

Substitute for form 1449A/PTO INFORMATION DISCLOSURE STATEMENT BY APPLICANT <i>(Use as many sheets as necessary)</i> E-filed via EFS-Web on December 19, 2013	Complete if Known	
	Application Number	
	Filing Date	
	First Named Inventor	Sandip AGARWALA
	Group Art Unit	
	Examiner Name	
Sheet 1 of 1		Attorney Docket No: 009462.P001

US PATENT DOCUMENTS					
Examiner Initial *	Cite No ¹	USP Document Number	Publication or Issue Date MM-DD-YYYY	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear

FOREIGN PATENT DOCUMENTS						
Examiner Initials*	Cite No ¹	Foreign Patent Document Country Code/Number/Kind Code (if known)	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear	T ²

OTHER DOCUMENTS -- NON PATENT LITERATURE DOCUMENTS			
Examiner Initials*	Cite No ¹	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T ²
	NPL1	"The Case for Persistent Full Clones," Deepstorage.net, http://getgreenbytes.com/wp-content/uploads/2013/05/FULL_CLONE_PERSISTENT_VDI-FINAL.pdf , 18 pages.	
	NPL2	RODEH, OHAD, "B-trees, Shadowing, and Clones," <u>ACM Transactions on Storage (TOS)</u> 3, no. 4, https://www.usenix.org/legacy/events/lsf07/tech/rodeh.pdf , (2008), 51 pages.	
	NPL3	RODEH, OHAD, "B-trees, Shadowing, and Clones," <u>ACM Transactions on Computational Logic</u> , Vol. V, No. N, (August 2007), 26 pages.	

EXAMINER

DATE CONSIDERED

Based on PTO/SB/D8A(09-05) - Substitute Disclosure Statement Form (PTO-1449) as modified by BSTZ 03/26/07
 * EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. ¹ Applicant's unique citation designation number (optional) ² Applicant is to place a check mark here if English language Translation is attached

Electronic Patent Application Fee Transmittal				
Application Number:				
Filing Date:				
Title of Invention:		SYSTEMS AND METHODS FOR IMPLEMENTING AN ENTERPRISE-CLASS CONVERGED COMPUTE-NETWORK-STORAGE APPLIANCE		
First Named Inventor/Applicant Name:		Sandip AGARWALA		
Filer:		Eric Replogle/Katie Nelson		
Attorney Docket Number:		009462.P001		
Filed as Large Entity				
Utility under 35 USC 111(a) Filing Fees				
Description	Fee Code	Quantity	Amount	Sub-Total in USD(\$)
Basic Filing:				
Utility application filing	1011	1	280	280
Utility Search Fee	1111	1	600	600
Utility Examination Fee	1311	1	720	720
Pages:				
Claims:				
Claims in Excess of 20	1202	22	80	1760
Independent claims in excess of 3	1201	3	420	1260
Miscellaneous-Filing:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD(\$)
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				
Miscellaneous:				
Total in USD (\$)				4620

Electronic Acknowledgement Receipt

EFS ID:	17718290
Application Number:	14135485
International Application Number:	
Confirmation Number:	4404
Title of Invention:	SYSTEMS AND METHODS FOR IMPLEMENTING AN ENTERPRISE-CLASS CONVERGED COMPUTE-NETWORK-STORAGE APPLIANCE
First Named Inventor/Applicant Name:	Sandip AGARWALA
Customer Number:	8791
Filer:	Eric Replogle/Katie Nelson
Filer Authorized By:	Eric Replogle
Attorney Docket Number:	009462.P001
Receipt Date:	19-DEC-2013
Filing Date:	
Time Stamp:	21:00:30
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	\$ 4620
RAM confirmation Number	9644
Deposit Account	022666
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

Charge any Additional Fees required under 37 C.F.R. Section 1.16 (National application filing, search, and examination fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.19 (Document supply fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.21 (Miscellaneous fees and charges)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Application Data Sheet	2013-12-19_9462P001_ADS.pdf	1279993 0beb971245de21783372d03fb848b4950d7584d6	no	9

Warnings:

Information:

2		2013-12-19_9462P001_Application.pdf	278004 2f30d167be5ce9c5c71161dad577136ed2a5795e	yes	52
---	--	-------------------------------------	--	-----	----

Multipart Description/PDF files in .zip description

Document Description	Start	End
Specification	1	43
Claims	44	51
Abstract	52	52

Warnings:

Information:

3	Drawings-only black and white line drawings	2013-12-19_9462P001_Figures.pdf	133725 e85cf4494b582d98a11040de57fd2f5055528367	no	21
---	---	---------------------------------	--	----	----

Warnings:

Information:

4	Oath or Declaration filed	2013-12-19_9462P001_DECS.pdf	279542 0b3e312c3a897bf35069951b43886e4680992502	no	5
---	---------------------------	------------------------------	--	----	---

Warnings:

Information:

5	Transmittal Letter	2013-12-19_9462P001_IDS_Transmittal.pdf	49933 06dcb4ba3d6399cbdd50b4bac935aa86cabce39	no	2
---	--------------------	---	--	----	---

Warnings:

Information:

6	Information Disclosure Statement (IDS) Form (SB08)	2013-12-19_9462P001_IDS.pdf	118832 64b882de424222b226dde08fe842c7742715b95f	no	1
---	--	-----------------------------	--	----	---

Warnings:

Information:

This is not an USPTO supplied IDS fillable form					
7	Non Patent Literature	NPL1_Persistent_full_clones.pdf	456054 4c82b39bd90d72ff00908b3ad27d081b12f edd25	no	18
Warnings:					
Information:					
8	Non Patent Literature	NPL2_rodeh.pdf	682765 73bea0b2aba25a6f8ba74c0a26a6e8be393 ec683	no	51
Warnings:					
Information:					
9	Non Patent Literature	NPL3_Rodeh_B-trees.pdf	498125 0e9135e67adf7ba517139833bcfa43e9a05 b6ea5	no	26
Warnings:					
Information:					
10	Fee Worksheet (SB06)	fee-info.pdf	38155 1fb6e0c240b7d117b4dcbe8e4c34acda784 3ecb2	no	2
Warnings:					
Information:					
Total Files Size (in bytes):			3815128		
This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503. <u>New Applications Under 35 U.S.C. 111</u> If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application. <u>National Stage of an International Application under 35 U.S.C. 371</u> If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course. <u>New International Application Filed with the USPTO as a Receiving Office</u> If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.					

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	009462.P001
		Application Number	
Title of Invention	Systems and Methods for Implementing an Enterprise-Class Converged Compute-Network-Storage Appliance		
The application data sheet is part of the provisional or nonprovisional application for which it is being submitted. The following form contains the bibliographic data arranged in a format specified by the United States Patent and Trademark Office as outlined in 37 CFR 1.76. This document may be completed electronically and submitted to the Office in electronic format using the Electronic Filing System (EFS) or the document may be printed and included in a paper filed application.			

Secrecy Order 37 CFR 5.2

☐	Portions or all of the application associated with this Application Data Sheet may fall under a Secrecy Order pursuant to 37 CFR 5.2 (Paper filers only. Applications that fall under Secrecy Order may not be filed electronically.)
--------------------------	---

Inventor Information:

Inventor 1					Remove	
Legal Name						
Prefix	Given Name	Middle Name	Family Name	Suffix		
	Sandip		AGARWALA			
Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service						
City	Cupertino	State/Province	CA	Country of Residence i	US	
Mailing Address of Inventor:						
Address 1		10630 Mine Court				
Address 2						
City	Cupertino	State/Province	CA			
Postal Code	95014	Country i	US			
Inventor 2					Remove	
Legal Name						
Prefix	Given Name	Middle Name	Family Name	Suffix		
	Abhishek		CHATURVEDI			
Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service						
City	Mountain View	State/Province	CA	Country of Residence i	US	
Mailing Address of Inventor:						
Address 1		2114 Windrose Place				
Address 2						
City	Mountain View	State/Province	CA			
Postal Code	94043	Country i	US			
Inventor 3					Remove	
Legal Name						
Prefix	Given Name	Middle Name	Family Name	Suffix		
	Shravan		GAONKAR			
Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service						

Application Data Sheet 37 CFR 1.76		Attorney Docket Number		009462.P001	
		Application Number			
Title of Invention		Systems and Methods for Implementing an Enterprise-Class Converged Compute-Network-Storage Appliance			

City	Gainesville	State/Province	FL	Country of Residence i	US
-------------	-------------	-----------------------	----	-------------------------------	----

Mailing Address of Inventor:

Address 1	4000 NW 51st Street				
Address 2	Apartment B37				
City	Gainesville	State/Province	FL		
Postal Code	32606	Country i	US		
Remove					
Inventor 4					
Legal Name					
Prefix	Given Name	Middle Name	Family Name	Suffix	
	Mallikarjunan		MAHALINGAM		
Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service					
City	Cupertino	State/Province	CA	Country of Residence i	US

Mailing Address of Inventor:

Address 1	22009 Oakdell Place				
Address 2					
City	Cupertino	State/Province	CA		
Postal Code	95014	Country i	US		
Remove					
Inventor 5					
Legal Name					
Prefix	Given Name	Middle Name	Family Name	Suffix	
	Sazzala		REDDY		
Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service					
City	Los Altos	State/Province	CA	Country of Residence i	US

Mailing Address of Inventor:

Address 1	530 Torwood Court				
Address 2					
City	Los Altos	State/Province	CA		
Postal Code	94022	Country i	US		
Remove					
Inventor 6					
Legal Name					
Prefix	Given Name	Middle Name	Family Name	Suffix	
	Smit		SHAH		
Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service					
City	Sunnyvale	State/Province	CA	Country of Residence i	US

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	009462.P001
		Application Number	
Title of Invention	Systems and Methods for Implementing an Enterprise-Class Converged Compute-Network-Storage Appliance		

Mailing Address of Inventor:

Address 1	1229 Vicente Drive		
Address 2	Apartment 28		
City	Sunnyvale	State/Province	CA
Postal Code	94086	Country i	US

Inventor 7[Remove](#)**Legal Name**

Prefix	Given Name	Middle Name	Family Name	Suffix
	Faraz		SHAIKH	

Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service

City	Sunnyvale	State/Province	CA	Country of Residence i	US
------	-----------	----------------	----	------------------------	----

Mailing Address of Inventor:

Address 1	1000 Escalon Avenue		
Address 2	Apartment Q2132		
City	Sunnyvale	State/Province	CA
Postal Code	94085	Country i	US

Inventor 8[Remove](#)**Legal Name**

Prefix	Given Name	Middle Name	Family Name	Suffix
	Praveen		VEGULLA	

Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service

City	Cupertino	State/Province	CA	Country of Residence i	US
------	-----------	----------------	----	------------------------	----

Mailing Address of Inventor:

Address 1	10846 Willowbrook Way		
Address 2			
City	Cupertino	State/Province	CA
Postal Code	95014	Country i	US

Inventor 9[Remove](#)**Legal Name**

Prefix	Given Name	Middle Name	Family Name	Suffix
	Krishna		YADAPPANAVAR	

Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service

City	Sunnyvale	State/Province	CA	Country of Residence i	US
------	-----------	----------------	----	------------------------	----

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	009462.P001
		Application Number	
Title of Invention	Systems and Methods for Implementing an Enterprise-Class Converged Compute-Network-Storage Appliance		

Mailing Address of Inventor:

Address 1		420 Cirrus Avenue	
Address 2			
City	Sunnyvale	State/Province	CA
Postal Code	94087	Country i	US
Inventor 10			Remove
Legal Name			
Prefix	Given Name	Middle Name	Family Name
	Jeffrey	A.	ZABARSKY
Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service			
City	Sunnyvale	State/Province	CA
Country of Residence i		US	

Mailing Address of Inventor:

Address 1		990 Ticonderoga Drive	
Address 2			
City	Sunnyvale	State/Province	CA
Postal Code	94087	Country i	US
All Inventors Must Be Listed - Additional Inventor Information blocks may be generated within this form by selecting the Add button. Add			

Correspondence Information:

Enter either Customer Number or complete the Correspondence Information section below. For further information see 37 CFR 1.33(a).	
☐ An Address is being provided for the correspondence Information of this application.	
Customer Number	08791
Email Address	mail@bstz.com Add Email Remove Email

Application Information:

Title of the Invention	Systems and Methods for Implementing an Enterprise-Class Converged Compute-Network-Storage Appliance		
Attorney Docket Number	009462.P001	Small Entity Status Claimed ☐	
Application Type	Nonprovisional		
Subject Matter	Utility		
Total Number of Drawing Sheets (if any)	19	Suggested Figure for Publication (if any)	

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	009462.P001
		Application Number	
Title of Invention	Systems and Methods for Implementing an Enterprise-Class Converged Compute-Network-Storage Appliance		

Publication Information:
☐ Request Early Publication (Fee required at time of Request 37 CFR 1.219)

☒ **Request Not to Publish.** I hereby request that the attached application not be published under 35 U.S.C. 122(b) and certify that the invention disclosed in the attached application **has not and will not** be the subject of an application filed in another country, or under a multilateral international agreement, that requires publication at eighteen months after filing.

Representative Information:

Representative information should be provided for all practitioners having a power of attorney in the application. Providing this information in the Application Data Sheet does not constitute a power of attorney in the application (see 37 CFR 1.32). Either enter Customer Number or complete the Representative Name section below. If both sections are completed the customer Number will be used for the Representative Information during processing.

Please Select One:	☒ Customer Number	☐ US Patent Practitioner	☐ Limited Recognition (37 CFR 11.9)
Customer Number	08791		

Domestic Benefit/National Stage Information:

This section allows for the applicant to either claim benefit under 35 U.S.C. 119(e), 120, 121, or 365(c) or indicate National Stage entry from a PCT application. Providing this information in the application data sheet constitutes the specific reference required by 35 U.S.C. 119(e) or 120, and 37 CFR 1.78.

When referring to the current application, please leave the application number blank.

Prior Application Status	Pending	Remove	
Application Number	Continuity Type	Prior Application Number	Filing Date (YYYY-MM-DD)
	Claims benefit of provisional	61739685	2012-12-19
Additional Domestic Benefit/National Stage Data may be generated within this form by selecting the Add button.			Add

Foreign Priority Information:

This section allows for the applicant to claim priority to a foreign application. Providing this information in the application data sheet constitutes the claim for priority as required by 35 U.S.C. 119(b) and 37 CFR 1.55(d). When priority is claimed to a foreign application that is eligible for retrieval under the priority document exchange program (PDX) the information will be used by the Office to automatically attempt retrieval pursuant to 37 CFR 1.55(h)(1) and (2). Under the PDX program, applicant bears the ultimate responsibility for ensuring that a copy of the foreign application is received by the Office from the participating foreign intellectual property office, or a certified copy of the foreign priority application is filed, within the time period specified in 37 CFR 1.55(g)(1).

Remove			
Application Number	Country ⁱ	Filing Date (YYYY-MM-DD)	Access Code ⁱ (if applicable)

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	009462.P001
		Application Number	
Title of Invention	Systems and Methods for Implementing an Enterprise-Class Converged Compute-Network-Storage Appliance		

Additional Foreign Priority Data may be generated within this form by selecting the **Add** button.

Add

Statement under 37 CFR 1.55 or 1.78 for AIA (First Inventor to File) Transition Applications

☐ This application (1) claims priority to or the benefit of an application filed before March 16, 2013 and (2) also contains, or contained at any time, a claim to a claimed invention that has an effective filing date on or after March 16, 2013.

NOTE: By providing this statement under 37 CFR 1.55 or 1.78, this application, with a filing date on or after March 16, 2013, will be examined under the first inventor to file provisions of the AIA.

Authorization to Permit Access:

☒ Authorization to Permit Access to the Instant Application by the Participating Offices

If checked, the undersigned hereby grants the USPTO authority to provide the European Patent Office (EPO), the Japan Patent Office (JPO), the Korean Intellectual Property Office (KIPO), the World Intellectual Property Office (WIPO), and any other intellectual property offices in which a foreign application claiming priority to the instant patent application is filed access to the instant patent application. See 37 CFR 1.14(c) and (h). This box should not be checked if the applicant does not wish the EPO, JPO, KIPO, WIPO, or other intellectual property office in which a foreign application claiming priority to the instant patent application is filed to have access to the instant patent application.

In accordance with 37 CFR 1.14(h)(3), access will be provided to a copy of the instant patent application with respect to: 1) the instant patent application-as-filed; 2) any foreign application to which the instant patent application claims priority under 35 U.S.C. 119(a)-(d) if a copy of the foreign application that satisfies the certified copy requirement of 37 CFR 1.55 has been filed in the instant patent application; and 3) any U.S. application-as-filed from which benefit is sought in the instant patent application.

In accordance with 37 CFR 1.14(c), access may be provided to information concerning the date of filing this Authorization.

Applicant Information:

Providing assignment information in this section does not substitute for compliance with any requirement of part 3 of Title 37 of CFR to have an assignment recorded by the Office.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	009462.P001
		Application Number	
Title of Invention	Systems and Methods for Implementing an Enterprise-Class Converged Compute-Network-Storage Appliance		

Applicant 1				Remove	
If the applicant is the inventor (or the remaining joint inventor or inventors under 37 CFR 1.45), this section should not be completed. The information to be provided in this section is the name and address of the legal representative who is the applicant under 37 CFR 1.43; or the name and address of the assignee, person to whom the inventor is under an obligation to assign the invention, or person who otherwise shows sufficient proprietary interest in the matter who is the applicant under 37 CFR 1.46. If the applicant is an applicant under 37 CFR 1.46 (assignee, person to whom the inventor is obligated to assign, or person who otherwise shows sufficient proprietary interest) together with one or more joint inventors, then the joint inventor or inventors who are also the applicant should be identified in this section.					
Clear					
☒ Assignee		☐ Legal Representative under 35 U.S.C. 117		☐ Joint Inventor	
☐ Person to whom the inventor is obligated to assign.			☐ Person who shows sufficient proprietary interest		
If applicant is the legal representative, indicate the authority to file the patent application, the inventor is:					
Name of the Deceased or Legally Incapacitated Inventor :					
If the Applicant is an Organization check here. ☒					
Organization Name		Storvisor, Inc.			
Mailing Address Information:					
Address 1		333 W. El Camino Real, Suite 350			
Address 2					
City		Sunnyvale	State/Province		CA
Country ⁱ	US	Postal Code		94087	
Phone Number			Fax Number		
Email Address					
Additional Applicant Data may be generated within this form by selecting the Add button. Add					

Assignee Information including Non-Applicant Assignee Information:

Providing assignment information in this section does not substitute for compliance with any requirement of part 3 of Title 37 of CFR to have an assignment recorded by the Office.	
Assignee 1	
Complete this section if assignee information, including non-applicant assignee information, is desired to be included on the patent application publication. An assignee-applicant identified in the "Applicant Information" section will appear on the patent application publication as an applicant. For an assignee-applicant, complete this section only if identification as an assignee is also desired on the patent application publication.	
Remove	
If the Assignee or Non-Applicant Assignee is an Organization check here. ☐	

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	009462.P001
		Application Number	
Title of Invention	Systems and Methods for Implementing an Enterprise-Class Converged Compute-Network-Storage Appliance		

Prefix	Given Name	Middle Name	Family Name	Suffix

Mailing Address Information For Assignee including Non-Applicant Assignee:

Address 1			
Address 2			
City		State/Province	
Country i		Postal Code	
Phone Number		Fax Number	
Email Address			

Additional Assignee or Non-Applicant Assignee Data may be generated within this form by selecting the Add button.

Signature:

NOTE: This form must be signed in accordance with 37 CFR 1.33. See 37 CFR 1.4 for signature requirements and certifications

Signature	/Eric S. Replogle/		Date (YYYY-MM-DD)	2013-12-19
First Name	Eric S.	Last Name	Replogle	Registration Number
			52161	

Additional Signature may be generated within this form by selecting the Add button.

This collection of information is required by 37 CFR 1.76. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 23 minutes to complete, including gathering, preparing, and submitting the completed application data sheet form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

Privacy Act Statement

The Privacy Act of 1974 (P.L. 93-579) requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C. 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether the Freedom of Information Act requires disclosure of these records.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (i.e., GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspections or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

UNITED STATES PATENT APPLICATION
FOR
SYSTEM AND METHODS FOR IMPLEMENTING AN ENTERPRISE-CLASS CONVERGED COMPUTE-
NETWORK-STORAGE INFRASTRUCTURE

INVENTOR:

SANDIP AGARWALA
ABHISHEK CHATURVEDI
SHRAVAN GAONKAR
MALLIKARJUNAN MAHALINGAM
SAZZALA REDDY
SMIT SHAH
FARAZ SHAIKH
PRAVEEN VEGULLA
KRISHNA YADAPPANAVAR
JEFFREY A. ZABARSKY

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
1279 OAKMEAD PARKWAY
SUNNYVALE, CA 94085-4040

(408) 720-8300

CERTIFICATE OF ELECTRONIC FILING

I hereby certify that this correspondence is being deposited with the United States Patent and Trademark Office via electronic filing through the United States Patent and Trademark Electronic Filing System on:

December, 19, 2013

Date of Deposit

Katie Nelson

Name of Person Mailing Correspondence

/ Katie Nelson /

Signature

December 19, 2013

Date

SYSTEM AND METHODS FOR IMPLEMENTING AN ENTERPRISE-CLASS CONVERGED COMPUTE-NETWORK-STORAGE INFRASTRUCTURE

RELATED APPLICATIONS

[0001] Applicant claims the benefit of priority of prior, co-pending provisional application Serial No. 61/739,685, filed December 19, 2012, the entirety of which is incorporated by reference.

FIELD OF INVENTION

[0002] This invention relates generally to a storage system and more particularly to software defined distributed storage system.

