



(19) **United States**

(12) **Patent Application Publication**

**Yang et al.**

(10) **Pub. No.: US 2018/0069944 A1**

(43) **Pub. Date: Mar. 8, 2018**

(54) **AUTOMATIC DATA REPLICA MANAGER IN DISTRIBUTED CACHING AND DATA PROCESSING SYSTEMS**

**Publication Classification**

(51) **Int. Cl.**  
*H04L 29/08* (2006.01)  
*H04L 12/24* (2006.01)  
*H04L 12/26* (2006.01)

(52) **U.S. Cl.**  
 CPC ..... *H04L 67/2842* (2013.01); *H04L 41/5022* (2013.01); *H04L 67/1002* (2013.01); *H04L 43/0888* (2013.01); *H04L 41/0672* (2013.01); *H04L 41/0677* (2013.01); *H04L 43/0852* (2013.01); *H04L 67/148* (2013.01)

(71) Applicant: **Samsung Electronics Co., Ltd.**, Suwon-si (KR)

(72) Inventors: **Zhengyu Yang**, Boston, MA (US); **Jiayin Wang**, Dorchester, MA (US); **Thomas David Evans**, San Marcos, CA (US)

(21) Appl. No.: **15/408,328**

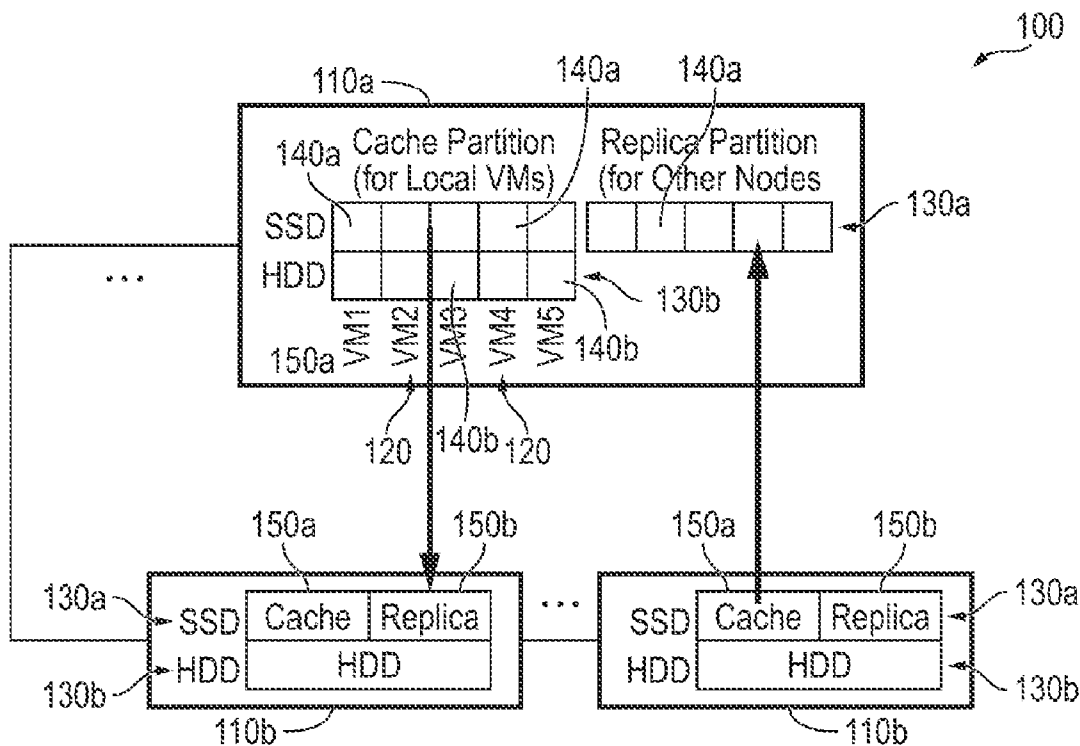
(22) Filed: **Jan. 17, 2017**

**Related U.S. Application Data**

(60) Provisional application No. 62/384,078, filed on Sep. 6, 2016, provisional application No. 62/404,167, filed on Oct. 4, 2016.

