce a data object resolution change;

sharing the data object resolution change with one or more other sites of the plurality of sites;

at a second site of the plurality of sites:

receiving an update reflecting the data object resolution change made at the first site;

wherein the update includes:

an identification of each of the two or more data objects,

data that indicates that the two or more data objects were resolved together, and

for each of the two or more data objects, a version vector for the data object;

comparing, for each of one or more of the two or more data objects, the version vector for the data object received in the update to a version vector at the second site for the data object to determine whether the data object resolution change and a version at the second site of the data object are identical, ordered, or concurrent;

determining, based on the comparing, that the data object resolution change is concurrent with a version at the second site of at least one of the two or more data objects; and

in response to determining that the data object resolution change is concurrent with a version at the second site of at least one of the two or more data objects, determining that the data object resolution change conflicts with a version at the second site of at least one of the two or more data objects;

wherein the method is performed by a plurality of computing devices.**]**

[5. One or more non-transitory computer-readable media storing instructions which, when executed by a plurality of computing devices, cause performing a method for sharing and deconflicting data changes in a multimaster database system comprising a plurality of sites, the method comprising:

at a first site of the plurality of sites, making a first change to a set of one or more links connecting two data objects by adding or removing one or more links from the set of links;

wherein the set of links is associated at the first site with a first version vector for the set of one or more links;

at a second site of the plurality of sites, making a second change to the set of links connecting the two data objects by adding or removing a link from the set of links;

wherein the set of links is associated at the second site with a second version vector for the set of links;

sharing the first change with the second site of the plurality of sites;

receiving, at the second site, an update reflecting the first change to the set of links at the first site;

wherein the update includes:

an identification of the set of links,

data reflecting the first change to the set of links at the first site, and

the first version vector for the set of links;

at the second site, comparing the first version vector for the set of links to the second version vector for the set of links to determine whether the first change to the set of links at the first set and the second change to the set of links at the second site are identical, ordered, or concurrent.**]**

[6. The one or more non-transitory computer-readable media of claim **5**, wherein the second change to the set of links is made at the second site before the first change to the set of links is made at the first site; and wherein the method further comprises determining, based on the comparing, that the first and second changes are ordered.**]**

[7. The one or more non-transitory computer-readable media of claim **5**, wherein the second change to the set of links is made at the second site before the first change to the set of links is made at the first site; and wherein the method further comprises determining, based on the comparing, that the first and second changes are concurrent.**]**

[8. One or more non-transitory computer-readable media storing instructions which, when executed by a plurality of computing devices, cause performing a method for sharing and deconflicting data changes in a multimaster database system comprising a plurality of sites, the method comprising:

at a first site of the plurality of sites:

resolving two or more data objects together to produce a data object resolution change;

sharing the data object resolution change with one or more other sites of the plurality of sites;

at a second site of the plurality of sites:

receiving an update reflecting the data object resolution change made at the first site;

wherein the update includes:

an identification of each of the two or more data objects,

data that indicates that the two or more data objects were resolved together, and

for each of the two or more data objects, a version vector for the data object;

comparing, for each of one or more of the two or more data objects, the version vector for the data object received in the update to a version vector at the second site for the data object to determine whether the data object resolution change and a version at the second site of the data object are identical, ordered, or concurrent;

determining, based on the comparing, that the data object resolution change is concurrent with a version at the second site of at least one of the two or more data objects; and

in response to determining that the data object resolution change is concurrent with a version at the second site of at least one of the two or more data objects, determining that the data object resolution change conflicts with a version at the second site of at least one of the two or more data objects.]

9. In a multimaster database system comprising a plurality of sites, a method for sharing and deconflicting data changes, the method comprising:

at a first site of the plurality of sites, making a first change to a set of one or more links connecting two data objects by adding or removing one or more links from the set of links;

wherein the set of links is associated at the first site with a first version vector for the set of links, the first version vector versioning the set of links separately and independently of the two data objects connected by the set of links;

at a second site of the plurality of sites, making a second change to the set of links connecting the two data objects by adding or removing a link from the set of links;

wherein the set of links is associated at the second site with a second version vector for the set of links;

sharing the first change with the second site of the plurality of sites;

receiving, at the second site, an update reflecting the first change to the set of links at the first site;

wherein the update includes:

an identification of the set of links,

data reflecting the first change to the set of links at the first site, and

the first version vector for the set of links;

at the second site, comparing the first version vector for the set of links to the second version vector for the set of links to determine whether the first change to the set of links at the first set and the second change to the set of links at the second site are identical, ordered, or concurrent;

wherein the method is performed by a plurality of computing devices.