BACKGROUND OF THE INVENTION

[0003] Enterprise storage systems currently available are proprietary storage appliances that integrate the storage controller functions and the storage media into the same physical unit. This centralized model makes it harder to independently scale the storage systems' capacity, performance and cost. Users can get tied to one expensive appliance without the flexibility of adapting it to different application requirements that may change over time. For small and medium scale enterprise, this may require huge upfront capital cost. For larger enterprise datacenters, new storage appliances are added as the storage capacity and performance requirements increase. These operate in silos and impose significant management overheads.

SUMMARY OF THE DESCRIPTION

[0004] A software defined distributed storage system that separates the storage controller functions with that of storage media is described. In an exemplary embodiment, a storage controller client dispatches an input/output request to one of a plurality of storage controller servers in a software defined distributed storage system. The software defined distributed storage system further includes a plurality of virtual nodes distributed among a plurality of physical nodes, and each of the plurality of physical nodes is hosted on one of the plurality of storage controller servers. Each physical node may be a commodity server hardware with one or more physical storage media (Hard Disk Drive, Solid State Drive, Flash devices, etc.) for storing data and its metadata. Each virtual node is allocated some physical space in one or more physical storage media where its contents are stored with enough redundancy to recover from failures. The storage controller client receives the input/output request. The storage controller client further computes a target virtual node for the input/output request, where the target virtual node is one of the plurality of virtual nodes. Using the computed target virtual node, the storage controller client determines a target physical node that corresponds to the target virtual node, where the target physical node is one of the plurality of physical nodes. In addition, the storage controller client dispatches the input/output request to a target storage controller server that corresponds to the target physical node, wherein the target storage controller server is one of the plurality of storage controller servers.

[0005] In another embodiment, a storage controller server receives a write operation request to store data, where the write operation request is dispatched from a storage controller client to the storage controller server using a deterministic placement function. In response to the write operation request, the storage controller server appends the data to a write log in an active log partition of the storage controller server.

[0006] In a further embodiment, a storage controller server receives a read operation request, where the read operation request is dispatched from a storage controller client to the storage controller server using a deterministic placement function. In response to receiving the read

operation request, the storage controller server reads the data that is indicated in the read operation request, where at least part of the data is stored in a write log that is in an active log partition of the storage controller server.

[0007] Other methods and apparatuses are also described.

BRIEF DESCRIPTION OF THE DRAWINGS

- [0008] The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings in which like references indicate similar elements.
- [0009] Figure 1 is an illustration of one embodiment of a high-level view of StorFS system.
- [0010] Figure 2 is an illustration of one embodiment of a system including some of the components that comprises the storage controller server.
- [0011] Figure 3 is an illustration of one embodiment of an efficient distributed replication.
- [0012] Figure 4 is a flow diagram of one embodiment of a process for Active Mirror Set (AMS) management with strong consistency and high availability.
- [0013] Figure 5 is a flow diagram of one embodiment of a process for KV_put processing.
- [0014] Figure 6 is a flow diagram of one embodiment of a process for KV_get processing.
- [0015] Figure 7 is an illustration of one embodiment of the Block Management layer.
- [0016] Figure 8 is a flow diagram of one embodiment of a process to perform input/output processing.
- [0017] Figure 9 is a flow diagram of one embodiment of a process to perform synchronous mirroring.
- [0018] Figure 10 is a flow diagram of one embodiment of a process to asynchronously mirror data and metadata.
- [0019] Figure 11 is a flow diagram of one embodiment of a process to handle a failure of a component.
- [0020] Figure 12 is a flow diagram of one embodiment of a process to perform a distributed failure recovery.
- [0021] Figure 13 is a block diagram of one embodiment of an AMS module to process for Active Mirror Set (AMS) management with strong consistency and high availability.
- [0022] Figure 14 is a block diagram of one embodiment of a KV_put module to process a KV_put operation.

[0023] Figure 15 is a block diagram of one embodiment of a KV_get module to process a KV_get operation.

[0024] Figure 16 is a block diagram of one embodiment of an input/output dispatch module to dispatch an input/output request.

[0025] Figure 17 is a block diagram of one embodiment of a mirroring module to perform synchronous mirroring.

[0026] Figure 18 is a block diagram of one embodiment of an asynchronous module to asynchronously mirror data and metadata.

[0027] Figure 19 is a block diagram of one embodiment of a failure module to handle a failure of a component.

[0028] Figure 20 is a block diagram of one embodiment of a recovery module to perform a distributed failure recovery.

[0029] Figure 21 illustrates one example of a typical computer system, which may be used in conjunction with the embodiments described herein.

DETAILED DESCRIPTION

[0030] A software defined distributed storage system called StorFS that separates the storage controller functions with that of storage media is described. In the following description, numerous specific details are set forth to provide thorough explanation of embodiments of the present invention. It will be apparent, however, to one skilled in the art, that embodiments of the present invention may be practiced without these specific details. In other instances, well-known components, structures, and techniques have not been shown in detail in order not to obscure the understanding of this description.

[0031] Reference in the specification to “one embodiment” or “an embodiment” means that a particular feature, structure, or characteristic described in connection with the embodiment can be included in at least one embodiment of the invention. The appearances of the phrase “in one embodiment” in various places in the specification do not necessarily all refer to the same embodiment.

[0032] In the following description and claims, the terms “coupled” and “connected,” along with their derivatives, may be used. It should be understood that these terms are not intended as synonyms for each other. “Coupled” is used to indicate that two or more elements, which may or may not be in direct physical or electrical contact with each other, co-operate or interact with each other. “Connected” is used to indicate the establishment of communication between two or more elements that are coupled with each other.

[0033] The processes depicted in the figures that follow, are performed by processing logic that comprises hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general-purpose computer system or a dedicated machine), or a combination of both. Although the processes are described below in terms of some sequential operations, it should be appreciated that some of the operations described may be performed in different order. Moreover, some operations may be performed in parallel rather than sequentially.

[0034] The terms “server,” “client,” and “device” are intended to refer generally to data processing systems rather than specifically to a particular form factor for the server, client, and/or device.

[0035] A software defined distributed storage system called StorFS that separates the storage controller functions with that of storage media is described. In one embodiment, this separation allows the storage controller to be co-located with the user application in the same physical units or reside separately in a dedicated set of storage units. This set of storage units functions as a single unified storage system, and balances the capacity and performance load across the distributed components. A cache accelerated distributed storage design along with efficiency and reliability features can provide a higher level of determinism in terms of quality of service and performance. The ability to converge compute and storage significantly simplifies the management and reduces both the capital cost as well as the operational cost.

[0036] In one embodiment, the capacity and performance of a storage system can be increased in two ways: scale-up and/or scale-out. Scaling-up refers to adding more resources (e.g. storage, compute, etc.) in the same physical storage unit. In this embodiment, there can be a limit up to which a system can be scaled up. Thus, further scaling requires adding more physical units that work in cohesion to provide increased capacity and performance. This is called scale-out and has been the motivation of many distributed storage system designs. In an ideal scale-out system, adding the resources increases the capacity and performance by the same ratio without impacting the reliability characteristics of the storage system. In practice, however, this ideal scale-out is hard to achieve. This is because previous distributed systems rely on a centralized component for synchronization of the distributed system’s various tasks and managing/storing of the distributed system metadata. In one embodiment, this metadata is data that describes the distributed system and the location of users’ data. This centralized metadata becomes the bottleneck because the centralized metadata distributed storage system is not able to scale beyond a certain limit.

[0037] In one embodiment, the StorFS system is a storage system that includes a plurality of storage nodes coupled together by an interconnection network. In this embodiment, the StorFS system provides a distributed storage of data to one or more local and/or remote applications that make use of the storage and/or storage services provided the StorFS system. In one embodiment, each of the storage nodes includes a storage controller that is used to manage storage requests of the local and/or remote applications using the storage. The storage controller can include a storage controller client (SC client), a storage controller server (SC server), or both the SC client and SC server. In one embodiment, the SC client handles and dispatches application storage requests to an SC server. The SC server handles the storage requests and can additionally perform storage services (e.g., synchronous and asynchronous replication, distributed data deduplication, distributed garbage collection, data flushing, and distributed recovery).

[0038] In another embodiment, the StorFS system uses the concept of virtual node (vNode) as the unit of data routing and management. In one embodiment, there are four types of vNode:

- *Namespace vNode*: A vNode that is associated with the routing and storage of the StorFS file system namespace (directory structure, file names, etc.) metadata.
- *FileTree vNode*: A vNode that is associated with the routing and storage of the StorFS 'File Tree' metadata. File tree metadata is used to locate the data on the persistent storage.
- *Data vNode*: A vNode that is associated with the routing and storage of the data
- *Cache vNode*: A vNode that is associated with the routing of the client IOs from SC client to one of the SC servers. Multiple cache vNodes are grouped together to form an abstraction called 'Write Log Group', where updates from the clients are logged.

Namespace and FileTree vNodes are collectively called metadata vNodes. In one embodiment, there is a one-to-one correspondence between the metadata vNodes and the cache vNodes; i.e. for every metadata vNode, there exists one and only one cache vNode. In another embodiment, there can be a many to one relationship between metadata vNodes and cache vNodes (and vice versa). The number of metadata and cache vNodes is specified as a configuration parameter during the cluster creation and this number remains fixed for the entire lifecycle of the cluster. In

one embodiment, data vNodes are created on demand when new data is written to the StorFS system. For the purpose of resiliency and high availability, the StorFS system introduces enough redundancy (e.g., RAID-1) to recover from failures. The redundancy level is configurable based on user-defined policies. Cache vNodes can only be placed on storage devices that allow fast random access like SSD, flash, NVRAM, etc. Data and metadata vNodes can be placed on any non-volatile persistent storage like HDD, SSD, etc.

[0039] The cluster resource manager (CRM) maintains a database that maps vNodes to their ‘Mirror Set’. The mirror set defines the actual physical locations of the corresponding vNode.

$$V_i \rightarrow \{M_1, M_2 \dots M_k\}, \text{ where } M\text{'s are the } k \text{ way mirrors of vNode } V_i \quad (1)$$

$$M_i = \{P_x, D_y, S_z\}$$

In one embodiment, a mirror is defined by the tuple $\{P_x, D_y, S_z\}$ where P_x , indicates the physical node, D_y indicates the disk within that node and S_z denotes the “*Segment Stor*” within the disk. In one embodiment, a “*Segment Stor*” is a thin provisioned logical partition in StorFS. One of the ‘ M_i ’ in the mirror set is marked as ‘primary’ and is responsible for mirroring data/metadata to the other mirrors in the mirror set.

[0040] In one embodiment, the SC client dispatches storage requests from the applications to an SC server that stores the data referenced by the storage request. In this embodiment, the storage request includes an entity identifier that references the stored data and an offset that is an offset into that stored data. The entity identifier can be an iNode, a file descriptor, key-value pair identifier, etc. If the storage request is a write of data to the StorFS system, the SC client determines which of the plurality SC servers is to handle the write storage request. In one embodiment, the SC client computes the SC server to handle the write storage request using a deterministic placement function. In this embodiment, the deterministic placement function uses the input parameters to compute the virtual node that (or will) store the data referenced in the write storage request. In one embodiment, by using the deterministic placement function, the

StorFS system does not need to store location metadata to determine the initial virtual node for the data in the request. Instead, the system stores metadata for the virtual node to the physical node conversion. With the virtual node identification, in one embodiment, the SC client determines the physical node identification and the SC server that corresponds to this physical node.

[0041] The SC server that stores the data referenced by the write storage request, receives this request. In one embodiment, the SC server stores this data in a write log by appending the data at the end of the write log. This can lead to a fast write for the data that allows a quick write storage fulfillment for the requesting application. In addition and in one embodiment, the SC server can synchronously replicate the data referenced in the write storage request. In this embodiment, the SC server determines the mirrors for this virtual node using a dynamic table that tracks the mirrors for each virtual node. In addition, the SC server sends the data to the SC servers that are used for the mirror virtual nodes.

[0042] In one embodiment, when the write log is full, the SC server flushes the write log to persistent storage. In this embodiment, for each virtual node that stores data in the full write log, the StorFS system determines new virtual node for that data and sends the data to the SC server that corresponds to the new virtual node. The SC server stores this data in persistent storage. In addition, the StorFS system can determine if the data that is stored in the write log for that virtual node is a duplicate of some other data stored elsewhere in the StorFS system. In one embodiment, during the flushing operation, the StorFS system determines if an entity is stored elsewhere by using a deduplication hints table. In one embodiment, the hints table stores characteristics of the top-K storage entities in the StorFS system. If there is a match with the entity being flushed, the StorFS system updates the metadata for this entity, but does not flush the entity to the persistent storage. If not, the StorFS system flushes the entity to persistent storage.

[0043] In one embodiment, during a flush to persistent storage, the StorFS system can asynchronously replicate each virtual node in the write log. In this embodiment, the StorFS

system determines the mirrors of the virtual node and asynchronously sends the data and metadata of that virtual node to the mirror virtual nodes. In one embodiment, the StorFS system transfers the data and metadata of the virtual node as a large bulk transfer so that this data transfer is more efficient.

[0044] In addition to the write operations of the StorFS system outlined above, the StorFS system can perform additional storage services on the stored data. In one embodiment, the StorFS system can perform distributed failure recovery, distributed deduplication, and distributed garbage collection. In one embodiment, a virtual node can have one or more mirror nodes, where each of the mirror nodes is stored in a different fault domain. If a component of the virtual node goes down and/or becomes unresponsive, a new virtual node is allocated. The StorFS copies the data and/or metadata for the failed virtual node from the one or more mirror virtual nodes to the newly allocated virtual node. In one embodiment, the StorFS copies the data from the mirror nodes in large segment, making the restoration of that virtual node more efficient.

[0045] Furthermore, in one embodiment, the StorFS system can perform distributed data deduplication. As described above, during a flushing of data from the write log to persistent storage, the StorFS system checks to determine if the data being moved is duplicated elsewhere in the StorFS system. The StorFS system uses a hints table to determine whether data is duplicated. In one embodiment, the hints table stores statistics about a partial set of data in the StorFS system. By using a partial set of data statistics, the StorFS can do the data deduplication check efficiently. The StorFS system creates the hints table by creating a live object map of the data stored in the StorFS system. Using this map, the StorFS system determines a top-K statistics on the data stored in the StorFS system and uses these statistics in the hints table. In one embodiment, there is a global hints table that is used by each of the SC servers to determine if data is being duplicated in the StorFS system. In one embodiment, the hints table map stores a fingerprint for a VBA of content stored in the StorFS system.

[0046] In one embodiment, the StorFS system stores data in logically sequential segments. As data is deleted, these segments become unreferenced and can be reused. In one embodiment,

the StorFS system walks a metadata tree to determine which of the segments are not referenced. For each unreferenced segment, the StorFS system collects the unreferenced segments and makes the segments available for future use. In addition, these segments can be used for fast distributed recovery of a vNode that is hosted by component that fails. In one embodiment, if a component of the StorFS system fails, the StorFS system determines which vNodes are affected by component recovery. For each affected vNode, the StorFS system can either create a new mirror vNode and resync this new vNode or resync that affected vNode.

[0047] Figure 1 is an illustration of one embodiment of a high-level view of StorFS system 100. In Figure 1, the StorFS system 100 includes storage nodes 102A-C coupled by an interconnection network 116. While in one embodiment, three storage nodes 102A-C are illustrated as part of the StorFS system 100, in alternate embodiments, there can be more or less storage nodes. For example and in one embodiment, the StorFS system 100 can include up to several hundred storage nodes. In one embodiment, each storage node 102A-C includes a storage controller (SC) client (also called dispatcher) 108A-C, a storage controller (SC) server 110A-C, or both. The SC servers 110A-C manage their underlying storage (e.g., Hard disk drive (HDD), Solid state drives (SSD), PCIe flash, etc.) and collectively provide reliable and unified storage functionality to the SC clients. The SC client 108A-C processes input/output (I/O) requests from the applications that are local to its physical storage node and routes them to the appropriate SC servers for processing. For example and in one embodiment, SC client 108A can send an I/O request locally to SC Server 110A and/or remotely to SC Servers 110B or 110C. The system is capable of exposing many different interfaces to the application like file (e.g. NFS, CIFS), object, key-value, etc. In one embodiment, the storage node 102A-C can be server, blade server, personal computer, etc., or any other type of device capable of storing data. In one embodiment, the management server 112A-C is an agent that is used to communicate system management data and commands regarding the corresponding storage node 102A-C with the management client 114.

[0048] In one embodiment, the design of the StorFS system 100 distributes both the data and the metadata, and this system 100 does not require storing a complete global map for locating individual data blocks in our system. The responsibility of managing metadata is offloaded to each individual storage nodes 102A-C. In one embodiment, a cluster manager (CRM) resides on each SC Server 110 maintains some global metadata, which is small compared to the local metadata. In one embodiment, each logical file (or entity) is partitioned into equal sized “stripe units”. The location of a stripe unit is determined based on a mathematical placement function Equation (1):

$$\begin{aligned} Virtual_Node\# &= Hash(Entity_Id, Stripe_Unit\#) \% Total_Virtual_Nodes \\ Stripe_Unit\# &= \frac{offset}{Stripe_Unit_Size} \% Stripe_Unit_Per_Stripe \end{aligned} \quad (2)$$

The *Entity_{Id}* is an identification of a storage entity that is to be operated upon, the *Total_Virtual_Nodes* is the total number of virtual nodes in the StorFS system 100, the offset is an offset into the storage entity, and the *Stripe_Unit_Size* is the size of each stripe unit in the StorFS system 100. The value *Stripe_Unit_Per_Stripe* is described further below. In one embodiment, the storage entity is data that is stored in the StorFS system 100. For example and in one embodiment, the storage entity could be a file, an object, key-value pair, etc. In this example, the *Entity_{Id}* can be an iNode value, a file descriptor, an object identifier, key/value identifier, etc. In one embodiment, an input to a storage operation is the *Entity_{Id}* and the *offset* (e.g., a write, read, query, create, delete, etc. operations). In this embodiment, the *Entity_{Id}* is a globally unique identification.

[0049] In one embodiment, the StorFS 100 system receives the *Entity_{Id}* and *offset* as input for each requested storage operation from an application 106A-C. In this embodiment, the StorFS system 100 uses the *offset* to compute a stripe unit number, *Stripe_Unit#*, based on the stripe unit size, *Stripe_Unit_Size*, and the number of virtual nodes that the entity can be spread across, *Stripe_Unit_Per_Stripe*. Using the stripe unit number and the entity identifier (*Entity_{Id}*), the StorFS system 100 computes the virtual node identifier. As described below, the StorFS system 100 uses a hash function to compute the virtual node identifier. With the virtual node identifier,

the StorFS 100 can identify which physical node the storage entity is associated with and can route the request to the corresponding SC server 110A-C.

[0050] In one embodiment, each vNode is a collection of either one or more data or metadata objects. The StorFS system 100 does not store data and metadata in the same virtual node. This is because data and metadata may have different access patterns and quality of service (QoS) requirements. In one embodiment, a vNode does not span across two devices (e.g. a HDD). A single storage disk of a storage node 102A-C may contain multiple vNodes. In one embodiment, the placement function uses that a deterministic hashing function and that has good uniformity over the total number of virtual nodes. A hashing function as known in the art can be used (e.g., Jenkins hash, murmur hash, etc.). In one embodiment, the “*Stripe_Unit_Per_Stripe*” attribute determines the number of total virtual nodes that an entity can be spread across. This enables distributing and parallelizing the workload across multiple storage nodes (e.g., multiple SC servers 110A-C). In one embodiment, the StorFS system 100 uses a two-level indexing scheme that maps the logical address (e.g. offset within a file or an object) to a virtual block address (VBA) and from the VBAs to physical block address (PBA). In one embodiment, the VBA can be a tuple with following information:

$$\text{VBA} = \{\text{vNode ID, volume ID, object ID, offset, fingerprint}\}$$

In this embodiment, the VBA is prefixed by the ID of the vNode in which they are stored. This vNode identifier (ID) is used by the SC client and other StorFS system 100 components to route the I/O to the correct cluster node. The physical location on the disk is determined based on the second index, which is local to a physical node. The volume (also known as datastore) ID, object ID and offset are optional items. They are used to preserve the spatial locality of the VBAs. In one embodiment, a VBA contains the ‘fingerprint’ of the object fragment it represents and is unique across the StorFS cluster. No two objects in the cluster will have the same VBA.

[0051] In one embodiment, the cluster manager (CRM) maintains a database of virtual node (vNode) to physical node (pNode) mapping. In this embodiment, each SC client and server caches the above mapping and computes the location of a particular data block using the above

function in Equation (1). In this embodiment, the cluster manager need not be consulted for every I/O. Instead, the cluster manager is notified if there is any change in 'vNode' to 'pNode' mapping, which may happen due to node/disk failure, load balancing, etc. This allows the StorFS system to scale up and parallelize/distribute the workload to many different storage nodes. In addition, this provides a more deterministic routing behavior and quality of service. By distributing I/Os across different storage nodes, the workloads can take advantage of the caches in each of those nodes, thereby providing higher combined performance. Even if the application migrates (e.g. a virtual machine migrates in a virtualized environment), the routing logic can fetch the data from the appropriate storage nodes. Since the placement is done at the stripe unit granularity, access to data within a particular stripe unit goes to the same physical node. Access to two different stripe units may land in different physical nodes. The striping can be configured at different level (e.g. file, volume, etc.) Depending on the application settings, the size of a stripe unit can range from a few megabytes to a few hundred megabytes. In one embodiment, this can provide a good balance between fragmentation (for sequential file access) and load distribution.