(57) **ABSTRACT**

A method of data storage includes determining a latency distance from a primary node to each of two or more replica nodes, choosing a preferred replica node of the two or more replica nodes based on the determined latency distances, and write-caching data into the preferred replica node.



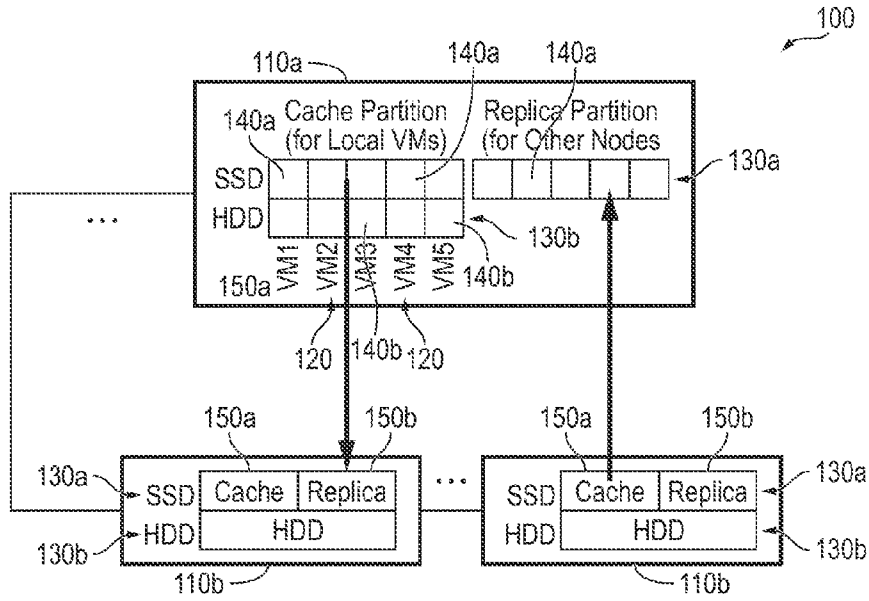


FIG. 1

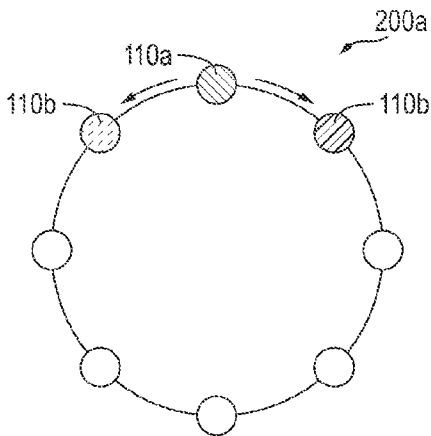