10. The method according to claim 9, wherein the second change to the set of links is made at the second site before the first change to the set of links is made at the first site; and wherein the method further comprises determining, based on the comparing, that the first and second changes are ordered.

11. The method according to claim 9, wherein the second change to the set of links is made at the second site before the first change to the set of links is made at the first site; and wherein the method further comprises determining, based on the comparing, that the first and second changes are concurrent.

12. In a multimaster database system comprising a plurality of sites, a method for sharing and deconflicting data changes, the method comprising:

at a first site of the plurality of sites:

resolving two or more data objects together to produce a data object resolution change;

sharing the data object resolution change with one or more other sites of the plurality of sites;

at a second site of the plurality of sites:

receiving an update reflecting the data object resolution change made at the first site;

wherein the update includes:

an identification of each of the two or more data objects,

data that indicates that the two or more data objects were resolved together, and

for each of the two or more data objects, a version vector for the data object;

comparing, for each of one or more of the two or more data objects, the version vector for the data object received in the update to a version vector at the second site for the data object to determine whether the data object resolution change and a version at the second site of the data object are identical, ordered, or concurrent;

determining, based on the comparing, that the data object resolution change is concurrent with a version at the second site of at least one of the two or more data objects; and

in response to determining that the data object resolution change is concurrent with a version at the second site of at least one of the two or more data objects, determining that the data object resolution change conflicts with a version at the second site of at least one of the two or more data objects;

wherein the method is performed by a plurality of computing devices.

13. One or more non-transitory computer-readable media storing instructions which, when executed by a plurality of computing devices, cause performing a method for sharing and deconflicting data changes in a multimaster database system comprising a plurality of sites, the method comprising:

at a first site of the plurality of sites, making a first change to a set of one or more links connecting two data objects by adding or removing one or more links from the set of links;

wherein the set of links is associated at the first site with a first version vector for the set of links, the first version *vector versioning the set of links separately and independently of the two data objects connected by the set of links;*

at a second site of the plurality of sites, making a second change to the set of links connecting the two data objects by adding or removing a link from the set of links;

wherein the set of links is associated at the second site with a second version vector for the set of links;

sharing the first change with the second site of the plurality of sites;

receiving, at the second site, an update reflecting the first change to the set of links at the first site;

wherein the update includes:

an identification of the set of links,

data reflecting the first change to the set of links at the first site, and

the first version vector for the set of links;

at the second site, comparing the first version vector for the set of links to the second version vector for the set of links to determine whether the first change to the set of links at the first set and the second change to the set of links at the second site are identical, ordered, or concurrent.

14. The one or more non-transitory computer-readable media of claim 13, wherein the second change to the set of links is made at the second site before the first change to the set of links is made at the first site; and wherein the method further comprises determining, based on the comparing, that the first and second changes are ordered.

15. The one or more non-transitory computer-readable media of claim 13, wherein the second change to the set of links is made at the second site before the first change to the set of links is made at the first site; and wherein the method further comprises determining, based on the comparing, that the first and second changes are concurrent.

16. One or more non-transitory computer-readable media storing instructions which, when executed by a plurality of computing devices, cause performing a method for sharing and deconflicting data changes in a multimaster database system comprising a plurality of sites, the method comprising:

at a first site of the plurality of sites:

resolving two or more data objects together to produce a data object resolution change;

sharing the data object resolution change with one or more other sites of the plurality of sites;

at a second site of the plurality of sites:

receiving an update reflecting the data object resolution change made at the first site;

wherein the update includes:

an identification of each of the two or more data objects,

data that indicates that the two or more data objects were resolved together, and

for each of the two or more data objects, a version vector for the data object;

comparing, for each of one or more of the two or more data objects, the version vector for the data object received in the update to a version vector at the second site for the data object to determine whether the data object resolution change and a version at the second site of the data object are identical, ordered, or concurrent;

determining, based on the comparing, that the data object resolution change is concurrent with a version at the second site of at least one of the two or more data objects; and

in response to determining that the data object resolution change is concurrent with a version at the second site of at least one of the two or more data objects, determining that the data object resolution change conflicts with a version at the second site of at least one of the two or more data objects.

* * * * *