[0052] In one embodiment, the distributed object layer provides a global distributed object namespace that is accessible over the network. At a very high level, distributed object layer provides two simple interfaces:

- KV_Put (key, object)
- object = KV_Get (key)

KV_Put and KV_Get are further described in Figures 5 and 6 below, respectively. In one embodiment, the key is the VBA that uniquely identifies the object and can be accessed from any storage control server in the cluster. In another embodiment, the key may be user-defined unique identifier. The distributed object layer routes the I/O requests to the storage control server that manages and stores the vNode indicated in the VBA. In one embodiment, the KV_Put interface is used to store an object referenced by the key. In one embodiment, the KV_Get interface is used to fetch an object referenced by the key.

[0053] Figure 2 is a block diagram of one embodiment of a storage control server 200. In one embodiment, the Block Management Layer 220 of the storage control server is responsible for formatting, allocating, and freeing storage in fixed block sizes. This layer provides access to different kinds of storage (e.g. SSD, HDD, etc.) in the system. In one embodiment, the Distributed Object Layer 218 of the storage control server uses an API of the Block Management Layer 220 to provide a global distributed object namespace that is accessible over the network. In one embodiment, the Storage Efficiency Layer 216 reduces the size of data footprint on the physical medium using techniques like compression, deduplication, etc. The reduction is achieved without deteriorating the performance or the reliability of the data storage. In one embodiment, the File System Layer 214 provides a logical global namespace abstraction to organize and locate data in the cluster. In one embodiment, the Data Service Layer 212 provides enterprise data services like disaster recovery, fine grained policy management, snapshots/clones, etc. In one embodiment, the Write Cache 208 and the Read Cache 210 Layers provide acceleration for write and read I/O respectively using fast storage devices. In one embodiment, the Write Cache Layers 208 includes the write log as described below. In one embodiment, the Presentation Layer 206 provides an interface to access the StorFS storage using well-known standard protocols like NFS, CIFS, REST, iSCSI, and/or another storage standard. In one embodiment, the Cluster Manager (CRM) Layer 202 is responsible for the coordination across distributed StorFS components, delegating responsibilities and maintaining a consistent global state of the system. In one embodiment, the Fault Tolerance Layer 204 is responsible for resiliency and making sure that the data is available and consistent even after the failure of a software or hardware component (disk, server, network, etc.). In one embodiment, the Garbage Collection Layer 222 is responsible for reclaiming dead space that result due to entities getting deleted or updated. This layer efficiently determines the storage blocks that are not used (or referenced) and makes them available for new data to be written. In one embodiment, the Storage Management Layer 224 provides a framework to configure, monitor, analyze and report on the operation of the overall StorFS cluster storage system as well as individual logical and

physical entities in the cluster. In one embodiment, each of the layers mentioned above are fully distributed and each layer does not rely on any centralized components for their operations.

[0054] Figure 4 is a flow diagram of one embodiment of a process 400 for Active Mirror Set (AMS) management with strong consistency and high availability. In one embodiment, process 400 is performed by the cluster resource manager 202 as described in Figure 2 above. In Figure 4, process 400 creates an AMS for the vNode (V_i). In one embodiment, the AMS is a mirror set for the vNode associated with the write log entry. In this embodiment, the AMS includes one or more mirrors that are used to replicate the data and metadata of that vNode. In one embodiment, the AMS for the vNode V_i is:

$$V_i \rightarrow \{M_1^{primary}, M_2^{active}, M_3^{active}\} \quad (3)$$

With this AMS for V_i , the primary mirror $M_1^{primary}$ handles the new I/O and replicates the I/O to the other mirrors in the AMS, M_2^{active} and M_3^{active} . At block 404, process 400 determines the type of mirror that failed (e.g., a primary or the non-primary mirror failure). In one embodiment, one of these mirrors may fail because of a disk failure, node failure and/or rack failure. If the primary mirror fails, at block 406, process 400 marks the failed mirror as an inactive mirror.

With the inactive mirror as part of the AMS, process 400 elects a new primary and updates the AMS for V_i . In this embodiment, the AMS for V_i will be:

$$V_i \rightarrow \{M_1^{inactive}, M_2^{primary}, M_3^{active}\} \quad (4)$$

In this embodiment, $M_2^{primary}$ handles the new I/O and replicates the new I/O to M_3^{active} . In addition, since the $M_1^{inactive}$ is inactive, $M_2^{primary}$ does not replicate the new I/O to the inactive mirror. Execution proceeds to block 410 below.

If a non-primary fails, at block 408, process 400 marks the failed mirror as inactive and updates the AMS for V_i . In this embodiment, the AMS for V_i will be:

$$V_i \rightarrow \{M_1^{primary}, M_2^{active}, M_3^{inactive}\} \quad (5)$$

In this embodiment, $M_1^{primary}$ continues to handle the new I/O and replicates the new I/O to M_2^{active} . In addition, since the $M_3^{inactive}$ is inactive, $M_1^{primary}$ does not replicate the new I/O to the inactive mirror. Execution proceeds to block 410 below.

[0055] At block 410, process 400 detects that the inactive mirror has come back up. In one embodiment, process 400 resynchronizes the inactive mirror with the current primary. In one embodiment, process 400 resynchronizes the inactive mirror by copying over segments missing in the inactive mirror from one or more active mirrors. For example and in one embodiment, process 400 resynchronizes the inactive mirror as described in Figure 12 below. In one embodiment, at the end of the mirror resync, process 400 marks the inactive mirror as active. In this embodiment, the updated AMS or V_i will be:

$$V_i \rightarrow \{M_1^{primary}, M_2^{active}, M_3^{active}\} \quad (6)$$

In this embodiment, at the end of the mirror resync, the inactive mirror is marked as active. The mirrors are in sync at this stage. In this embodiment, $M_1^{primary}$ handle the new I/O and replicates the new I/O to M_2^{active} and M_3^{active} .

[0056] As described above, the StorFS system provides an interface, KV_put, to store an object referenced by a key. Figure 5 is a flow diagram of one embodiment of a process 500 for KV_put processing. In one embodiment, process 500 is performed by the distributed object layer 218 as described in Figure 2 above. In Figure 5, process 500 begins by receiving the VBA and the object that is to be stored at block 502. In one embodiment, the VBA is the key used to reference the object. At block 504, process 500 determines the mirror set of the object's vNode. In one embodiment, the object's vNode is determined from the input VBA received at block 502. For example and in one embodiment, the VBA received includes the vNode as one of the elements of the tuple for the VBA.

[0057] Process 500 determines if the local pNode is the primary mirror of the determined vNode at block 506. In one embodiment, the local pNode can be determined if this pNode corresponds to the primary mirror based on the vNode for the VBA and a pNode/vNode correspondence table. If the local pNode is not the primary, process 500 sends the VBA and object to the primary mirror at block 508. In one embodiment, by sending the VBA and object, the target pNode can store the object at the VBA. Execution proceeds to block 512 below. If the local pNode is the primary mirror, store the VBA and object in the primary's segment store. In

one embodiment, the segment store is a “Segment Stor,” described above, that is a thin provisioned logical partition in StorFS. In one embodiment, the data in the mirrors is stored by segment, which allows for efficient storage of the content for that vNode. Execution proceeds to block 512 below.

[0058] At block 512, process 500 sends the VBA and object to the other mirrors. In one embodiment, process 500 sends the VBA and object to the active mirrors of the AMS of the vNode for the object. By sending the VBA and object to the active mirrors, process 500 provides data replication for each object store. Process 500 determines if any of the non-primary mirrors failed in response to the VBA and object being sent to them. If a non-primary mirror failed, process 500 marks that non-primary mirror as inactive. In one embodiment, the StorFS system attempts to recover the inactive mirror as described above. Execution proceeds to block 518 below. If none of non-primary mirrors fail, execution proceeds to block 518 below.

[0059] At block 518, process 500 determines if the primary mirror failed. If the primary mirror did not fail in response to the store attempt, process 500 returns success to the client at block 524. If the primary mirror did fail, at block 520, process 500 determines if the error is recoverable. An error is recoverable if there is an active mirror other than the failed primary available. If the error is recoverable, execution proceeds to block 504, in which process 500 repeats the put process. If the error is not recoverable, process 500 returns a failure to the client.

[0060] As described above, the StorFS system can additionally provide an interface, KV_get, to fetch an object referenced by a key. Figure 6 is a flow diagram of one embodiment of a process 600 for KV_get processing. In one embodiment, process 600 is performed by the distributed object layer 218 as described in Figure 2 above. In Figure 6, process 600 begins by receiving the VBA for the object that is to be retrieved at block 602. In one embodiment, the VBA is the key used to reference the object. At block 604, process 600 determines the mirror set of the object’s vNode. In one embodiment, the object’s vNode is determined from the input VBA received at block 602. For example and in one embodiment, the VBA received includes the vNode as one of the elements of the tuple for the VBA.

[0061] Process 600 determines if the local pNode is the primary mirror of the determined vNode at block 606. In one embodiment, the local pNode can be determined if this pNode corresponds to the primary mirror based on the vNode for the VBA and a pNode/vNode correspondence table (e.g., the CRM routing table). If the local pNode is not the primary, process 600 sends the request for object to the primary mirror at block 608. Execution proceeds to block 612 below. If the local pNode is the primary mirror, process 600 fetches the object from the primary's segment store. In one embodiment, the segment store is the uniform framework to manage different kind of storage devices and abstracts out the complexity of low-level block management from the upper layers in the StorFS software stack. Execution proceeds to block 612 below.

[0062] At block 612, process 600 determines if the fetch was successful. In one embodiment, process 600 determines if the fetch is successful by examining a return code associated with fetch attempt. If the fetch is successful, process 600 returns the object to the client at block 614. If the fetch is not successful, process 600 attempts a read repair at block 616. In one embodiment, read repair involves fetching data from an active mirror and 'repairing' the primary mirror with the fetched data. Process 600 determines if the read repair is successful at block 618. If the read repair is successful, the object is fetched and process 600 returns the object to the client at block 614 as described above.

[0063] If the read repair is not successful, at block 620, process 600 determines if the error is recoverable. In one embodiment, process 600 determines that the error is recoverable by determining whether there is an alternative active mirror from where data can be fetched. If the error is recoverable, process 600 marks the primary mirror as inactive and elects a new primary at block 624. In one embodiment, by electing a new primary, process 600 can use this primary to do another fetch of the object. Execution proceeds to block 604 above in which process 600 attempts the fetch using the newly elected primary. If the error is not recoverable, process 600 returns a failure at block 622.

[0064] Figure 8 is an illustration of one embodiment of a process 800 to perform input/output processing. In one embodiment, process 800 is performed by the Storage Controller 104 of Figure 1. In Figure 8, process 800 begins by receiving an input/output request from a client at block 802. In one embodiment, the client can be an application that uses the StorFS system, such as applications 86A-D as described in Figure 1 above. In one embodiment, the I/O request can be a request to read content from the StorFS system, write content to the StorFS system, and/or request status of content stored in the StorFS system. At block 804, process 800 validates the input/output request. In one embodiment, validation may comprise of simple I/O attributes consistency check, permission checks, admission control, etc. If the request is not validated, at block 806, process 800 return an error to the requesting client (e.g., error code, throws an exception, etc.). If the request is validated, at block 808, process 80 computes the vNode for the input/output request using the placement function. In one embodiment, process 800 uses Equation (1) described above to compute the vNode for the I/O request. In addition, process 800 determines the pNode corresponding to the determined vNode. In one embodiment, process 800 determines the corresponding pNode using the CRM routing table as described above with reference to Figure 2 above.

[0065] At block 810, process 800 sends the input/put request to SC server on the corresponding pNode. For example and in one embodiment, if process 800 executing on SC client 88A on physical server 82A and the corresponding pNode is on the SC Server 18C of physical server 82C, process 800 sends the input/output request to the SC server 18C on physical server 82C. At block 812, process 800 determines if the input/output request is related to creating or deleting an entity. In one embodiment, an entity is data that is stored in the StorFS system 80. For example and in one embodiment, the entity could be a file, an object, key/value pair, etc. If the request is not related to creating or deleting an entity, execution proceeds to block 816 below. If the I/O request is related to creating or deleting an entity, at block 814, process 800 determines which of the vNodes need to be updated. In one embodiment, process 800 determines the vNodes to be updated by computing all the vNodes that the entity can reside using the placement

function (Equation (1)). At block 818, process 800 sends the input/output request to the SC server that corresponds to the vNodes determined at block 814.

[0066] At block 816, process 800 receives an I/O request for write log processing. Process 800 determines if the write log processing is complete at block 820. If the write log processing is not complete, at block 822, process 800 wait for the write log processing to complete and execution proceeds to block 820. If the write log processing is complete, process 800 returns.

[0067] In one embodiment, an application can use some type of middleware libraries to access networked storage. For example and in one embodiment, virtual machines may use the hypervisor's built-in NFS client or iSCSI client to access backend storage. In a clustered environment, the goal of this middleware is not only to provide connectivity to backend storage but also availability by failing over to other cluster nodes in case of failure. In one embodiment, the StorFS system uses an intelligent I/O dispatcher that intercepts I/O requests from the client (e.g. NFS Client running inside a hypervisor) and directs them to appropriate cluster nodes. In one embodiment, the I/O dispatching is performed by a SC Client, such as SC Client 108A-C as described in Figure 1 above. In one embodiment, there are three modes of operation: i) I/O requests are routed to the SC Client running on the same physical node (i.e. to 'localhost'); ii) A common FQDN (Fully Qualified Domain Name) entry for the cluster pointing to the 'localhost' or '127.0.0.1' is created in the DNS configuration file (typically /etc/hosts) of each cluster nodes; and iii) Intercepting and rerouting incoming I/O requests using software-based routing mechanism like DVS (Distributed Virtual Switch) and "Network Service Insertion" framework available in many popular virtualization platforms (e.g. VMware).

Log structured sequential storage layout

[0068] In one embodiment, the Log structured sequential storage layout provides an efficient use of SSD and HDD I/O bandwidth available for incoming writes. In this embodiment, both the data and metadata are written to a new disk location as opposed to rewriting them at their old location. This layout transforms the incoming random I/O traffic to large sequential writes in the

backend of the physical storage media (e.g., HDD, SSD, flash, etc.). In addition, this layout not only increases the write performance, but also enables many kinds of data services, provides protection from accidental deletes, etc. This layout additionally increases the SSDs lifetime by writing in large erase cycle blocks. However, a problem with the log structured layout is fragmentation of the physical storage media. This may not be a big problem in SSDs because of their high random I/O performance, but fragmentation can deteriorate performance in case of HDD spindles. In addition, Log structured log enables variable-sized chunks to be stored in it. This enables compression and deduplication of data blocks and stores them tightly on the storage media without any alignment restriction (like sector alignment, etc.).

[0069] As mentioned earlier, the StorFS system uses a two-level indexing scheme that maps the logical address to a virtual block address (VBA) and from the VBAs to physical block address (PBA). The system allows for objects with contiguous VBAs to be stored close to each other. This addressing scheme further enables fast lookup of objects from the underlying media and reduces defragmentation significantly. As described above, VBAs are unique across the StorFS cluster and no two objects in the cluster will have the same VBA.

[0070] Figure 7 is an illustration of one embodiment of the Block Management layer 700. In Figure 7, the detailed software stack of the Block Management Layer 700 is illustrated. The Block Storage API provides a standardized interface to create a “Block Set” 704, allocate/free blocks within a Block Set 704, perform IO to blocks, etc. This is accomplished by invoking an internal block driver API. Block drivers can provide different behaviors and, if the bottom of the driver itself calls the standard Block Storage API, drivers may be stacked one on top of another. For example, in Figure 7, block API 712D is stacked upon Block Partition Driver 712C, which is stacked upon another Block API 712B and Block File Driver 712A. For freeing an HDD block 714A, the block partition driver 712C marks the freed block as “free pending” and invokes the API provided by the block file driver 712A to return the freed block. The block file driver, in this case, does no operation (NOOP 714C) on it and returns success to the partition driver, which

finally marks the block as “free acknowledged” 714D and returns back to the client that invoked the original request.

[0071] When creating/opening a Block Set, a hierarchical path can be specified prepended with the string “<driverName>:” to indicate which driver should be used. If no driver name is specified, the Block File (“file:”) driver 712A is assumed and the path is treated as a standard Unix file or device. The Block File driver splits the file into fixed sized blocks and exports a single Block Set with all physical blocks allocated with physical block numbering (0 - (numBlocks - 1)). The Block Partition driver (“part:”) 712C can be stacked on top of the Block File 712A driver to allow a single physical disk Block Set to be partitioned into multiple Block Sets.

[0072] In one embodiment, the Block Flash driver (“flash:”) 718C can be stacked between the Block Partition 718E and File 718A drivers to provide flash specific semantics such as I/O scheduling (e.g. limiting the number of concurrent writes, preventing concurrent reads/writes, etc.) and invoking “TRIM” on block free. In one embodiment, freeing a flash-based block 720A is similar to freeing an HDD-based block 714A, except that, in this embodiment, the block flash driver gets invoked for the free request, which performs a “SCSI TRIM” 720C before invoking the block file driver. In one embodiment, a TRIM releases and prepares the specified region of a flash-based device for future writes.

[0073] In one embodiment, the segment store uses the block abstraction to build collections of ordered segments. Each segment within a Segment Store is uniquely identified by its ‘logical id’, which is a monotonically increasing 64-bit number. This provides a uniform framework to manage different kind of storage devices and abstracts out the complexity of low-level block management from the upper layers in the StorFS software stack.

[0074] In one embodiment, both the data and metadata are distributed as well as replicated, which allows for recovery in case of failure. In one embodiment, the replication is performed across different ‘fault domains’ so that a failure of a component in one domain does not impact the availability of components in other domains. A set of HDDs, server nodes, or a physical

server rack is a few examples of fault domain. This allows the StorFS system to tolerate different kind of failures: disk failures, node failures and/or rack failures. Replication of data, however, can impose a cost on a storage system. For example, replication can impose significant overhead in terms of capacity and steady state performance.

[0075] Figure 3 is an illustration of one embodiment of an efficient distributed replication. In Figure 3, the system 300 uses a combination of synchronous and asynchronous techniques to efficiently replicate data across the cluster. In one embodiment, storage nodes of SC servers 302A-C can each receive an incoming client write 304A-C. In one embodiment, the storage node 302A receives the incoming client write with synchronous replicate module 306A. The synchronous replicate module 306A receives the write 304A and stores the contents of the write in the high-speed storage 312A (e.g., NVRAM, Flash, SSD, etc.). In one embodiment, the synchronous replicate module 306A stores the write contents at the backend of the high-speed storage 312A as described above. A flusher module 308A flushes the contents of the write to the asynchronous replication module 310A. In one embodiment, the flusher module 308A flushes the contents on to the persistent storage.

[0076] In one embodiment, the asynchronous replication module 310A stores the content to persistent storage 314A. In this embodiment, the asynchronous replication module 310A can additionally replicate this content to the other storage nodes 302B-C via the asynchronous replication modules 310B-C, respectively. These receiving asynchronous replication modules 310B-C, each store the content in the corresponding local perspective storage 314B-C, respectively.

[0077] To facilitate replication, the block storage layer generates a unique monotonically increasing logical ID for the data/metadata segment system 300 writes. The replication engine uses these logical IDs to synchronize large chunks of segments across different mirrors. This scheme eliminates the need of expensive distributed coordination, metadata comparison, and/or read-modify-write to replicate data and metadata. New incoming writes are logged and synchronously replicated in SSDs or other high-speed storage medium. Periodically, a ‘flusher’

308A-C moves the logged content asynchronously from the high-speed storage to its permanent location for persistence.

[0078] As described above, the StorFS system can perform synchronous replication of the data being stored in the in the write log. Figure 9 is a flow diagram of one embodiment of a process 900 to perform synchronous replication of a write log entry. In one embodiment, process 900 is performed by process 500 at block 512 of Figure 5 above to perform this synchronous replication. In Figure 9, process 900 begins by receiving the write log entry at block 902. In one embodiment, the write log entry is an entry to store an entity in the write log. At block 904, process 900 gets an active mirror set (AMS) for the write log entry. In one embodiment, the AMS is a mirror vNode set for the vNode associated with the write log entry. In this embodiment, the AMS includes this vNode and one or more active vNodes that is used to replicate the data and metadata of that vNode. Process 900 prepares the write log entry at block 906. In one embodiment, the write log entry consists of vNode ID, timestamp, checksum and other metadata along with the incoming data from the SC client.

[0079] At block 908, process 900 synchronously sends the write log entry to the pNodes that correspond to the mirror vNodes. In one embodiment, process 900 determines the corresponding SC server for each mirror vNode by looking up the corresponding pNodes in a vNode to pNode mapping table via the CRM. In one embodiment, the write log entry is synchronous so that the success of the writing to the write log can be acknowledged to the requesting client. In this embodiment, a synchronous write allows that the data is reliably stored before write is acknowledged to the client. At block 910, process 900 receives an acknowledgement from each pNode. While in one embodiment, there is one SC server for each pNode, in alternate embodiments there can be more than one SC server for each pNode. In one embodiment, the acknowledgement indicates the write log entry have been stored at the corresponding pNode.

[0080] As described above, the flusher asynchronously sends data and metadata to the target vNode and the mirror nodes to replicate the data and metadata being moved to persistent storage. Figure 10 is a flow diagram of one embodiment of a process 1000 to asynchronously replicate

data and metadata. In Figure 10, process 1000 begins by receiving the data and the metadata, for each target vNode for the asynchronous replication at block 1002. In one embodiment, the asynchronous replication is on a vNode by vNode basis. In this embodiment, the asynchronous replication can include a replication of one or more vNodes. In one embodiment, each vNode may contain a few bytes or up to several megabytes or greater.