FIG. 2A

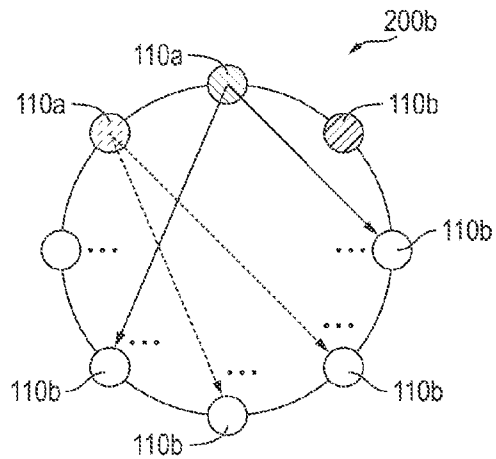


FIG. 2B

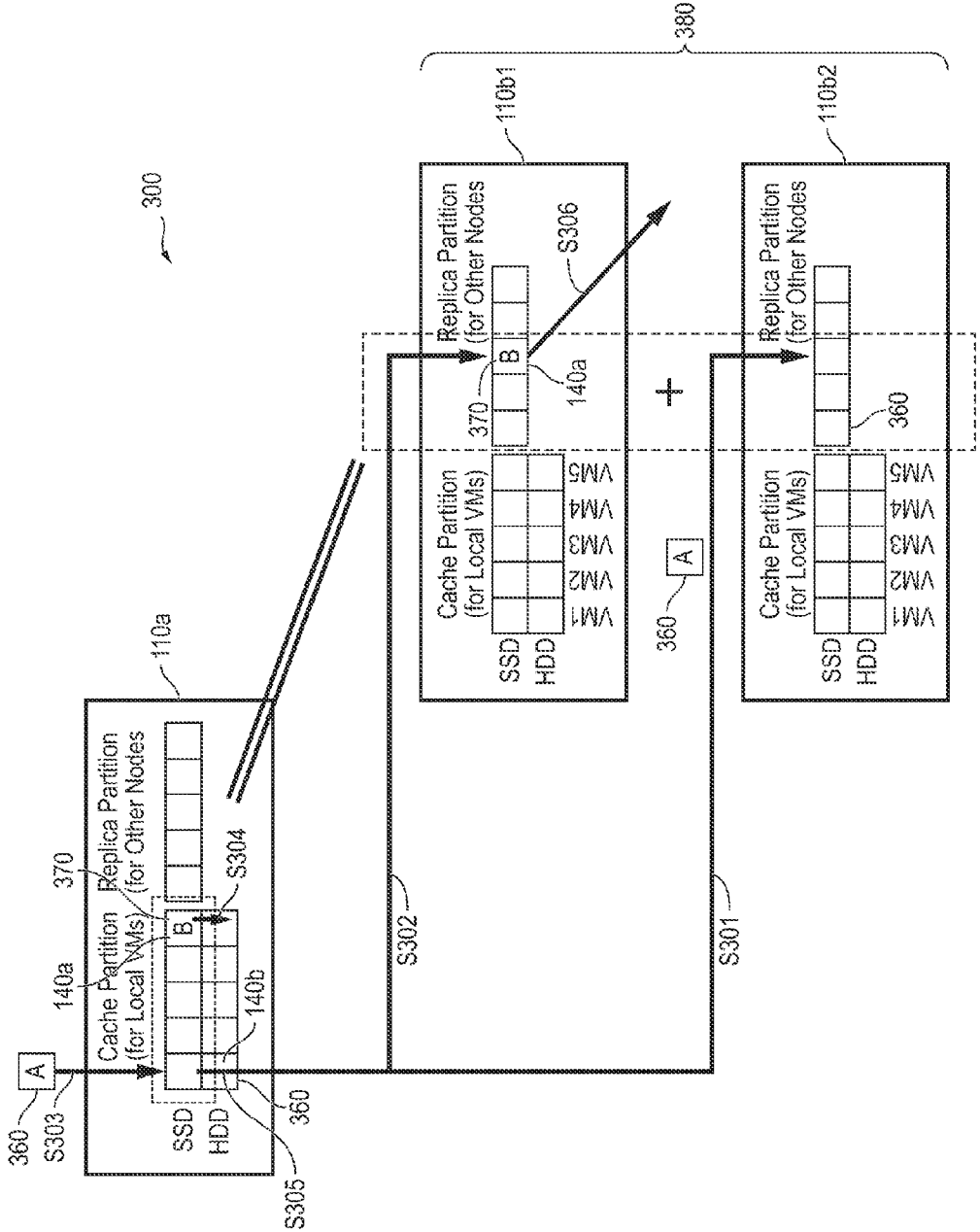


FIG. 3

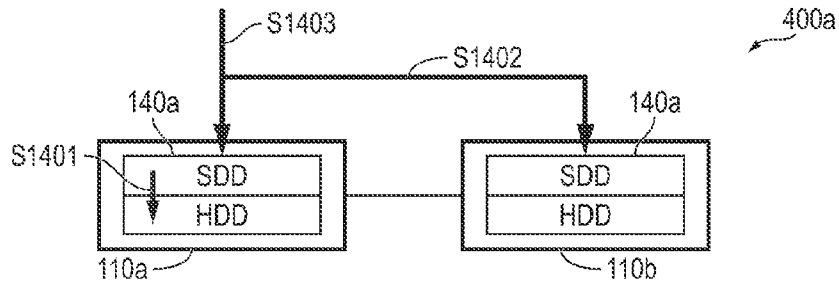


FIG. 4A

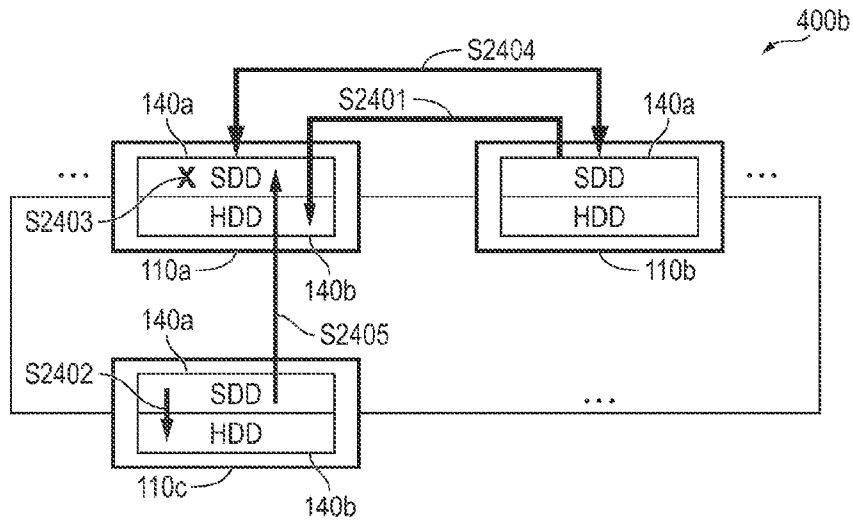


FIG. 4B

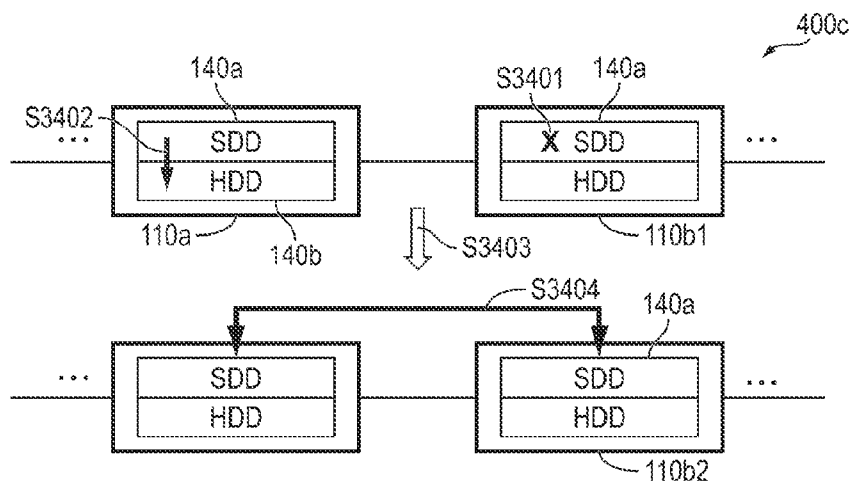


FIG. 4C

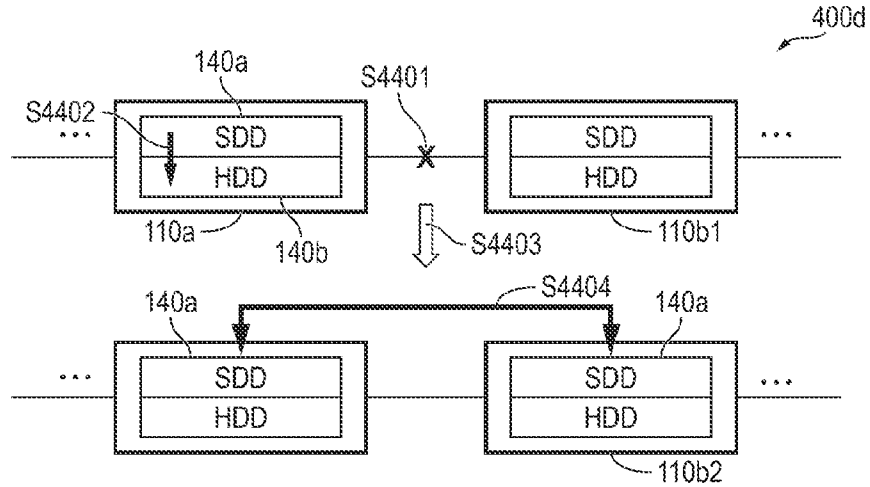


FIG. 4D

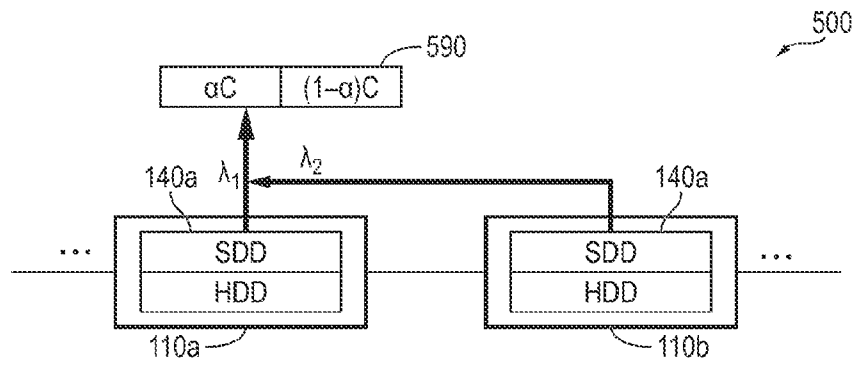


FIG. 5