[0081] At block 1004, process 1000 determines an AMS for the target vNode. In one embodiment, the AMS for a target node includes the target vNode and one or more active replica vNodes that are used to store the replication data and metadata for the target vNode. The AMS is described further below. Process 1000 asynchronously sends the data and metadata for that vNode to the target vNode and other mirror nodes in the AMS at block 1006. In one embodiment, an asynchronous send of the data is one where the send is sent to the SC server that corresponds to the vNode receiving the data and control is returned to process 1000 before the receiving SC server completes the write. In one embodiment, the receiving SC server receives the data and metadata and stores this content in the persistent storage of the SC server. After this storing of the data, the SC server sends an acknowledgement to process 1000. At block 1008, process 1000 sends an acknowledgement that the data and metadata has been stored. In one embodiment, process 1100 completes when the process has received the acknowledgements for each of the issued writes.

[0082] In a typical replication based system, strong consistency is obtained by enforcing the following:

$$(R+W) > N \quad (4)$$

where 'R' and 'W' are the minimum number of replicas that must respond correctly to a 'read' and 'write' respectively, N is the total number of replicas. The value of N does not change once it has been set. This implies that if there are three replicas, R+W must always be greater than 3. In one embodiment, the StorFS system relaxes this requirement and yet provides strong consistency guarantee. For each vNode, the CRM maintains an Active Mirror Set (AMS), which defines the currently active replicas for that vNode. If a replica goes out of sync (e.g. because of

failure, etc.), the replica is dropped from AMS. If a replica comes back up, the replica syncs with the corresponding replicas in the AMS and is added back to AMS upon full resynchronization (as referred to as a “resync”). The modified requirement for strong consistency in StorFS is:

$$(R+W) > |AMS| \quad (5)$$

where $|AMS|$ is the cardinality of the AMS corresponding to a vNode. Thus the, variable sized AMS can provide greater service availability because of the less stringent requirements and without compromising strong consistency guarantees. The AMS is further described in Figure 4 below.

[0083] In one embodiment, the StorFS system is designed to handle multiple failures. When such a failure happens, the StorFS system recovers from multiple sources in parallel, which significantly reduces the total recovery time. This distributed recovery is made possible in part due to our placement logic that enables the mirrors to find different location of the replicas and stream large chunk of data from those replicas. Also, if a particular replica lags behind the “primary” replica because of failures, system reboot, etc., the StorFS system can resynchronize itself with other active replicas by fetching those segments whose logical IDs are greater than the ones it has. In this embodiment, there is no need to perform an expensive “diff-and-merge” operation across the distributed set of replicas.

[0084] Figure 11 is a flow diagram of one embodiment of a process 1100 to handle a failure of a component. In one embodiment, process 1100 is performed by the fault tolerance layer 204 of an SC server 200 as described in Figure 2 above. In one embodiment, the failure recovery is performed by each SC server in the StorFS system, which allows for a distributed failure recovery. In this embodiment, the StorFS system is not dependent on a single node for failure recovery. In Figure 11, process 1100 begins by monitoring the StorFS system at block 1102. In one embodiment, process 1100 monitors the storage cluster of the StorFS system for failures of any one of the components. A failure of the component in the StorFS system can be an entire

node (e.g., a storage node) or a component of the node (e.g., disk, network, controller, etc.). At block 1104, process 1100 determines if a heartbeat of component failed or if an I/O operation failed. In one embodiment, process 1100 could detect a component failure by determining that a heartbeat of component is no longer reporting. For example and in one embodiment, heartbeats are small messages that are exchanged between the storage nodes. If a storage node fails to receive heartbeat message from its peer node, it indicate a potential failure of that peer node. In another embodiment, process 1100 detects an I/O failure if a component of the SC server 200 that is performing process 1100 detects an I/O failure, such as a failure to write an entry to a write log, performing a synchronous replication, performing an asynchronous replication, sending content to persistent storage, monitoring, garbage collection, deduplication, etc.

[0085] If there is no heartbeat or I/O failure is detected, execution proceeds to block 1102. If there is a heartbeat or an I/O failure, at block 1106, process 1100 determined the failed component related to the heartbeat or I/O failure detect at block 1104. In one embodiment, the failed component could be the SC server performing process 1100 or could be a remote SC server. In one embodiment, the failed component could be failed server of hardware component (e.g., disk, network, server, etc.).

[0086] Based on the failed component, process 1100 determines the vNodes impacted by the component failure at block 1110. In one embodiment, process 1100 uses the CRM table to determined which vNodes are impacted by the component failure. In this embodiment, process 1100 determines the whether the entire pNode has failed or a component within a pNode (e.g., disk, controller, network interface, etc.). If pNode fails, all the vNodes in that pNode are impacted. If a pNode component fails, one or some of the vNodes associated with that component are impacted. In one embodiment, the CRM table includes information mapping vNodes to the pNode as well as which vNode is dependent on which component of the pNode (e.g., which disk stores which vNode, etc.). There can be one or more vNodes impacted by the failed component. Process 1100 additionally performs a processing loop (blocks 1112-1118) to perform a distributed recovery of the vNodes that correspond to the failed component.

[0087] At block 1114, process 1100 updates the AMS of that vNode. In one embodiment, process 1100 marks that vNode as passive, which indicates that the vNode is not an active vNode of the AMS. In one embodiment, a passive vNode is a node that is not available to perform I/O operations. In one embodiment, a passive vNode attempts to catch up via the resync operation as described in Figure 12 below. In this embodiment, once a passive vNode is resynced, the vNode becomes an active vNode. In addition, process 1100 marks the vNode as a candidate for recovery. In one embodiment, recovering a vNode consists of copying data from other active vNode in the passive vNode's AMS, so that the passive vNode can be brought up to date with the current content for that vNode. Process 1100 determines if the number of active mirror vNodes in the AMS for that passive vNode is greater than zero. If the number of active node in the AMS is greater than zero, execution proceeds to block 1102 above. In one embodiment, the passive vNode is marked for recovery and recovery of that vNode happens synchronously as described in Figure 13 below. If the number of active nodes if is zero, at block 1118, process 1100 starts the StorFS recovery. In one embodiment, the StorFS recover is further described in Figure 12 below. At block 1120, the process 1100 completes the recovery.

[0088] Figure 12 is a flow diagram of one embodiment of a process 1200 to perform a distributed failure recovery of one or more vNodes. In Figure 12, process 1200 begins by performing processing loop (blocks 1202 – 1218) to perform a distributed recovery for each vNode. At block 1204, process 1200 determines if the vNode is marked for recovery. If the vNode is not marked for recovery, execution proceeds to block 1202 above. In one embodiment, a vNode marked for recovery is a vNode that was or is associated with a failed component and the content of that vNode may be out of sync with the content that is to be stored on that vNode. For example and in one embodiment, a SC server could fail and the vNode stored on that SC server would be marked for recovery. If the vNode is marked for recovery, execution proceeds to block 1206.

[0089] At block 1206, process 1200 determines if the number of active mirrors in the AMS for this vNode is greater than the threshold. In one embodiment, if the number of active mirror is

large enough, then that vNode is not necessary for that vNode to be recovered. This means that there are enough active mirrors for the content that is stored in AMS vNode set. If the number of mirrors in the AMS for this vNode is greater than the threshold, execution proceeds to block 1202 above and this vNode is not recovered. If there are enough active vNodes, the passive vNode remains marked for recovery and recovery happens at a later time. If the number of mirrors in the AMS for this vNode is less than or equal to the threshold, process 1200 determines if the node hosting the vNode is alive. In one embodiment, process 1200 could send a heartbeat message to the node to determine if the node is alive. For example and in one embodiment, this node may have temporarily failed and restored itself and is now back up. If the node hosting vNode is alive, execution proceeds to block 1212, where process 1200 starts the resync process for this vNode.

[0090] If the node hosting the vNode is not alive, at block 1210, process 1200 creates a new passive mirror vNode for this vNode. In one embodiment, this new passive mirror vNode is used for recovery. In one embodiment, process 1200 requests from the CRM to create the passive mirror vNode. In this embodiment, the CRM determines a placement of the new mirror vNode based on the fault domain of the failed vNode. For example and in one embodiment, the CRM places that new mirror vNode in a fault domain that is different from the fault domain of the failed vNode and other vNodes in the AMS set of the failed vNode. Process 1200 further marks this new mirror vNode as passive. In this embodiment, the new passive vNode is the node used for the resync at block 1212 below. Execution proceeds to block 1212 below.

[0091] At block 1214, process 1200 performs a resync for the passive vNode. The passive vNode to be resynced could be the original vNode marked for recovery or a new passive mirror vNode created for the resync. In one embodiment, the resync operation occurs by the passive node hosting the passive mirror vNode sending requests to one, some, or all of the active mirrors in the AMS set for the passive vNode. In one embodiment, the passive mirror fetches the segments from other active mirrors whose ID is higher than its (passive mirror's) highest segment ID. In one embodiment, the data in the mirrors is stored by segment, which allows for

efficient storage of the content for that vNode. In one embodiment, the passive vNode that is being resync may have some the content already stored (e.g., if a node that hosts the failed vNode fails and recovers, this failed vNode may have some or most of the content and is missing the most recent content). In this embodiment, the failed vNode will need the just the missing content. This missing content is identified by the segment ID. By comparing segment ID and not using “diff-and-copy” approach, the resync process is more efficient because comparison of segment IDs is a more efficient process than comparison of the actual stored content. In addition, the segment IDs are stored sequentially, which allows the resync process to determine which is the latest segment ID successfully stored on the passive vNode. Once that is determined, the resync process can copy segment ID greater than the last successful segment ID to the passive vNode. In one embodiment, the resync process can copy content from different active mirror vNodes. By resyncing using segments, the resync process can use bulk transfers to copy content in logical segment ID order.

[0092] At block 1216, process 1200 checks if the active mirror vNodes and the passive vNode being resync have the same highest segment ID. If these vNodes do have the same highest segment ID, the resync process for this passive vNode is complete and execution proceeds to block 1218. At block 1218, process 1200 marks the passive vNode as active and adds the passive vNode into the AMS. Execution proceeds to block 1202 above, where another vNode is resynced. If these vNodes do not have the same highest segment ID, execution proceeds to block 1214 above.

[0093] In one embodiment, the StorFS system distributes a storage workload across the cluster by stripping it across multiple virtual nodes. Stripping achieves capacity and performance load balancing and provides higher cumulative performance rather than being bottlenecked by one cluster node. One of the characteristics as well as the challenges in a hybrid storage/compute cluster is the constant change in the StorFS systems’ behavior. For example, an application’s requirements may change; workloads or resources may be added or removed to/from the cluster; and/or there can be a failure in the StorFS system (e.g. a network failure, disk failure, server

failure, etc.) that may change the cluster's resource availability. The StorFS system is designed to continuously adapt to these kinds of changes by automatically balancing load across available resources. These resources are of different kinds: CPU, network, memory, storage capacity, storage bandwidth, etc.

[0094] In one embodiment, the StorFS system performs a multi-dimensional optimization across this heterogeneous set of resources. This is facilitated by estimating the requirements of each virtual node, estimating resource availability in each cluster nodes and migrating one or more virtual nodes from one physical node to the other. For example and in one embodiment, if the network becomes a bottleneck in a physical node (pNode), one or more metadata vNodes in that pNode can be relocated to another pNode that has more network capacity available. Similarly, if a pNode runs out of disk capacity, a new data vNode can be created on other pNodes or a vNode can be relocated to another pNode with more disk capacity. In one embodiment, this migration is achieved by creating a new replica of the source vNode at the destination pNode. The vNode at the destination pNode is populated in parallel from the replicas of the source vNode. While the destination vNode is being updated, the source vNode may change due to new I/O from the client. This new I/O is be appended to the source vNode and assigned new logical sequence ID. The destination vNode will continuously sync from the source based on these sequence IDs. Once they are in sync, the vNode to pNode mapping in the Cluster Manager's routing table is updated so that I/Os are routed to the new location. This ensures that the storage service remains uninterrupted when a vNode is being moved. This load distribution happens on a continuous basis and ensures that resources are utilized optimally and each workload receives the resource it need.

[0095] In one embodiment, the StorFS system is capable of supporting different policies at different level of granularity: at the file system level, at the individual file level, etc. These policies specify the performance, reliability, and other quality of service requirements of the applications. The system translate these high level polices to low-level system configuration. For example, the StorFS system is able to automatically select the type of storage medium (SSD v.

HDD) based on application performance requirement. Similarly, based on these policies, the StorFS system can decide whether to compress or de-duplicate the data, the level of mirroring, etc.

[0096] Data services like snapshot, cloning, backup, disaster recovery, etc., can be critical for any enterprise storage systems. A naïve way to implement such data services is to perform a full copy of the current storage. This is, however, very inefficient and takes lot of time and resources. More advanced storage systems implement these data services by keeping track of changes since the clones (or snapshots) were created instead of full copy. This consumes less storage resources. Single node data services are relatively easier to build, as the metadata is readily available in one central place. Building efficient data services in a distributed storage system poses significant challenges.

[0097] In one embodiment, the StorFS system uses techniques to efficiently track changes in a distributed setting. For example and in one embodiment, the StorFS is capable of performing low overhead data services at different level of granularity: from the file system level to the individual files level. As discussed above, the StorFS system stripes file and file system data across multiple storage nodes in our cluster. Snapshotting a file, for example, involves creating a small metadata entry in the storage vNodes containing that file. This snapshot process is extremely fast and require minimal amount of resources. There is no data copy involved. Furthermore, the StorFS permits creating large number of snapshots (or clones) without any deterioration in performance.

[0098] Figure 13 is a block diagram of one embodiment of an AMS 1300 module to process for AMS management with strong consistency and high availability. In one embodiment, the AMS module 1300 is part of the cluster resource manager 202 that performs the AMS management process 400 as described in Figure 4 above. In one embodiment, the AMS module 1300 includes a mark mirror inactive module 1302, mirror fail type module 1304, primary mirror fail module 1306, non-primary mirror module 1308, and inactive mirror up module 1310. In one embodiment, the mark mirror inactive module 1302 marks a mirror as

inactive as described in Figure 4, block 402 above. The mirror fail type module 1304 determines the type of mirror that failed as described in Figure 4, block 404 above. The primary mirror fail module 1306 processes a primary mirror failure as described in Figure 4, block 406 above. The non-primary mirror module 1308 processes a non-primary mirror failure as described in Figure 4, block 408 above. The inactive mirror up module 1310 handles an inactive mirror coming back up as described in Figure 4, block 410 above.

[0099] Figure 14 is a block diagram of one embodiment of a KV_put module 1400 to process a KV_put operation. In one embodiment, the KV_put module 1400 is part of the distributed object layer 218 that performs the KV_put process as described in Figure 5 above. In one embodiment, the KV_put 1400 module includes a receive input module 1402, determine vNode module 1404, local pNode module 1406, send to primary module 1408, store input module 1410, send to non-primary module 1412, non-primary fail module 1414, mark mirror inactive module 1416, primary mirror fail module 1418, error recoverable module 1420, and return module 1422. In one embodiment, the receive input module 1402 receives the input for the KV_put operation as described above in Figure 5, block 502 above. The determine vNode module 1404 determines the vNode for the KV_put operation as described above in Figure 5, block 504 above. The local pNode module 1406 determines if the local pNode is a primary mirror as described above in Figure 5, block 506 above. The send to primary module 1408 sends the input to the primary mirror as described above in Figure 5, block 508 above. The store input module 1410 stores the object in the primary mirror's segment store as described above in Figure 5, block 510 above. The send to non-primary module 1412 sends the input to the non-primary mirrors as described above in Figure 5, block 512 above. The non-primary fail module 1414 determines if any of the non-primary mirrors have failed as described above in Figure 5, block 514 above. The mark mirror inactive module 1416 marks the non-primary mirror as inactive as described above in Figure 5, block 516 above. The primary mirror fail module 1418 determines if the primary mirror has failed as described above in Figure 5, block 518 above. The error recoverable module 1420 determines if the error is recoverable as

described above in Figure 5, block 520 above. The return module 1422 returns the result as described above in Figure 5, blocks 522 and 524 above.

[00100] Figure 15 is a block diagram of one embodiment of a KV_get module 1500 to process a KV_get operation. In one embodiment, the KV_get module 1500 is part of the distributed object layer 218 that performs the KV_get process as described in Figure 6 above. In one embodiment, the KV_get module 1500 includes a receive request module 1502, determine vNode module 1504, local pNode module 1506, send request to primary module 1508, fetch module 1510, read repair module 1512, non-primary fail module 1514, mark primary inactive module 1516, and return module 1518. In one embodiment, the receive request module 1502 receives the request for the KV_get operation as described above in Figure 6, block 602 above. The determine vNode module 1504 determines the vNode for the KV_get operation as described above in Figure 6, block 604 above. The local pNode module 1506 determines if the local pNode is a primary mirror as described above in Figure 6, block 606 above. The send request to primary module 1508 sends the KV_get request to the primary mirror as described above in Figure 6, block 608 above. The fetch module 1510 fetches the requested object as described above in Figure 6, block 610 above. The read repair module 1512 performs a read repair as described above in Figure 6, block 616 above. The error recoverable module 1514 determines if the error is recoverable as described above in Figure 6, block 620 above. The mark primary inactive module 1516 marks the primary as inactive as described above in Figure 6, block 624 above. The return module 1518 returns success or failure as described above in Figure 6, blocks 614 and 622 above.

[00101] Figure 16 is a block diagram of one embodiment of an I/O dispatch module 1600 to dispatch an input/output request. In one embodiment, the I/O dispatch module 1600 part of the Storage Controller 104 that performs the I/O dispatch process as described in Figure 8 above. In one embodiment, the I/O dispatch module 1600 includes a receive I/O request module 1602, validate module 1604, compute vNode module 1606, send I/O request module 1608, vNode update module 1610, vNode send I/O request module 1612, write log processing module 1614,

and return module 1616. In one embodiment, the receive I/O request module 1602 receives the I/O request from a client as described in Figure 8, block 802 above. The validate module 1604 validates the request as described in Figure 8, block 804 above. The compute vNode module 1606 compute the target vNode as described in Figure 8, block 806 above. The send I/O request module 1608 sends the I/O request as described in Figure 8, block 808 above. The vNode update module 1610 determines the vNodes that need to be updated as described in Figure 8, block 810 above. The vNode send I/O request module 1612 sends the I/O request to the vNodes as described in Figure 8, block 812 above. The write log processing module 1614 processes the writing to the write logs as described in Figure 8, blocks 816-822 above. The return module 1616 returns the results or an error code as described in Figure 8, blocks 824 and 826 above.

[00102] Figure 17 is a block diagram of one embodiment of a mirroring module 1700 to perform synchronous mirroring. In one embodiment, the distributed object layer include the mirroring module 1700. In one embodiment, the mirroring module 1700 includes a receive write log entry module 1702, get AMS module 1704, prepare write log entry module 1706, send module 1708, and acknowledge module 1710. In one embodiment, the receive write log entry module 1702 receives the write log entry as described above in Figure 9, block 902. The get AMS module 1704 get the AMS for the write log entry as described above in Figure 9, block 904. The prepare write log entry module 1706 prepares the write log entry as described above in Figure 9, block 906. The send module 1708 send the write log entry to a mirror as described above in Figure 9, block 908. The acknowledge module 1710 receives an acknowledgement as described above in Figure 9, block 910.

[00103] Figure 18 is a block diagram of one embodiment of an asynchronous module 1800 to asynchronously mirror data and metadata. In one embodiment, the asynchronous module 1800 includes a receive data module 1802, determine AMS module 1804, asynchronous send module 1806, and receive acknowledgement module 1808. In one embodiment, the receive data module 1802 receives the input as described above in Figure 10, block 1002. The determine AMS module 1804 determines the AMS for the target node as described above in Figure 10, block

1004. The asynchronous send module 1806 asynchronous sends the data and metadata to the vNode as described above in Figure 10, block 1006. The receive acknowledgement module 1808 receives an acknowledgement as described above in Figure 10, block 1008.

[00104] Figure 19 is a block diagram of one embodiment of a failure module 1900 to handle a failure of a component. In one embodiment, the fault tolerance layer 204 includes the failure module 1900. In one embodiment, the failure module 1900 includes a monitor module 1902, detect failure module 1904, determine failed component module 1906, vNode module 1908, CRM update module 1910, and recovery module 1911. In one embodiment, the monitor module 1902 monitors the cluster as described above in Figure 11, block 1102. The detect failure module 1904 detects a failure as described above in Figure 11, block 1104. The determine failed component module 1906 determines the failed component as described above in Figure 11, block 1106. The vNode module 1908 determines the impacted vNodes as described above in Figure 11, block 1110. The CRM update module 1910 performs CRM updates as described above in Figure 11, block 1114. The recovery module 1912 starts the StorFS recovery as described above in Figure 11, block 1118.

[00105] Figure 20 is a block diagram of one embodiment of a recovery module 2000 to perform a distributed failure recovery. In one embodiment, the recovery module 2000 includes a marked module 2002, active mirror check module 2004, alive mirror module 2006, create mirror module 2008, resync module 2010, and mirror send module 2012. In one embodiment, the marked module 2002 determines if the vNode is marked for recovery as described above in Figure 12, block 1204. The active mirror check module 2004 checks if the number of active mirrors is greater than a threshold as described above in Figure 12, block 1206. The alive mirror module 2006 determines if the node hosting the passive mirror is alive as described above in Figure 12, block 1208. The create mirror module 2008 creates a new mirror as described above in Figure 12, block 1210. The resync module 2010 starts the resync process as described above in Figure 12, block 1212. The mirror send module 2012 sends requests to the active mirrors as described above in Figure 12, block 1214.

[00106] Figure 21 shows one example of a data processing system 2100, which may be used with one embodiment of the present invention. For example, the system 2100 may be implemented including a physical server 102A-C as shown in Figure 1. Note that while Figure 21 illustrates various components of a computer system, it is not intended to represent any particular architecture or manner of interconnecting the components as such details are not germane to the present invention. It will also be appreciated that network computers and other data processing systems or other consumer electronic devices, which have fewer components or perhaps more components, may also be used with the present invention.

[00107] As shown in Figure 21, the computer system 2100, which is a form of a data processing system, includes a bus 2103 which is coupled to a microprocessor(s) 2105 and a ROM (Read Only Memory) 2107 and volatile RAM 2109 and a non-volatile memory 2111. The microprocessor 2105 may retrieve the instructions from the memories 2107, 2109, 2111 and execute the instructions to perform operations described above. The bus 2103 interconnects these various components together and also interconnects these components 2105, 2107, 2109, and 2111 to a display controller and display device 2113 and to peripheral devices such as input/output (I/O) devices which may be mice, keyboards, modems, network interfaces, printers and other devices which are well known in the art. Typically, the input/output devices 2115 are coupled to the system through input/output controllers 2117. The volatile RAM (Random Access Memory) 2109 is typically implemented as dynamic RAM (DRAM), which requires power continually in order to refresh or maintain the data in the memory.

[00108] The mass storage 2111 is typically a magnetic hard drive or a magnetic optical drive or an optical drive or a DVD RAM or a flash memory or other types of memory systems, which maintain data (e.g. large amounts of data) even after power is removed from the system. Typically, the mass storage 2111 will also be a random access memory although this is not required. While Figure 21 shows that the mass storage 2111 is a local device coupled directly to the rest of the components in the data processing system, it will be appreciated that the present invention may utilize a non-volatile memory which is remote from the system, such as a network

storage device which is coupled to the data processing system through a network interface such as a modem, an Ethernet interface or a wireless network. The bus 2103 may include one or more buses connected to each other through various bridges, controllers and/or adapters as is well known in the art.

[00109] Portions of what was described above may be implemented with logic circuitry such as a dedicated logic circuit or with a microcontroller or other form of processing core that executes program code instructions. Thus processes taught by the discussion above may be performed with program code such as machine-executable instructions that cause a machine that executes these instructions to perform certain functions. In this context, a “machine” may be a machine that converts intermediate form (or “abstract”) instructions into processor specific instructions (e.g., an abstract execution environment such as a “process virtual machine” (e.g., a Java Virtual Machine), an interpreter, a Common Language Runtime, a high-level language virtual machine, etc.), and/or, electronic circuitry disposed on a semiconductor chip (e.g., “logic circuitry” implemented with transistors) designed to execute instructions such as a general-purpose processor and/or a special-purpose processor. Processes taught by the discussion above may also be performed by (in the alternative to a machine or in combination with a machine) electronic circuitry designed to perform the processes (or a portion thereof) without the execution of program code.

[00110] The present invention also relates to an apparatus for performing the operations described herein. This apparatus may be specially constructed for the required purpose, or it may comprise a general-purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), RAMs, EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus.

[00111] A machine readable medium includes any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer). For example, a machine readable medium includes read only memory (“ROM”); random access memory (“RAM”); magnetic disk storage media; optical storage media; flash memory devices; etc.

[00112] An article of manufacture may be used to store program code. An article of manufacture that stores program code may be embodied as, but is not limited to, one or more memories (e.g., one or more flash memories, random access memories (static, dynamic or other)), optical disks, CD-ROMs, DVD ROMs, EPROMs, EEPROMs, magnetic or optical cards or other type of machine-readable media suitable for storing electronic instructions. Program code may also be downloaded from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals embodied in a propagation medium (e.g., via a communication link (e.g., a network connection)).

[00113] The preceding detailed descriptions are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the tools used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[00114] It should be kept in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the above discussion, it is appreciated that throughout the description, discussions utilizing terms such as “receiving,” “determining,” “transmitting,” “computing,” “sending,” “forwarding,” “dispatching,”

“detecting,” “performing,” “scheduling,” “communicating,” “reading,” “writing,” “transferring,” “updating,” “returning,” “merging,” “appending,” “fetching,” or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[00115] The processes and displays presented herein are not inherently related to any particular computer or other apparatus. Various general-purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct a more specialized apparatus to perform the operations described. The required structure for a variety of these systems will be evident from the description below. In addition, the present invention is not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings of the invention as described herein.

[00116] The foregoing discussion merely describes some exemplary embodiments of the present invention. One skilled in the art will readily recognize from such discussion, the accompanying drawings and the claims that various modifications can be made without departing from the spirit and scope of the invention.

CLAIMS

What is claimed is:

1. A non-transitory machine-readable medium having executable instructions to cause one or more processing units to perform a method to dispatch an input/output request in a distributed storage system, the method comprising:

receiving, at a storage controller client, the input/output request, wherein the distributed storage system includes the storage controller client, a plurality of storage controller servers, a plurality of virtual nodes distributed among a plurality of physical nodes, and each of the plurality of physical nodes is hosted on one of the plurality of storage controller servers;

computing a target virtual node for the input/output request, wherein the target virtual node is one of the plurality of virtual nodes;

determining a target physical node that corresponds to the target virtual node, wherein the target physical node is one of the plurality of physical nodes; and

dispatching the input/output request to a target storage controller server that corresponds to the target physical node, wherein the target storage controller server is one of the plurality of storage controller servers.

2. The non-transitory machine-readable medium of claim 1, wherein the input/output request is selected from the group consisting of a write operation request, a read operation request, and a query operation request.

3. The non-transitory machine-readable medium of claim 1, wherein the computing of the target virtual node is computed using a deterministic placement function.

4. The non-transitory machine-readable medium of claim 1, wherein the storage controller client and the target storage controller are on different devices that are coupled together by a network.

5. The non-transitory machine-readable medium of claim 1, wherein the determining a target physical node comprises:
- performing a lookup with the target virtual node for the target physical node using a storage routing table.
6. The non-transitory machine-readable medium of claim 1, wherein the dispatching comprises:
- scheduling a dispatch of the input/output request to the target using a policy based on a characteristic of the input/output request; and
 - transmitting the input/output request to the target storage controller server based on the scheduling.
7. A non-transitory machine-readable medium having executable instructions to cause one or more processing units to perform a method to write data in a distributed storage system, the method comprising:
- receiving, at one of a plurality of storage controller servers, a write operation request to store data at a virtual node, wherein the write operation request is dispatched from a storage controller client to the one of the plurality of storage controller servers using a deterministic placement function, and the distributed storage system includes the storage controller client, and the plurality of storage controller servers; and
 - in response to receiving the write operation request, writing the data to a mirror set of the virtual node.
8. The non-transitory machine-readable medium of claim 7, wherein a mirror set includes one or more mirrors that is used to replicate data and metadata of the virtual node
9. The non-transitory machine-readable medium of claim 8, wherein the writing the data comprises:
- determining a primary mirror of the mirror set; and
 - storing the data from a segment store of the primary mirror.

10. The non-transitory machine-readable medium of claim 9, wherein if the writing of the data is not successful,
marking the primary mirror as inactive; and
electing a new primary mirror.
11. The non-transitory machine-readable medium of claim 9, wherein the segment store is thin provisioned logical partition of distributed storage system.
12. The non-transitory machine-readable medium of claim 9, wherein the writing of the data further comprises:
storing the data in an active mirror of the mirror set.
13. The non-transitory machine-readable medium of claim 12, wherein if the storing of the data in the active mirror is not successful,
marking the active mirror as inactive.
14. The non-transitory machine-readable medium of claim 7, wherein the data includes an object to be stored and a key to reference the object.
15. A non-transitory machine-readable medium having executable instructions to cause one or more processing units to perform a method to process a read operation request in a distributed storage system, the method comprising:
receiving, at one of a plurality of storage controller servers, the read operation request for a virtual node, wherein the write operation request is dispatched from a storage controller client to the storage controller server using a deterministic placement function, and the distributed storage system includes the storage controller client and the plurality of storage controller servers; and
in response to receiving the read operation request,
determining a mirror set for the read operation request, and
reading data indicated in the read operation request from the mirror set.

16. The non-transitory machine-readable medium of claim 15, wherein a mirror set includes one or more mirrors that is used to replicate data and metadata of the virtual node.

17. The non-transitory machine-readable medium of claim 16, wherein the reading the data comprises:

- determining a primary mirror of the mirror set; and
- fetching the data from a segment store of the primary mirror.

18. The non-transitory machine-readable medium of claim 17, wherein if the fetching of the data is not successful, fetching the data from an active mirror of the mirror set.

19. The non-transitory machine-readable medium of claim 17, wherein if the fetching of the data is not successful, repairing the primary mirror.

20. The non-transitory machine-readable medium of claim 17, wherein if the fetching of the data is not successful,
marking the primary mirror as inactive; and
electing a new primary mirror from an active mirror of the mirror set.

21. A non-transitory machine-readable medium having executable instructions to cause one or more processing units to perform a method to synchronously replicate data to a plurality of storage controller servers, the method comprising:
receiving a write log entry on one of the plurality of storage controller servers;
retrieving a target virtual node for the write log entry;
determining a plurality of virtual mirror nodes for the target virtual node, wherein each of the mirror nodes is located on different ones of the plurality of controller servers; and
synchronously sending the write log entry to each of the different ones of the plurality storage controller servers.

22. The non-transitory machine-readable medium of claim 21, further comprising:
writing the write log entry to the target virtual node.

23. The non-transitory machine-readable medium of claim 21, wherein the write log entry include virtual node identifier, a timestamp, a checksum, and data to be stored.
24. The non-transitory machine-readable medium of claim 21, wherein a virtual node is a selected from the group consisting of a collection of data objects and a collection of metadata objects.
25. The non-transitory machine-readable medium of claim 24, wherein a data object is selected from the group consisting of a file, an object, and a key-value pair.
26. The non-transitory machine-readable medium of claim 21, wherein the determining the plurality of virtual mirror nodes for the virtual node comprises:
retrieving the plurality of virtual mirror nodes from an active mirror set for the virtual node.
27. The non-transitory machine-readable medium of claim 26, wherein the active mirror set is a set of one or more active virtual nodes that are used to mirror data stored in the virtual node.
28. The non-transitory machine-readable medium of claim 26, wherein an active mirror virtual node is a mirror node that stores the same data as stored on the target virtual node.
29. The non-transitory machine-readable medium of claim 21, further comprising:
receiving an acknowledgement for a writing of the write log entry from the each of the different ones of the plurality storage controller servers.

30. A non-transitory machine-readable medium having executable instructions to cause one or more processing units to perform a method to asynchronously replicate data for persistent storage to a plurality of storage controller servers, the method comprising:

receiving data, metadata, and target virtual node on one of the plurality of storage controller servers;

determining a plurality of virtual mirror nodes for the target virtual node, wherein each of the mirror virtual nodes is located on different ones of the plurality of controller servers; and

asynchronously sending the data and metadata to each of the different ones of the plurality storage controller servers, wherein the data and metadata are stored in a persistent storage of a respective plurality of storage controller servers.

31. The non-transitory machine-readable medium of claim 30, wherein a virtual node is a selected from the group consisting of a collection of data objects and a collection of metadata objects.

32. The non-transitory machine-readable medium of claim 31, wherein a data object is selected from the group consisting of a file, an object, and a key-value pair.

33. The non-transitory machine-readable medium of claim 30, wherein the determining the plurality of virtual mirror nodes for the virtual node comprises:

retrieving the plurality of virtual mirror nodes from an active mirror set for the virtual node.

34. The non-transitory machine-readable medium of claim 30, wherein the active mirror set is a set of one or more active virtual nodes that is used to mirror data stored in the virtual node.

35. The non-transitory machine-readable medium of claim 34, wherein an active mirror virtual node is a mirror node that stores the same data as stored on the target virtual node.

36. A non-transitory machine-readable medium having executable instructions to cause one or more processing units to perform a method to handle a failure of a component of the distributed storage system, the method comprising:

detecting, by one of a plurality of storage controller servers, a component failure in the distributed storage system;
determining a virtual node that is impacted by the component failure;
retrieving an active mirror set for the impacted virtual node;
if a number of active mirror virtual nodes identified in the active mirror set is less than a threshold,
allocating a new virtual node, and
recovering stored data from the active mirror set for the new virtual node.

37. The non-transitory machine-readable medium of claim 36, wherein the active mirror set is a plurality of active virtual nodes that are used to mirror data stored in the virtual node.

38. The non-transitory machine-readable medium of claim 37, wherein each of the plurality of active virtual nodes is in a different fault domain, wherein a failure of a component for one of the plurality of active virtual node does not affect others of the plurality of active virtual nodes.

39. The non-transitory machine-readable medium of claim 38, wherein the new virtual node is allocated in a fault domain different from one of the fault domains used by the plurality of active virtual nodes.

40. The non-transitory machine-readable medium of claim 36, wherein an active virtual node is a virtual node that includes an up to date stored data for the active mirror set.

41. The non-transitory machine-readable medium of claim 36, wherein the recovering data comprises:

copying the stored data from an active virtual node of the active mirror set to the new virtual node, wherein after the copying, the active virtual node and the new virtual node have the same stored data.

42. The non-transitory machine-readable medium of claim 36, wherein the recovering data comprises:

copying the stored data from a plurality of active virtual nodes of the active mirror set to the new virtual node, wherein after the copying, the plurality of active virtual nodes and the new virtual node each have the same stored data.

43. The non-transitory machine-readable medium of claim 42, wherein data stored in the active mirror set is stored with a generation identifier and the recovering data uses the generation identifier.

ABSTRACT OF THE DISCLOSURE

A distributed storage system that dispatches an input/output request is described. In an exemplary embodiment, a storage controller client receives the input/output request, wherein the distributed storage system includes the storage controller client, a plurality of storage controller servers, a plurality of virtual nodes distributed among a plurality of physical nodes, and each of the plurality of physical nodes is hosted on one of the plurality of storage controller servers. The storage controller client further computes a target virtual node for the input/output request, where the target virtual node is one of the plurality of virtual nodes. Using the computed target virtual node, the storage controller client determines a target physical node that corresponds to the target virtual node, where the target physical node is one of the plurality of physical nodes. In addition, the storage controller client dispatches the input/output request to a target storage controller that corresponds to the target physical node, wherein the target storage controller server is one of the plurality of storage controller servers. In addition, the virtual node includes a set of one or more mirrored copies across different fault domains for ensuring resiliency and high availability.