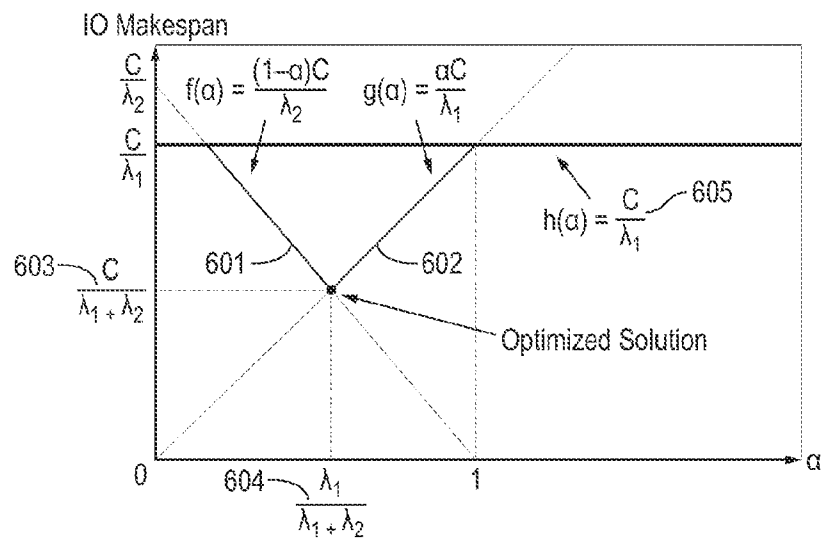


FIG. 6

**AUTOMATIC DATA REPLICA MANAGER IN DISTRIBUTED CACHING AND DATA PROCESSING SYSTEMS**

**CROSS-REFERENCE TO RELATED APPLICATION(S)**

**[0001]** This application claims priority to, and the benefit of, U.S. Provisional Application 62/404,167, filed on Oct. 4, 2016 in the U.S. Patent and Trademark Office, the entire content of which is incorporated herein by reference.

**FIELD**

**[0002]** One or more aspects of embodiments according to the present invention generally relate to datacenter operation including data storage and data recovery.

**BACKGROUND**

**[0003]** In the field of memory storage, datacenters that store large amounts of memory seek to accomplish high capacity, high performance, low power usage, and low cost.

**[0004]** Replication is the process of synchronizing data across multiple storage nodes of a datacenter cluster, which provides redundancy, and which increases data availability from the loss of a single storage node. Replication allows the system to recover from hardware failure and service interruptions, and even to increase read capacity. Thus, in an era of big data and cloud computing, storing multiple replicas (e.g., replicated data sets) is a common strategy for reliability and availability of datasets stored over remote cloud storage.

**[0005]** Additionally, certain databases, such as bank, stock market, and military databases, require that the data stored therein is replicable and is reliably stored.

**SUMMARY**

**[0006]** Aspects of embodiments of the present disclosure are directed toward datacenter operation, including data storage and data recovery.

**[0007]** According to an embodiment of the present invention, there is provided a method of data storage, the method including determining a latency distance from a primary node to each of two or more replica nodes, choosing a preferred replica node of the two or more replica nodes based on the determined latency distances, and write-caching data into the preferred replica node.

**[0008]** The two or more replica nodes may each have a smaller latency distance to the primary node than all other available nodes of a datacenter cluster including the primary node and the two or more replica nodes.

**[0009]** Each node of a datacenter cluster as a primary node may have a ranking of preferred replica nodes that are ranked based on network delays associated with paths between the primary node and respective ones of the replica nodes, and each node of the datacenter cluster may include a solid-state drive tier as a cache tier including a cache partition for storing data of local virtual machines and a replica partition for storing replica data from other nodes, and a hard disk drive tier as a back end storage tier.

**[0010]** Each node may include a physical host for running multiple virtual machines, each of the virtual machines sharing the solid-state drive tier and the hard disk drive tier in a respective node of the datacenter cluster, the method

further including periodically updating the rankings of the preferred replica nodes to account for changes in network delays.

**[0011]** The rankings of the preferred replica nodes may be further based on workload characteristics including access frequency of data associated with the nodes and service level agreements (SLAs) associated with the nodes.

**[0012]** According to another embodiment of the present invention, there is provided a method for enabling fault tolerance of a datacenter cluster caching system, the method including detecting a failure of the datacenter cluster, and performing load balancing among nodes of the datacenter cluster in accordance with the detected failure by moving data among storage tiers of the nodes, wherein the load balancing is performed without pausing a subsystem corresponding to the detected failure, and without copying an entirety of a dataset of a node of the datacenter cluster corresponding to the detected failure.

**[0013]** The method may further include migrating replica data from a first replica node in the datacenter cluster to a second replica node in a datacenter cluster by writing new incoming data on the second replica node, and maintaining unchanged cached data on the first replica node.

**[0014]** The method may further include evicting victim data from a solid-state drive of a primary node, receiving the new incoming data at the primary node, writing the new incoming data to a hard disk drive of the primary node, and removing the victim data from a solid-state drive of the first replica node.

**[0015]** The detected failure may include a virtual machine crash on a primary node of the datacenter cluster, and the method may further include writing dirty data from a solid-state drive of the primary node to a hard disk drive of the primary node, maintaining the dirty data in a solid-state drive of a replica node, flagging the dirty data in the solid-state drive of the replica node as nondirty, restarting the crashed virtual machine on the primary node, and forwarding incoming I/O requests to the solid-state drive of the primary node and to the solid-state drive of the replica node.

**[0016]** The detected failure may include a cache device failure on a primary node of the datacenter cluster such that information cannot be written to the primary node, and the method may further include writing dirty data from a solid-state drive of a replica node of the datacenter cluster to a hard disk drive of the primary node, maintaining the dirty data in the solid-state drive of the replica node, flagging the dirty data as nondirty, broadcasting the information to associated nodes of the datacenter cluster, and writing back dirty data from respective solid-state drives of the associated nodes of the datacenter cluster to respective hard disk drives of the associated nodes.

**[0017]** The method may further include flagging data in the solid-state drives of the associated nodes with a nondirty flag, replacing the solid-state drive on the primary node, continuing to write incoming I/O requests to the solid-state drive of the primary node and the solid-state drive of the replica node, and writing new replica data from the associated nodes to the solid-state drive of the primary node.