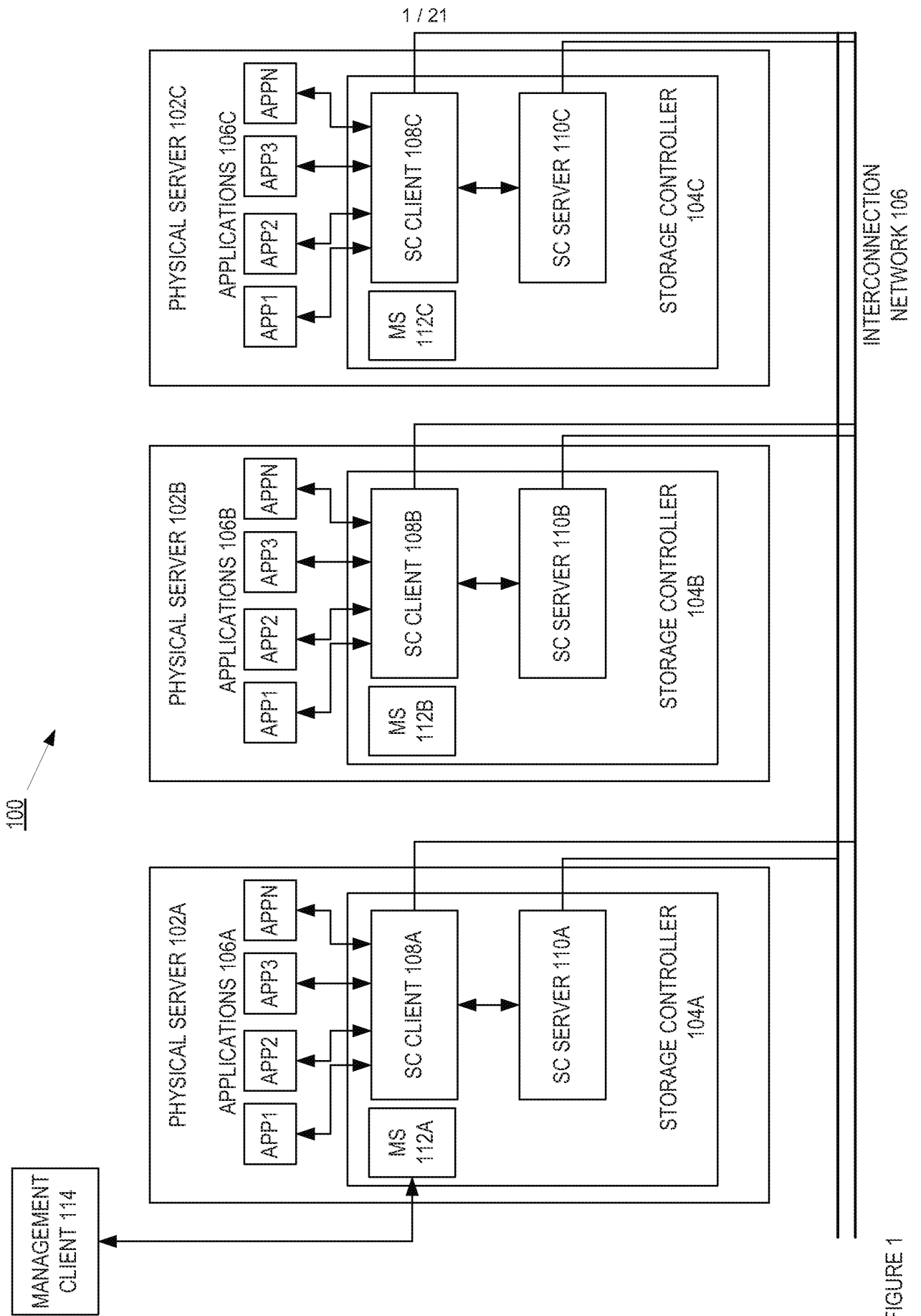


FIGURE 1

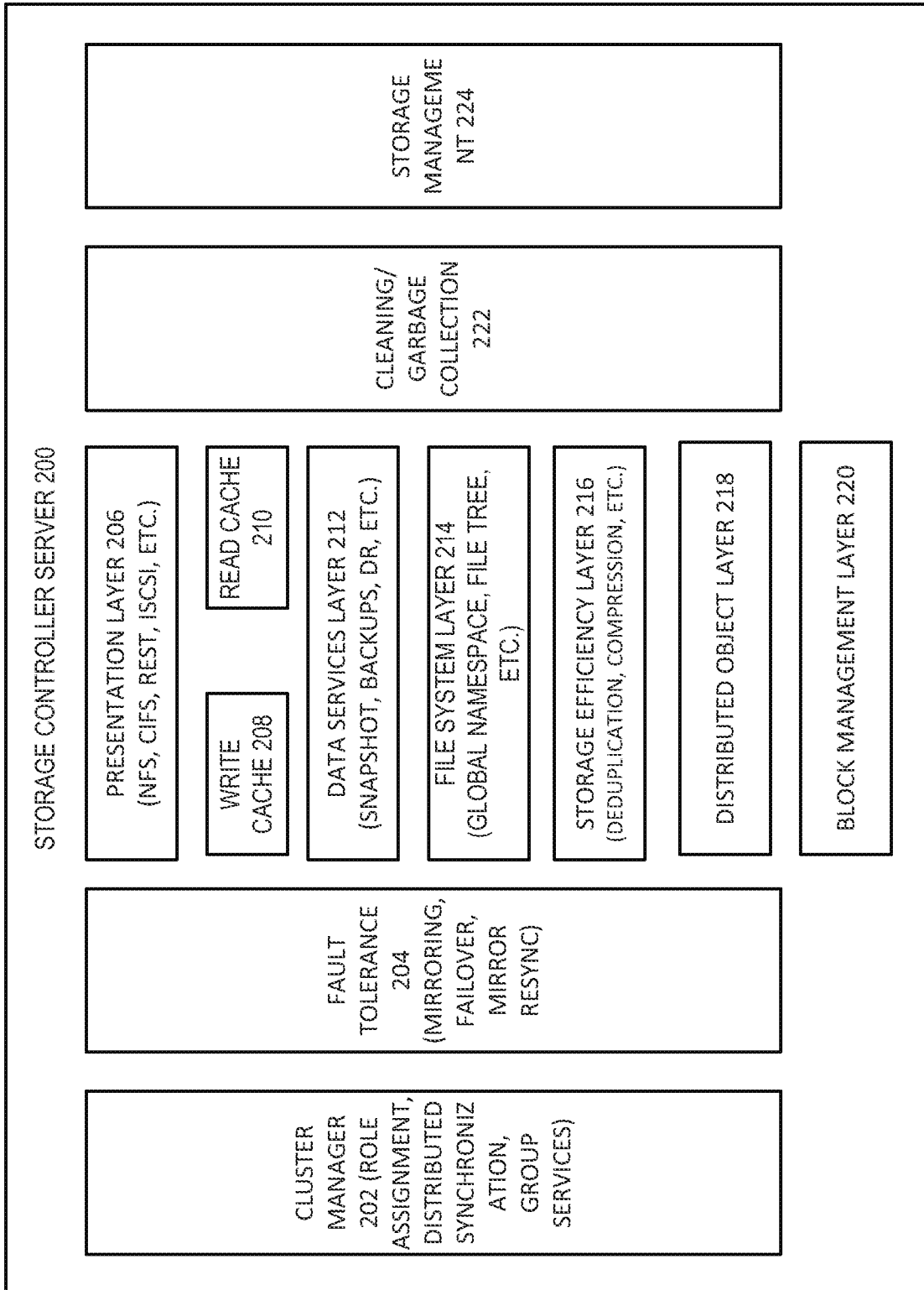


FIGURE 2

300

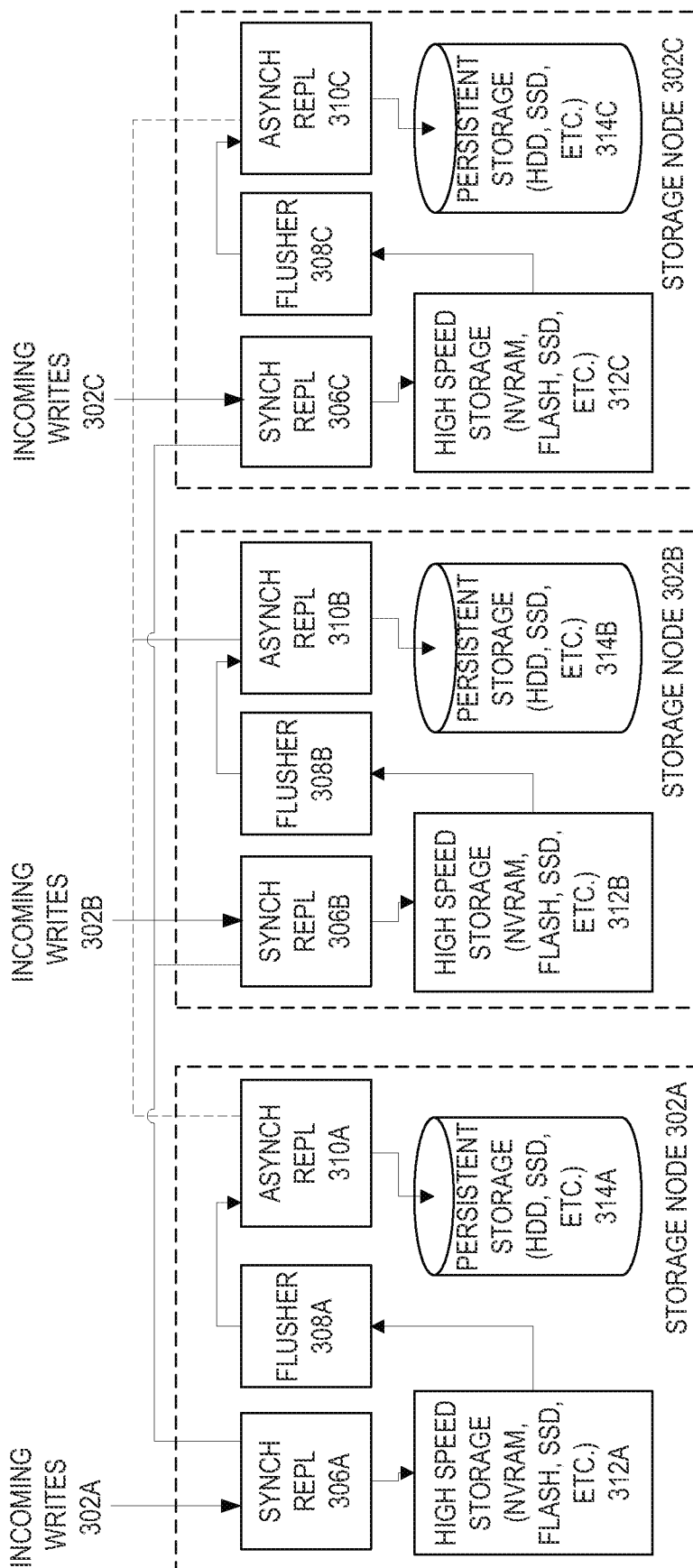


FIGURE 3

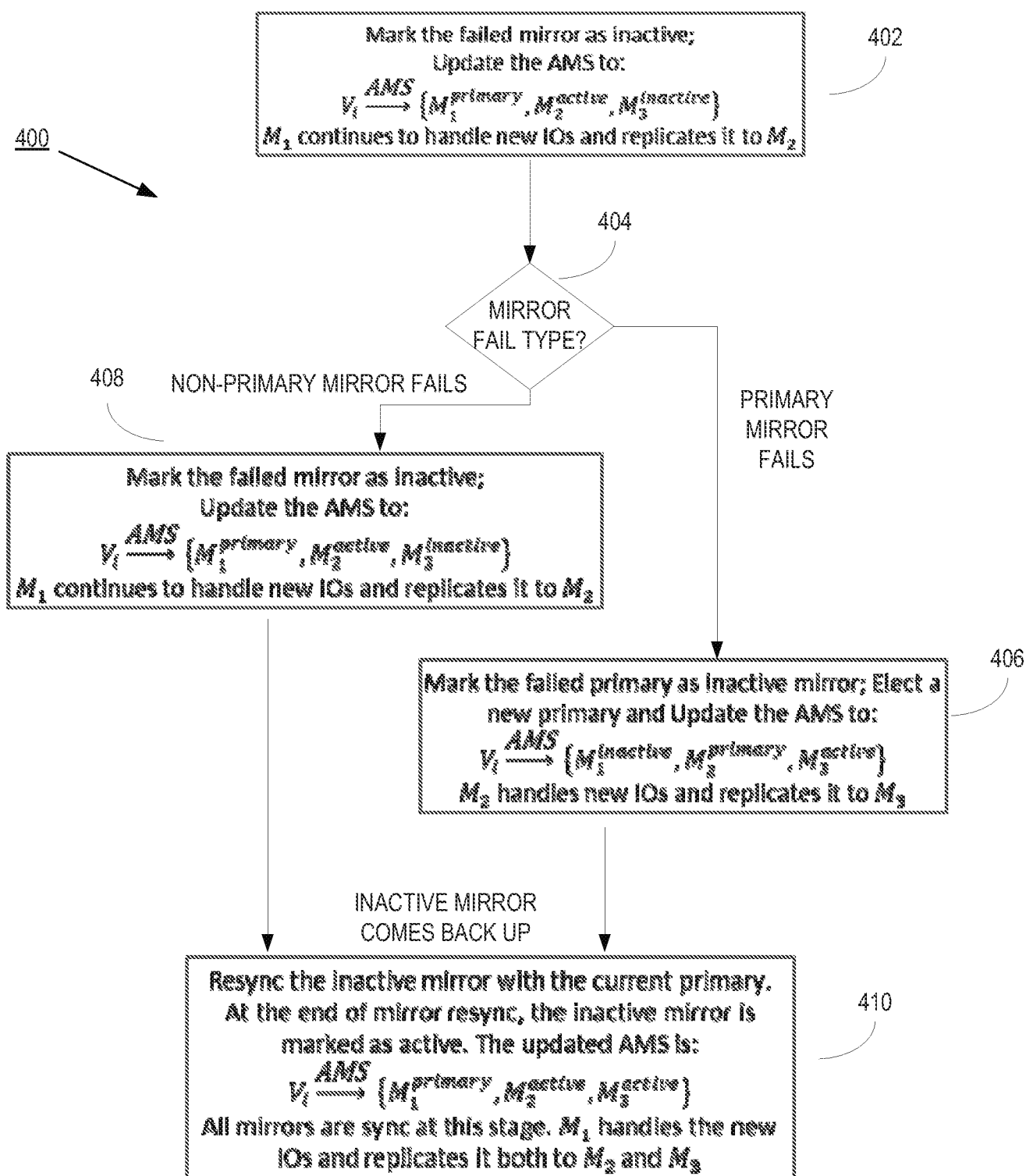
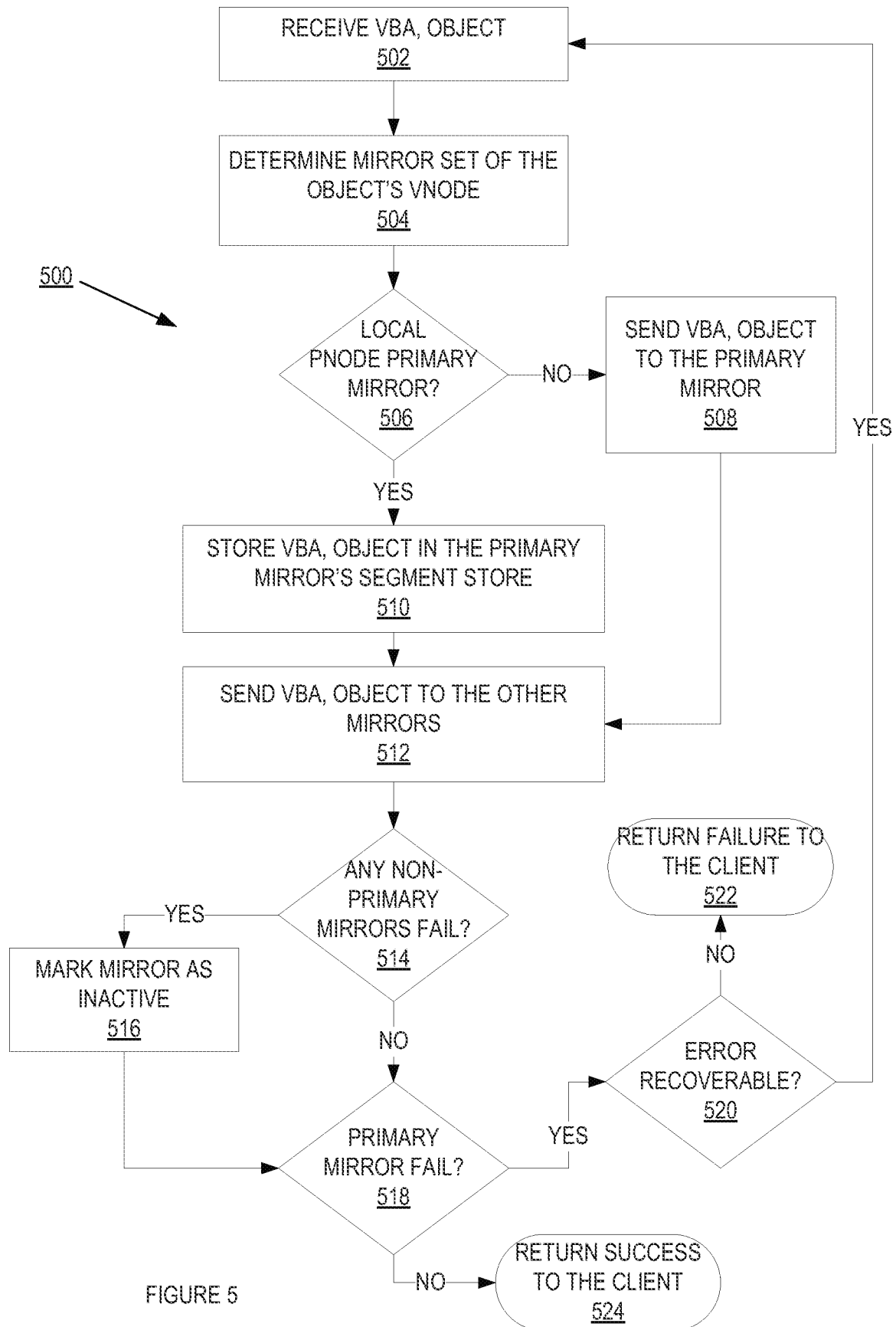
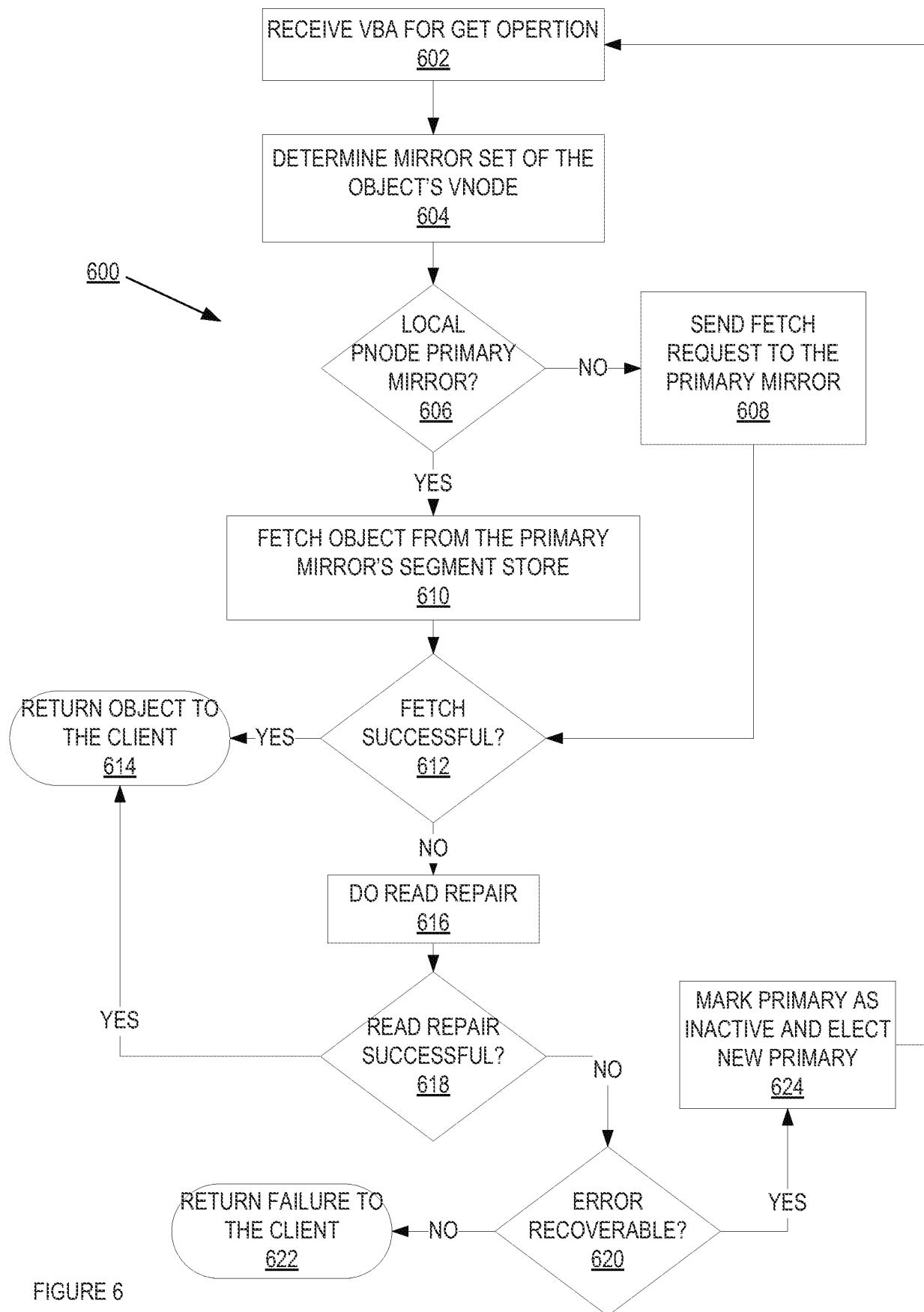