**[0018]** The detected failure may include a cache device failure on a first replica node of the datacenter cluster, and the method may further include disconnecting a primary node of the datacenter cluster, writing back dirty data from a solid-state drive of the primary node to a hard disk drive

of the primary node, finding a remote second replica node of the datacenter cluster, storing the dirty data in a solid-state drive of the second replica node, flagging the dirty data in the solid-state drive of the second replica node as nondirty, and continuing to write incoming I/O requests to the solid-state drive of the primary node and the solid-state drive of the second replica node.

[0019] Finding a remote second replica node of the datacenter cluster may include using a dynamic evaluation process.

[0020] The detected failure may include a communication failure between a primary node of the datacenter cluster and a first replica node of the datacenter cluster, and the method may further include writing back dirty data from a solid-state drive of the primary node to a hard disk drive of the primary node, finding a second replica node to replace the first replica node, writing I/O requests to a solid-state drive of the primary node and a solid-state drive of the second replica node, and broadcasting to all nodes of the datacenter cluster a request to release all old replica data corresponding to the first replica node.

[0021] Finding the second replica node may include using a dynamic evaluation process.

[0022] According to another embodiment of the present invention, there is provided a method of using replicated data to enable parallel prefetching from multiple nodes of a datacenter cluster, the method including splitting a dataset into two parts, and loading each of the two parts of the dataset from a corresponding node.

[0023] The corresponding nodes for each of the two parts of the dataset may include a primary node of the datacenter cluster, and a replica node of the datacenter cluster.

[0024]  $C$  may be a total size of the dataset,  $\alpha C$  may be a size of the one of the two parts from a solid-state drive of the primary node,  $\lambda_1$  may be an access speed of the solid-state drive of the primary node, and  $\lambda_2$  may be an access speed of a solid-state drive of the replica node, and the method may further include triggering parallel prefetching when  $C/\lambda_1$  is greater than or equal to a maximum of  $\alpha C/\lambda_1$  and  $(1-\alpha)C/\lambda_2$ .

[0025]  $\lambda_1$  and  $\lambda_2$  may each account for corresponding network delays.

[0026] The method may further include achieving a network makespan of

$$\frac{C}{\lambda_1 + \lambda_2} \text{ when } \alpha = \frac{\lambda_1}{\lambda_1 + \lambda_2}.$$

BRIEF DESCRIPTION OF THE DRAWINGS

[0027] FIG. 1 is a block diagram depicting a topological structure of a datacenter cluster, according to an embodiment of the present invention;

[0028] FIG. 2A depicts a method of selecting and assigning replica nodes of a datacenter cluster using a ring structure, according to an embodiment of the present invention;

[0029] FIG. 2B depicts a method of selecting and assigning replica nodes of a datacenter cluster using a network structure, according to an embodiment of the present invention;

[0030] FIG. 3 is a block diagram depicting an online migration cache policy, according to an embodiment of the present invention;

[0031] FIGS. 4A-4D are block diagrams depicting data recovery for different scenarios, according to one or more embodiments of the present invention;

[0032] FIG. 5 is a block diagram depicting an example of using replicate data to enable parallel prefetching from multiple nodes, according to an embodiment of the present invention; and

[0033] FIG. 6 is a graphical depiction of mathematical equations for finding an optimized solution of stream division, according to an embodiment of the present invention.

DETAILED DESCRIPTION

[0034] Features of the inventive concept and methods of accomplishing the same may be understood more readily by reference to the following detailed description of embodiments and the accompanying drawings. Hereinafter, example embodiments will be described in more detail with reference to the accompanying drawings, in which like reference numbers refer to like elements throughout. The present invention, however, may be embodied in various different forms, and should not be construed as being limited to only the illustrated embodiments herein. Rather, these embodiments are provided as examples so that this disclosure will be thorough and complete, and will fully convey the aspects and features of the present invention to those skilled in the art. Accordingly, processes, elements, and techniques that are not necessary to those having ordinary skill in the art for a complete understanding of the aspects and features of the present invention may not be described. Unless otherwise noted, like reference numerals denote like elements throughout the attached drawings and the written description, and thus, descriptions thereof will not be repeated. In the drawings, the relative sizes of elements, layers, and regions may be exaggerated for clarity.