FIGURE 4





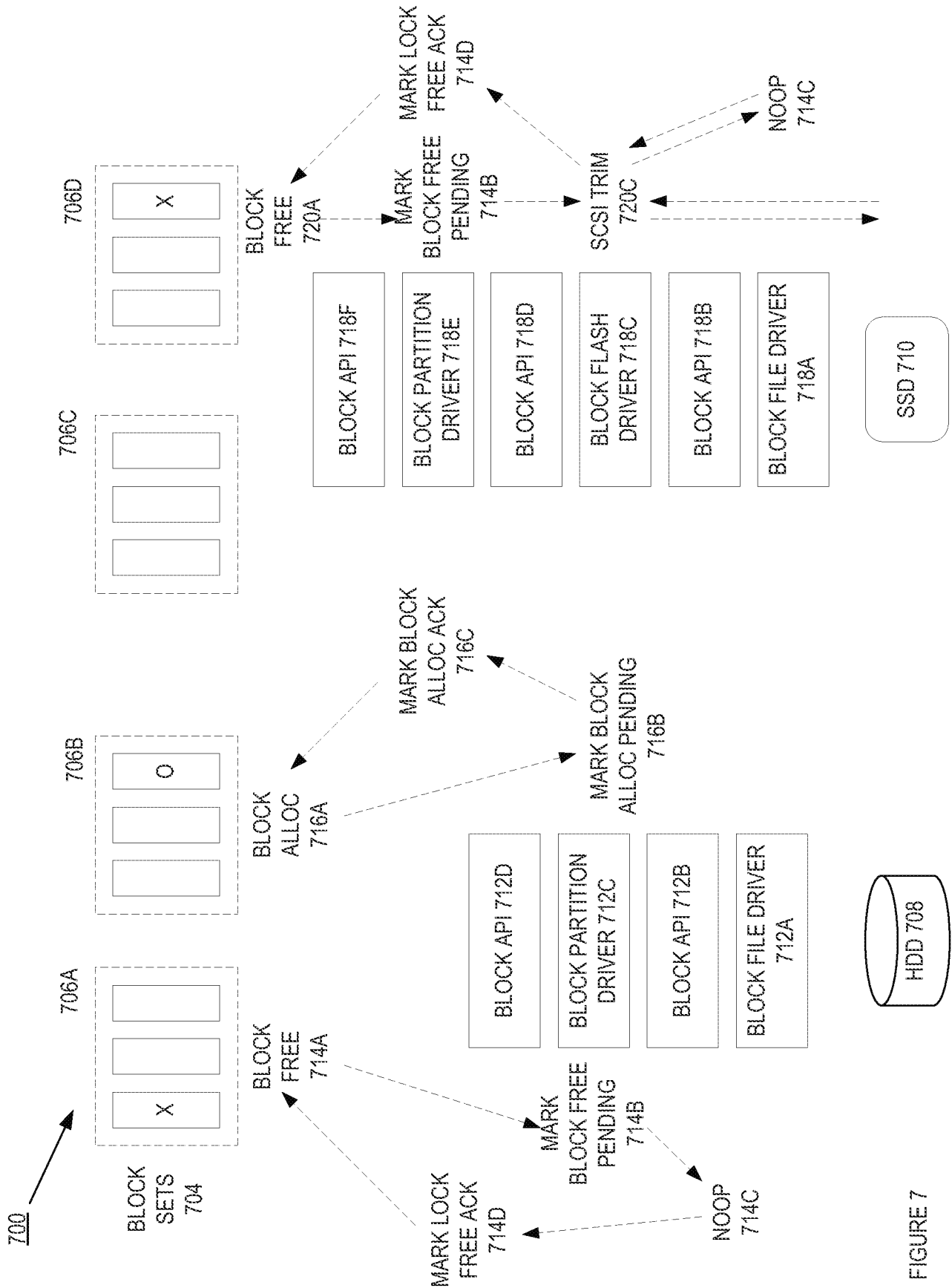


FIGURE 7

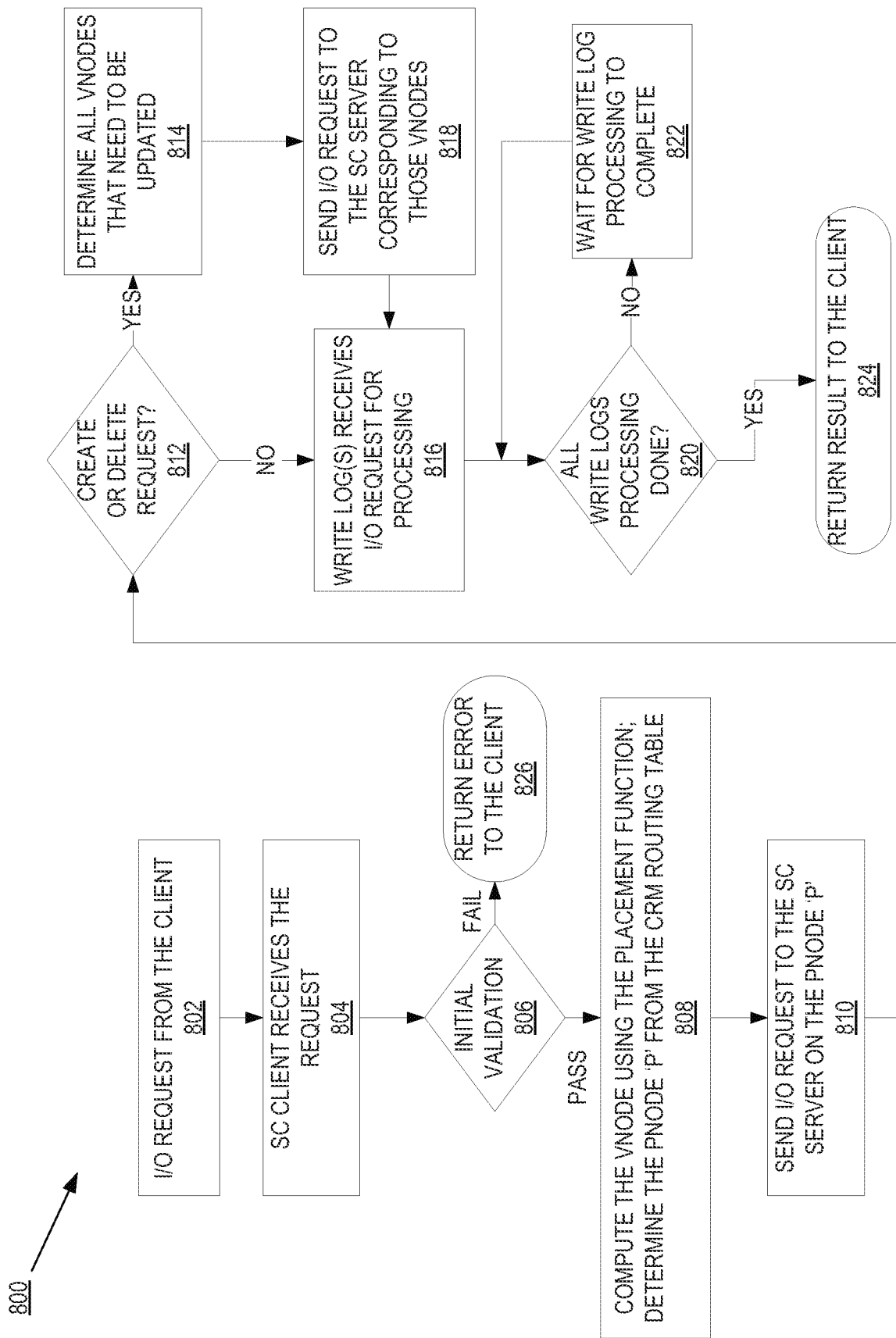


FIGURE 8

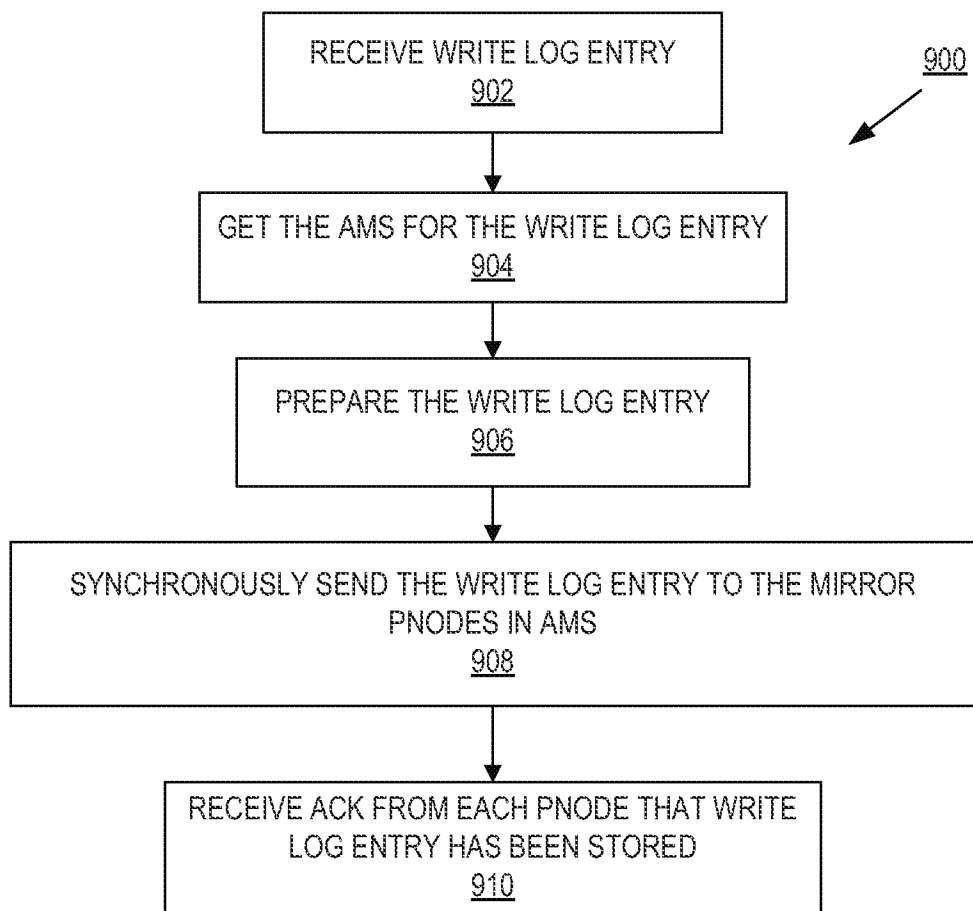


FIGURE 9

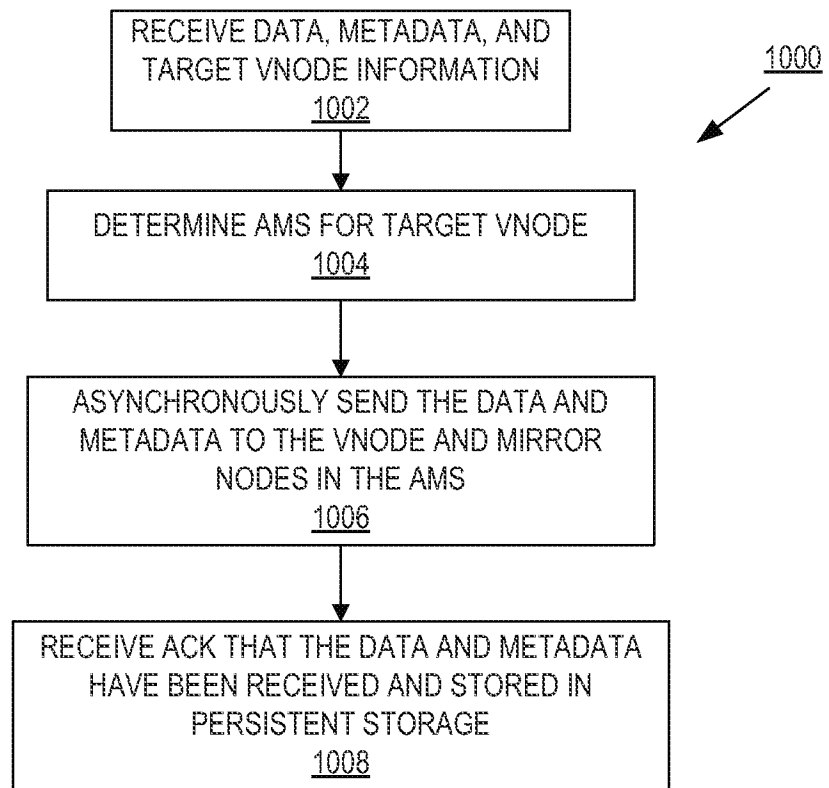


FIGURE 10

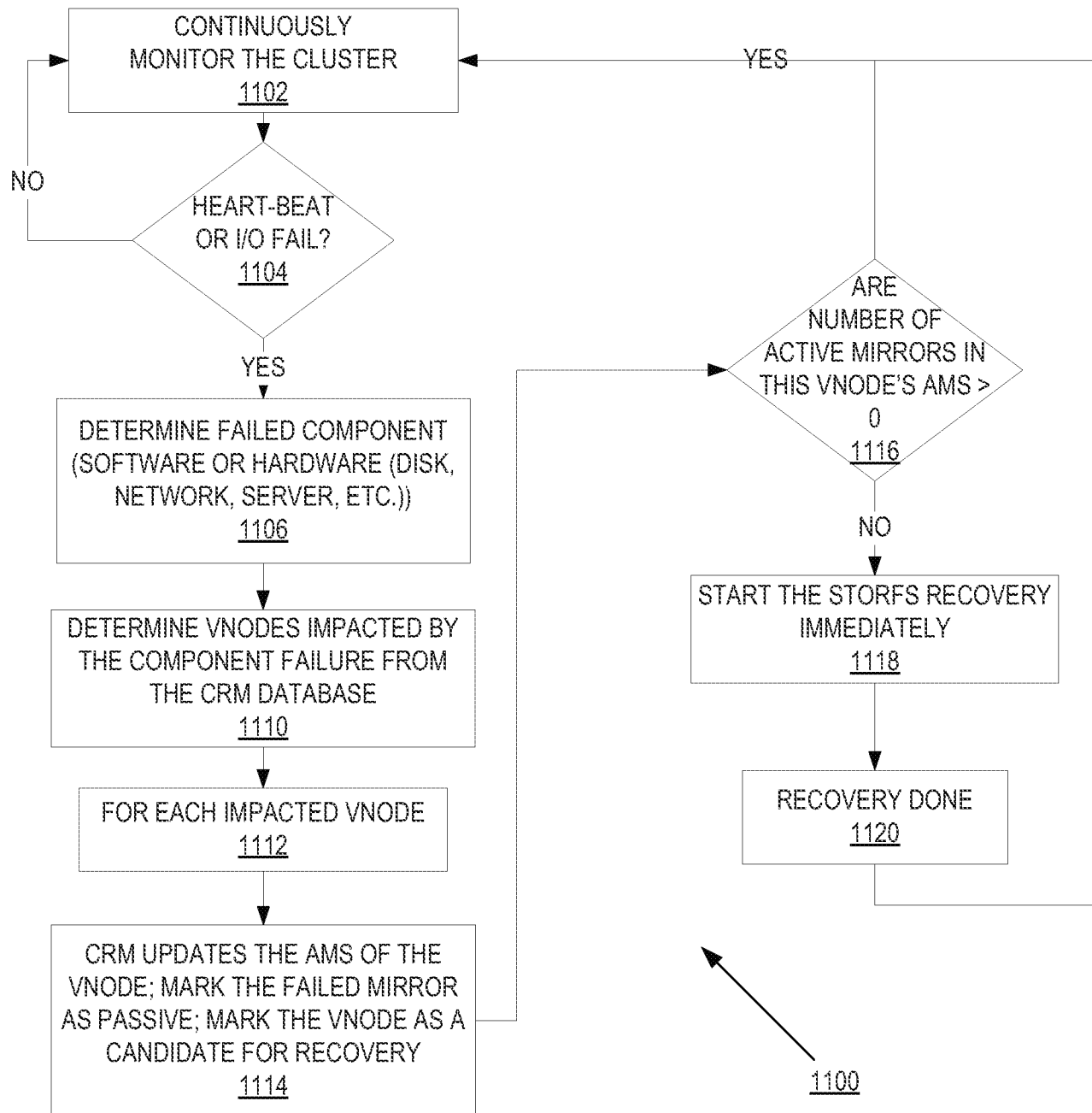


FIGURE 11

FIGURE 12

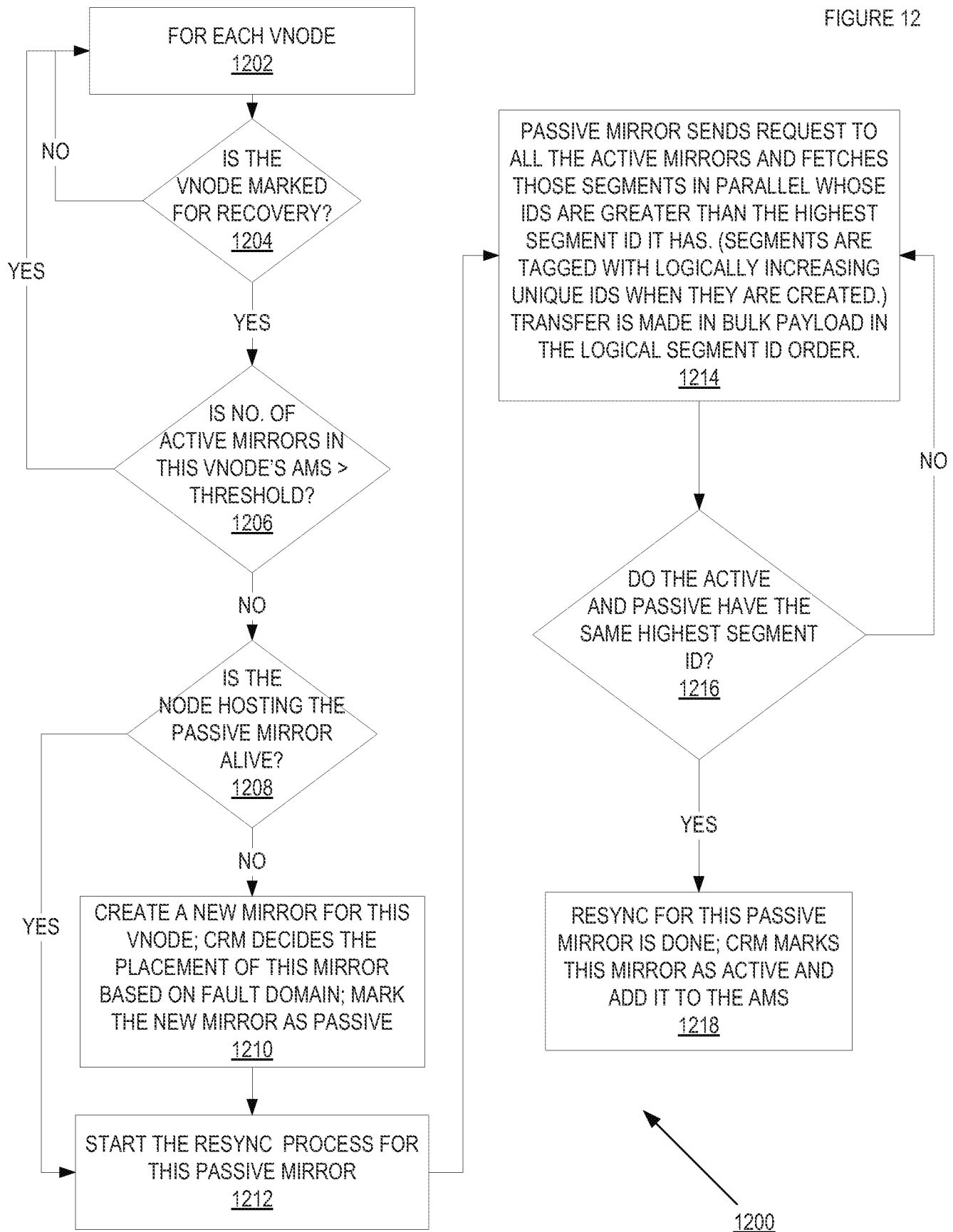


FIGURE 13

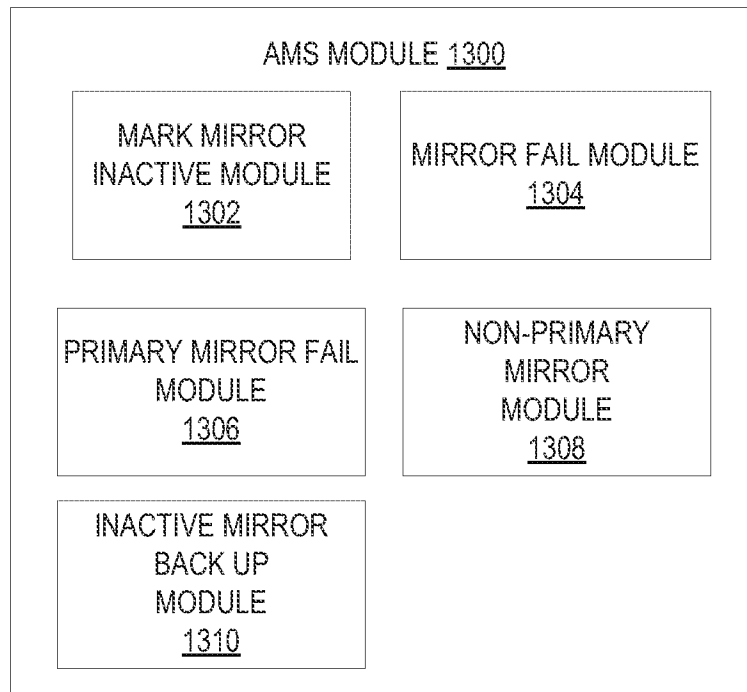


FIGURE 14

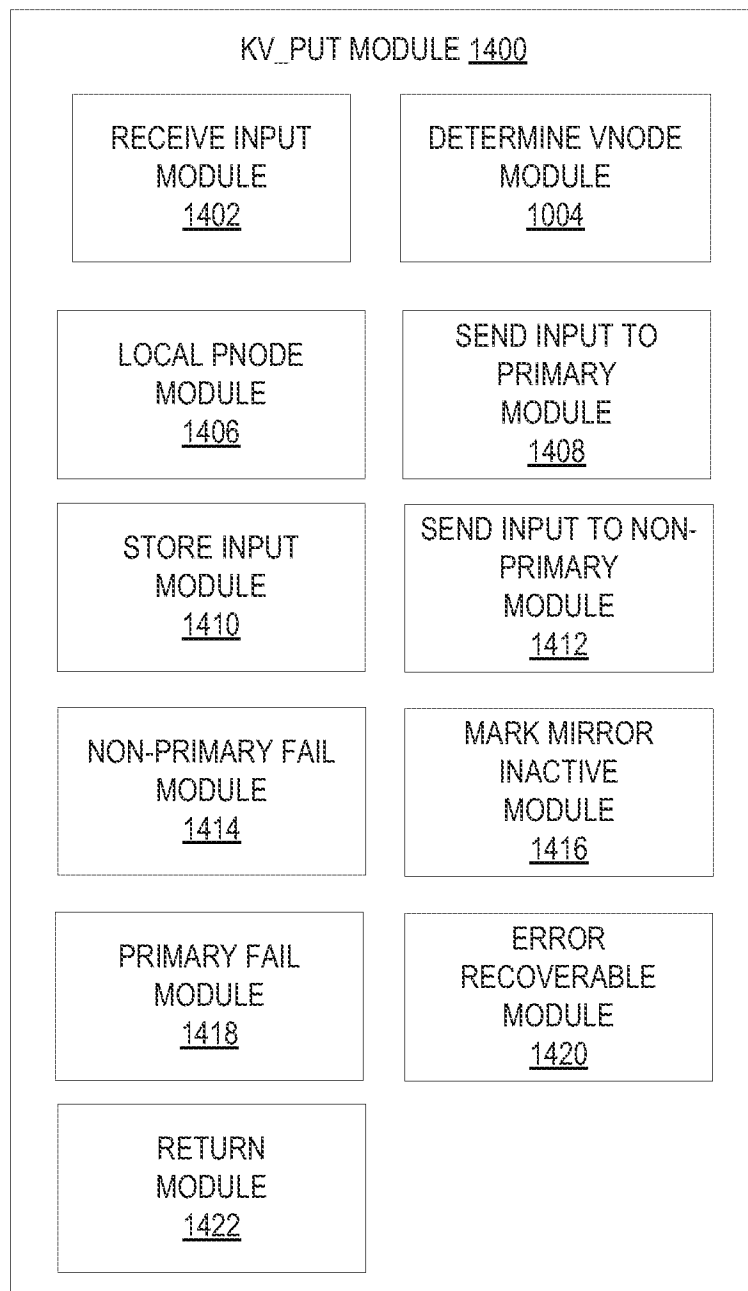


FIGURE 15

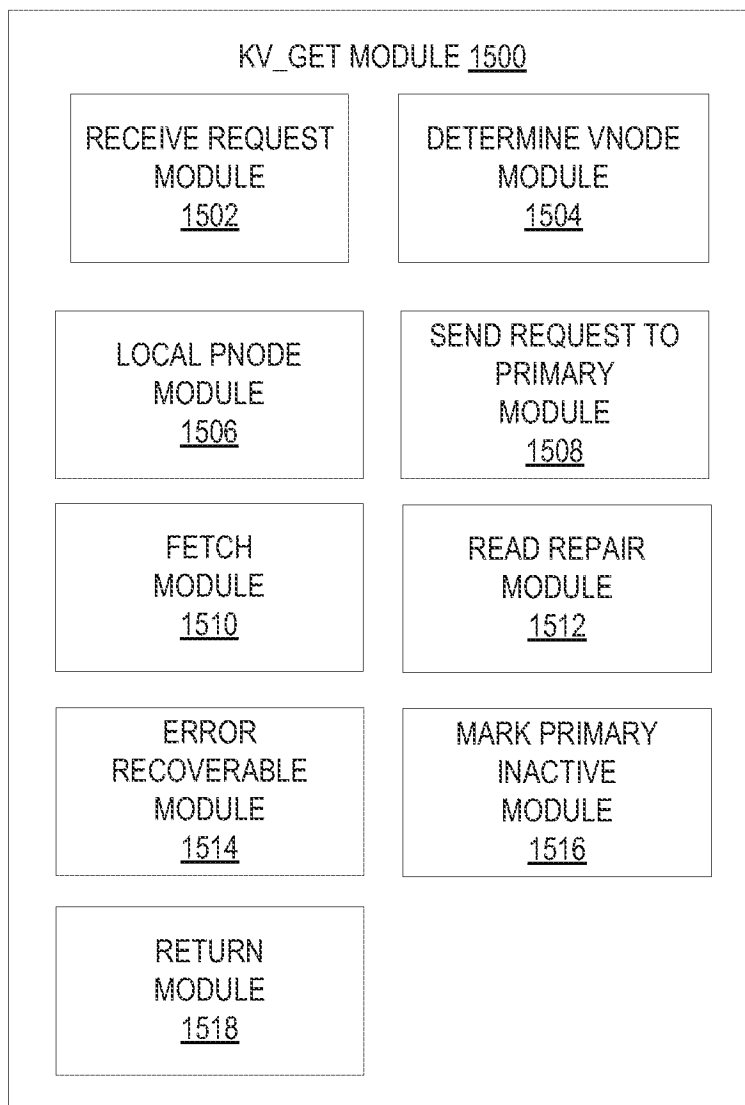


FIGURE 16

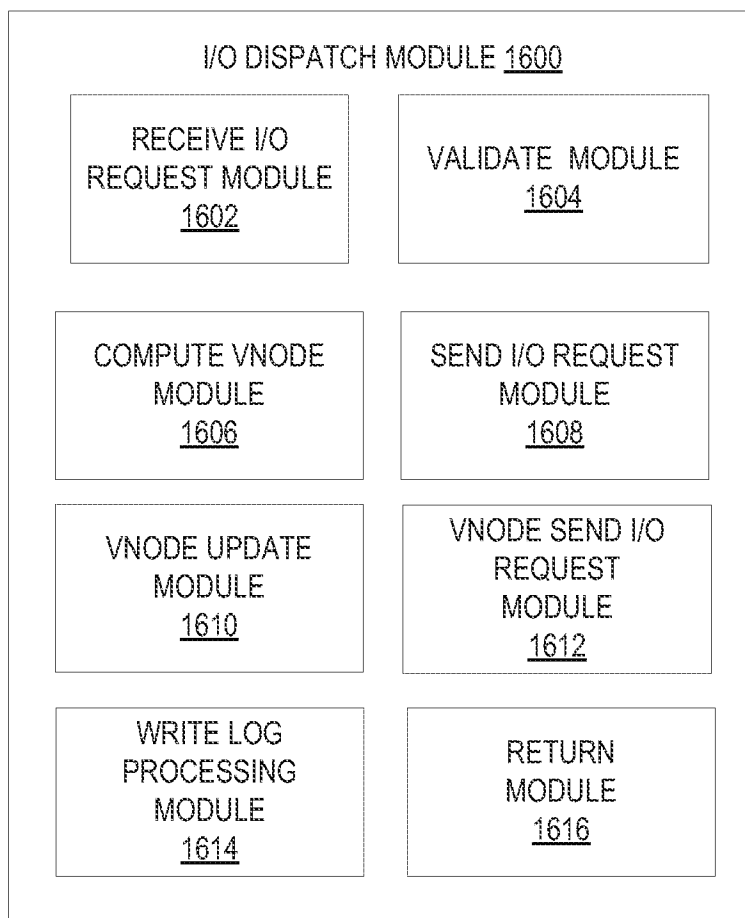


FIGURE 17

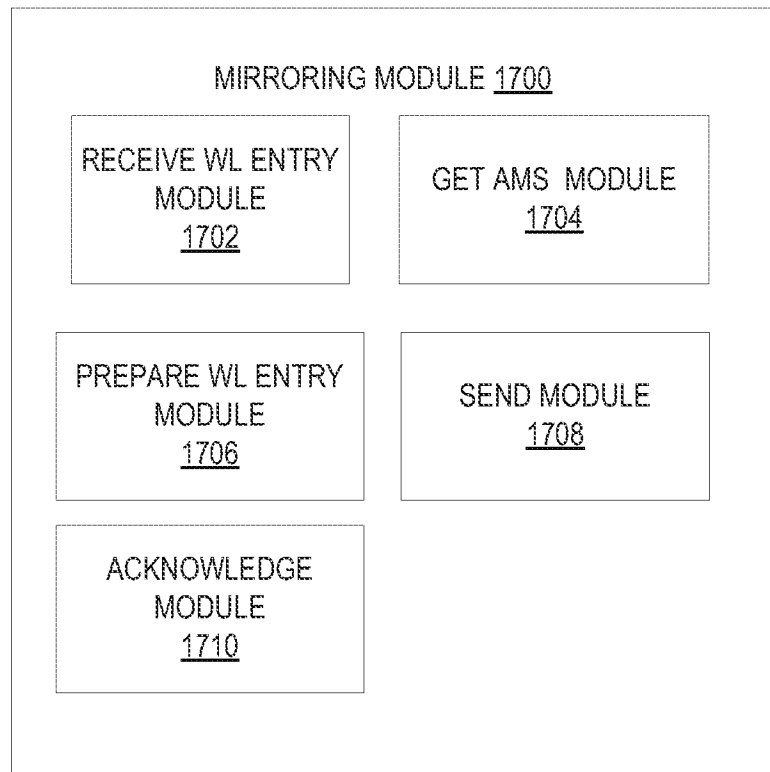


FIGURE 18

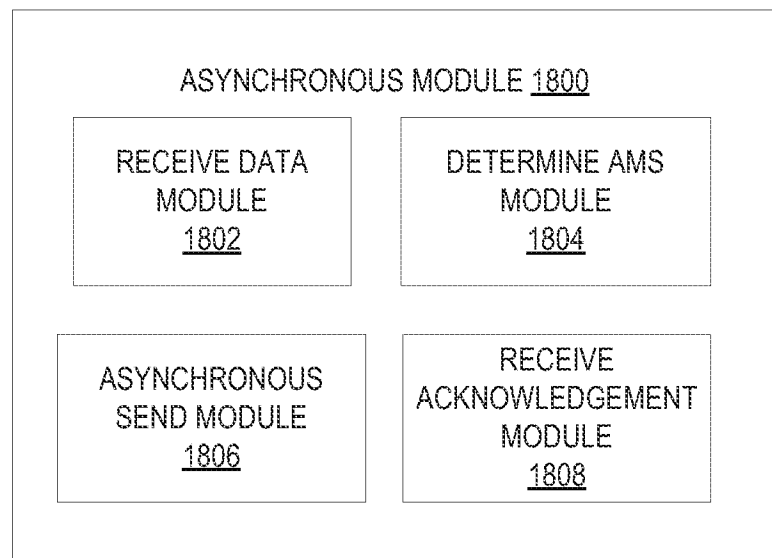


FIGURE 19

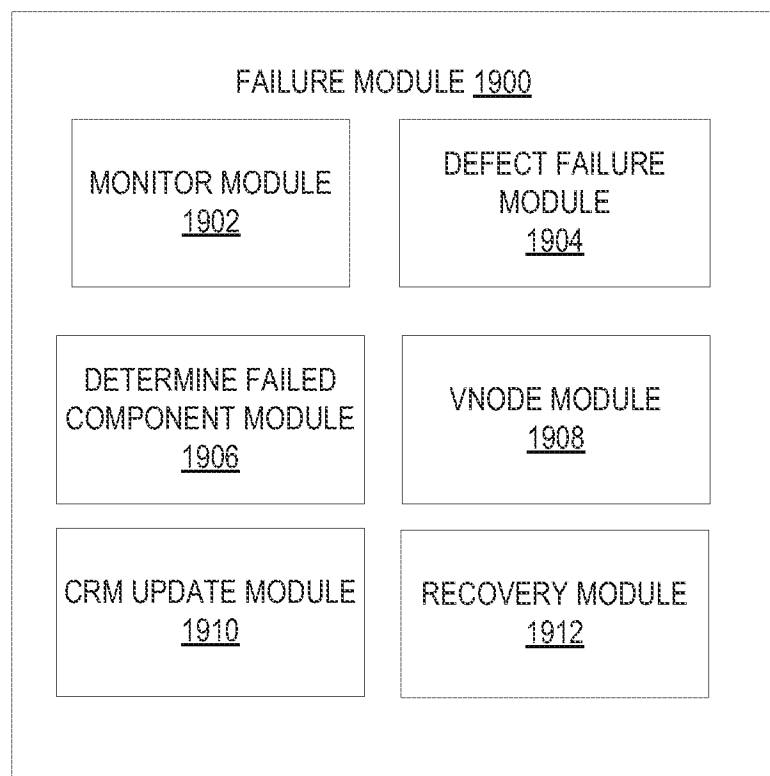


FIGURE 20



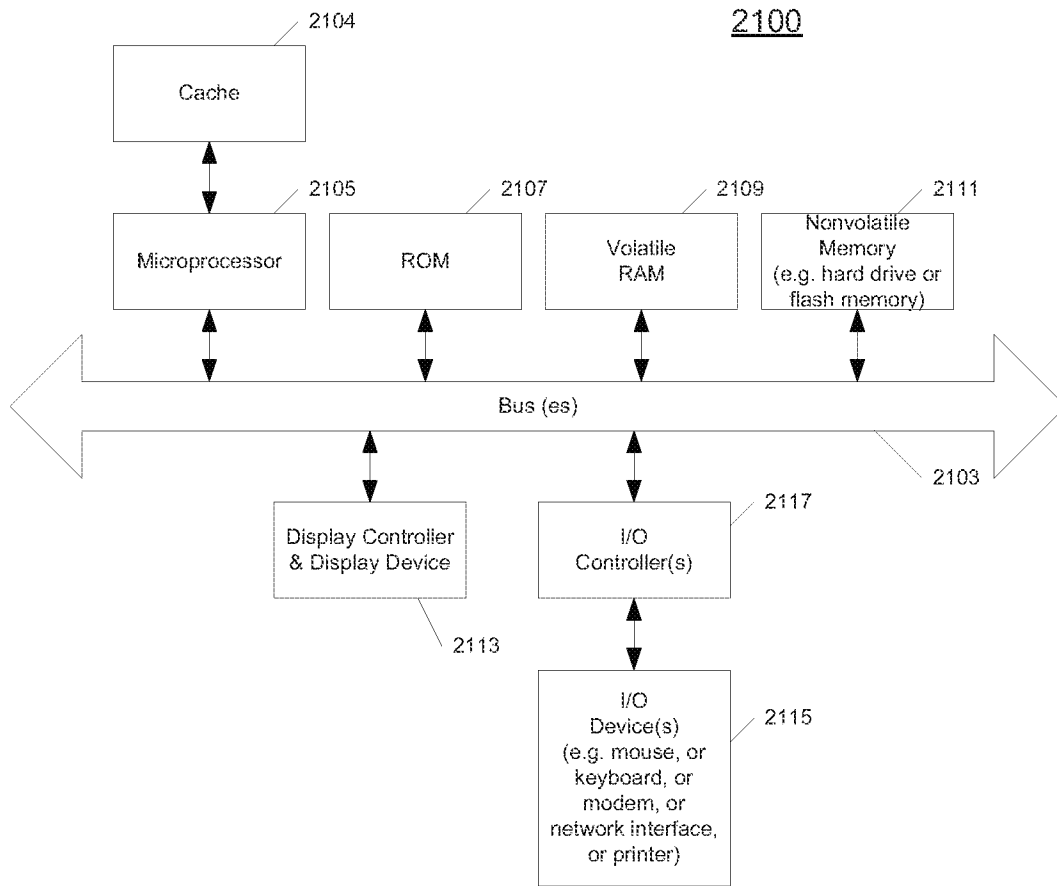


FIGURE 21

**DECLARATION AND POWER OF
ATTORNEY FOR UTILITY OR DESIGN
PATENT APPLICATION
(37 CFR 1.63)**



Declaration
Submitted
With Initial
Filing

OR



Declaration
Submitted After Initial
Filing (surcharge
(37 CFR 1.16(f))
required)

Attorney Docket
Number

009462 P001

First Named Inventor

Sandip AGARWALA

COMPLETE IF KNOWN

Application Number

Herewith

Filing Date

Art Unit

Unassigned

Examiner Name

Unassigned

**SYSTEMS AND METHODS FOR IMPLEMENTING AN ENTERPRISE-CLASS CONVERGED
COMPUTE-NETWORK-STORAGE APPLIANCE**

(Title of the Invention)

As a below named inventor, I hereby declare that:

This declaration is directed to:



The attached application.

OR



United States Application Number or PCT International application number _____
filed on _____

The above-identified application was made or authorized to be made by me.

I believe I am the original inventor or an original joint inventor of a claimed invention in the application.

I have reviewed and understand the contents of the above-identified application, including the claim(s).

I am aware of the duty to disclose to the Patent and Trademark Office all information known to me to be material to patentability as defined in 37 C.F.R. 1.56, including for a continuation-in-part application, information material to patentability that became available between the filing date of the prior application and the national or PCT international filing date of the continuation-in-part application.

I hereby acknowledge that any willful false statement made in this declaration is punishable under 18 U.S.C. 1001 by fine or imprisonment of not more than five (5) years, or both.

Willful false statements and the like are punishable by fine or imprisonment, or both (18 U.S.C. 1001), and may jeopardize the validity of the application or any patent issuing thereon. I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true.

Authorization To Permit Access To Application by Participating Office



If checked, the undersigned hereby grants the USPTO authority to provide the European Patent Office (EPO), the Japan Patent Office (JPO), the Korean Intellectual Property Office (KIPO), the World Intellectual Property Office (WIPO), and any other intellectual property offices in which a foreign application claiming priority to the above-identified patent application is filed access to the above-identified patent application. See 37 CFR 1.14(c) and (h). This box should not be checked if the applicant does not wish the EPO, JPO, KIPO, WIPO, or other intellectual property office in which a foreign application claiming priority to the above-identified patent application is filed to have access to the above-identified patent application.

In accordance with 37 CFR 1.14(h)(3), access will be provided to a copy of the above-identified patent application with respect to: 1) the above-identified patent application-as-filed; 2) any foreign application to which the above-identified patent application claims priority under 35 U.S.C. 119(a)-(d) if a copy of the foreign application that satisfies the certified copy requirement of 37 CFR 1.55 has been filed in the above-identified patent application; and 3) any U.S. application-as-filed from which benefit is sought in the above-identified patent application.

In accordance with 37 CFR 1.14(c), access may be provided to information concerning the date of filing the Authorization to Permit Access to Application by Participating Offices.

DECLARATION AND POWER OF ATTORNEY— Utility or Design Patent Application

APPOINTMENT OF PATENT PRACTITIONERS:

☒ I hereby appoint Practitioner(s) associated with the following Customer Number as my/our attorney(s) or agent(s) to prosecute this application and to transact all business in the United States Patent and Trademark Office connected therewith: **08791**

If this patent application is assigned, then each of the undersigned hereby authorizes the patent attorneys and patent agents named herein to accept and follow instructions from the assignee(s) as to any action to be taken in the Patent and Trademark Office regarding this application without direct communication between (1) the patent attorneys and patent agents and (2) each of the undersigned. In the event of a change in the persons from whom instructions may be taken, at least one patent attorney or patent agent named herein will be so notified by at least one of the undersigned.

Direct all correspondence to: ☒ The address associated with Customer Number: **08791** OR ☐ Correspondence address below

Name

Address

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
1279 Oakmead Parkway

City: **Sunnyvale** State: **California** Zip: **94085-4040**

Country: **U.S.A.** Telephone: **(408) 720-8300** Email: **MAIL@BSTZ.com**

WARNING:

Petitioner/applicant is cautioned to avoid submitting personal information in documents filed in a patent application that may contribute to identity theft. Personal information such as social security numbers, bank account numbers, or credit card numbers (other than a check or credit card authorization form PTO-2038 submitted for payment purposes) is never required by the USPTO to support a petition or an application. If this type of personal information is included in documents submitted to the USPTO, petitioners/applicants should consider redacting such personal information from the documents before submitting them to the USPTO. Petitioner/applicant is advised that the record of a patent application is available to the public after publication of the application (unless a non-publication request in compliance with 37 CFR 1.213(a) is made in the application) or issuance of a patent. Furthermore, the record from an abandoned application may also be available to the public if the application is referenced in a published application or an issued patent (see 37 CFR 1.14). Checks and credit card authorization forms PTO-2038 submitted for payment purposes are not retained in the application file and therefore are not publicly available. Petitioner/applicant is advised that documents which form the record of a patent application (such as the PTO/SB/01) are placed into the Privacy Act system of records DEPARTMENT OF COMMERCE, COMMERCE-PAT-7, System name: *Patent Application File*. Documents not retained in an application file (such as the PTO-2038) are placed into the Privacy Act system of COMMERCE/PAT-TM-10, System name: *Deposit Accounts and Electronic Funds Transfer Profiles*.

Legal Name of Sole or First Inventor:

(E.g., Given Name (first and middle (if any)) and Family Name or Surname)

Sandip AGARWALA

Inventor's Signature

Sandip Agarwala

Date

12/19/2013

Residence: City: **Cupertino** State: **CA** Country: **USA**

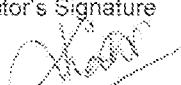
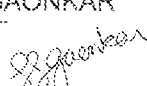
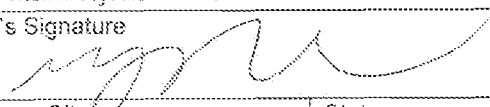
Mailing Address
10630 Mine Court

City: **Cupertino** State: **CA** Zip: **95014** Country: **USA**

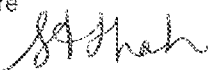
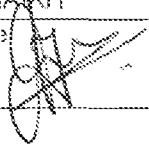
[Page 2]

Based on Forms PTO/AIA/08 and PTO/AIA/10 (06-12) as modified by Blakely, Sokoloff, Taylor & Zafman LLP on 09/17/2012.

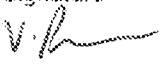
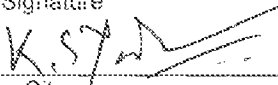
DECLARATION AND POWER OF ATTORNEY— Utility or Design Patent Application

Legal Name of Additional Joint Inventor, if any:			
(E.g., Given Name (first and middle (if any)) and Family Name or Surname) Abhishek CHATURVEDI			
Inventor's Signature 		Date Dec 19 th 2013	
Residence: City Mountain View	State CA	Country USA	
Mailing Address 2114 Windrose Place			
City Mountain View	State CA	Zip 94043	Country USA
Legal Name of Additional Joint Inventor, if any:			
(E.g., Given Name (first and middle (if any)) and Family Name or Surname) Shravan GAONKAR			
Inventor's Signature 		Date 12/19/2013	
Residence: City Gainesville	State FL	Country USA	
Mailing Address 4000 NW 51st Street, Apartment B37			
City Gainesville	State FL	Zip 32606	Country USA
Legal Name of Additional Joint Inventor, if any:			
(E.g., Given Name (first and middle (if any)) and Family Name or Surname) Mallikarjunan MAHALINGAM			
Inventor's Signature 		Date 12/19/2013	
Residence: City Cupertino	State CA	Country USA	
Mailing Address 22009 Oakdell Place			
City Cupertino	State CA	Zip 95014	Country USA

DECLARATION AND POWER OF ATTORNEY— Utility or Design Patent Application

Legal Name of Additional Joint Inventor, if any:			
(E.g., Given Name (first and middle (if any)) and Family Name or Surname) Sazzala REDDY			
Inventor's Signature			Date
Residence: City Los Altos	State CA	Country USA	
Mailing Address 530 Torwood Court			
City Los Altos	State CA	Zip 94022	Country USA
Legal Name of Additional Joint Inventor, if any:			
(E.g., Given Name (first and middle (if any)) and Family Name or Surname) Smit SHAH			
Inventor's Signature 			Date 12/19/2013
Residence: City Sunnyvale	State CA	Country USA	
Mailing Address 1229 Vicente Drive, Apartment 28			
City Sunnyvale	State CA	Zip 94086	Country USA
Legal Name of Additional Joint Inventor, if any:			
(E.g., Given Name (first and middle (if any)) and Family Name or Surname) Faraz SHAIKH			
Inventor's Signature 			Date 12/19/2013
Residence: City Sunnyvale	State CA	Country USA	
Mailing Address 1000 Escalon Avenue, Apartment Q2132			
City Sunnyvale	State CA	Zip 94085	Country USA

DECLARATION AND POWER OF ATTORNEY— Utility or Design Patent Application

Legal Name of Additional Joint Inventor, if any:			
(E.g., Given Name (first and middle (if any)) and Family Name or Surname) Praveen VEGULLA			
Inventor's Signature 		Date 12/19/13	
Residence: City Cupertino	State CA	Country USA	
Mailing Address 10846 Willowbrook Way			
City Cupertino	State CA	Zip 95014	Country USA
Legal Name of Additional Joint Inventor, if any:			
(E.g., Given Name (first and middle (if any)) and Family Name or Surname) Krishna YADAPPANAVAR			
Inventor's Signature 		Date 12/19/13	
Residence: City Sunnyvale	State CA	Country USA	
Mailing Address 420 Cirrus Avenue			
City Sunnyvale	State CA	Zip 94087	Country USA
Legal Name of Additional Joint Inventor, if any:			
(E.g., Given Name (first and middle (if any)) and Family Name or Surname) Jeffrey A. ZABARSKY			
Inventor's Signature		Date	
Residence: City Sunnyvale	State CA	Country USA	
Mailing Address 990 Ticonderoga Drive			
City Sunnyvale	State CA	Zip 94087	Country USA

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re Application:

Examiner: Not yet assigned

First Named Inventor:

Art Unit: Not yet assigned

Sandip AGARWALA

Confirmation No: Not yet assigned

Application No. Not yet assigned

Filed: Herewith

For: SYSTEMS AND METHODS FOR
IMPLEMENTING AN ENTERPRISE-CLASS
CONVERGED COMPUTE-NETWORK-
STORAGE APPLIANCE

EFS Filing
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

INFORMATION DISCLOSURE STATEMENT

Sir:

Enclosed is a copy of Information Disclosure Citation Form PTO-1449 or PTO/SB/08 together with copies of the documents cited on that form, except for copies not required to be submitted (e.g., copies of U.S. patents and U.S. published patent applications need not be enclosed). It is respectfully requested that the cited documents be considered and that the enclosed copy of Information Disclosure Citation Form PTO-1449 or PTO/SB/08 be initialed by the Examiner to indicate such consideration and a copy thereof returned to applicant(s).

Pursuant to 37 C.F.R. § 1.97, the submission of this Information Disclosure Statement is not to be construed as a representation that a search has been made and is not to be construed as an admission that the information cited in this statement is material to patentability.

Pursuant to 37 C.F.R. § 1.97, this Information Disclosure Statement is being submitted under one of the following (as indicated by an "X" to the left of the appropriate paragraph):

- X 37 C.F.R. §1.97(b).
- 37 C.F.R. §1.97(c). If so, then enclosed with this Information Disclosure Statement is one of the following:
- A statement pursuant to 37 C.F.R. §1.97(e) or
- Authorization for \$180.00 for the fee under 37 C.F.R. § 1.17(p).
- 37 C.F.R. §1.97(d). If so, then enclosed with this Information Disclosure Statement are the following:
- (1) A statement pursuant to 37 C.F.R. §1.97(e); and
 - (2) Authorization for \$180.00 for the fee under 37 C.F.R. §1.17(p) for submission of the Information Disclosure Statement.

If there are any additional charges, you are hereby authorized to charge Deposit Account No. 02-2666.

Respectfully submitted,

BLAKELY SOKOLOFF TAYLOR ZAFMAN LLP

Date: December 19, 2013

/Eric S. Replogle/
Eric S. Replogle
Reg. No. 52,161

1279 Oakmead Parkway
Sunnyvale, CA 94085-4040
(408) 720-8300

Document code: WFEE

United States Patent and Trademark Office
Sales Receipt for Accounting Date: 01/09/2014

SDUONG	SALE	#00000039	Mailroom Dt:	12/19/2013	022666	14135485
		01	FC : 1202	80.00	DA	
		02	FC : 1051	140.00	DA	