[0035] It will be understood that, although the terms “first,” “second,” “third,” etc., may be used herein to describe various elements, components, regions, layers and/or sections, these elements, components, regions, layers and/or sections should not be limited by these terms. These terms are used to distinguish one element, component, region, layer or section from another element, component, region, layer or section. Thus, a first element, component, region, layer or section described below could be termed a second element, component, region, layer or section, without departing from the spirit and scope of the present invention.

[0036] Spatially relative terms, such as “beneath,” “below,” “lower,” “under,” “above,” “upper,” and the like, may be used herein for ease of explanation to describe one element or feature’s relationship to another element(s) or feature(s) as illustrated in the figures. It will be understood that the spatially relative terms are intended to encompass different orientations of the device in use or in operation, in addition to the orientation depicted in the figures. For example, if the device in the figures is turned over, elements described as “below” or “beneath” or “under” other elements or features would then be oriented “above” the other elements or features. Thus, the example terms “below” and “under” can encompass both an orientation of above and below. The device may be otherwise oriented (e.g., rotated

90 degrees or at other orientations) and the spatially relative descriptors used herein should be interpreted accordingly.

**[0037]** It will be understood that when an element, layer, region, or component is referred to as being “on,” “connected to,” or “coupled to” another element, layer, region, or component, it can be directly on, connected to, or coupled to the other element, layer, region, or component, or one or more intervening elements, layers, regions, or components may be present. In addition, it will also be understood that when an element or layer is referred to as being “between” two elements or layers, it can be the only element or layer between the two elements or layers, or one or more intervening elements or layers may also be present.

**[0038]** The terminology used herein is for the purpose of describing particular embodiments only and is not intended to be limiting of the present invention. As used herein, the singular forms “a” and “an” are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms “comprises,” “comprising,” “includes,” and “including,” when used in this specification, specify the presence of the stated features, integers, steps, operations, elements, and/or components, but do not preclude the presence or addition of one or more other features, integers, steps, operations, elements, components, and/or groups thereof. As used herein, the term “and/or” includes any and all combinations of one or more of the associated listed items. Expressions such as “at least one of,” when preceding a list of elements, modify the entire list of elements and do not modify the individual elements of the list.

**[0039]** As used herein, the term “substantially,” “about,” and similar terms are used as terms of approximation and not as terms of degree, and are intended to account for the inherent deviations in measured or calculated values that would be recognized by those of ordinary skill in the art. Further, the use of “may” when describing embodiments of the present invention refers to “one or more embodiments of the present invention.” As used herein, the terms “use,” “using,” and “used” may be considered synonymous with the terms “utilize,” “utilizing,” and “utilized,” respectively. Also, the term “exemplary” is intended to refer to an example or illustration.

**[0040]** When a certain embodiment may be implemented differently, a specific process order may be performed differently from the described order. For example, two consecutively described processes may be performed substantially at the same time or performed in an order opposite to the described order.

**[0041]** The electronic or electric devices and/or any other relevant devices or components according to embodiments of the present invention described herein may be implemented utilizing any suitable hardware, firmware (e.g. an application-specific integrated circuit), software, or a combination of software, firmware, and hardware. For example, the various components of these devices may be formed on one integrated circuit (IC) chip or on separate IC chips. Further, the various components of these devices may be implemented on a flexible printed circuit film, a tape carrier package (TCP), a printed circuit board (PCB), or formed on one substrate. Further, the various components of these devices may be a process or thread, running on one or more processors, in one or more computing devices, executing computer program instructions and interacting with other system components for performing the various functional-

ities described herein. The computer program instructions are stored in a memory which may be implemented in a computing device using a standard memory device, such as, for example, a random access memory (RAM). The computer program instructions may also be stored in other non-transitory computer readable media such as, for example, a CD-ROM, flash drive, or the like. Also, a person of skill in the art should recognize that the functionality of various computing devices may be combined or integrated into a single computing device, or the functionality of a particular computing device may be distributed across one or more other computing devices without departing from the spirit and scope of the exemplary embodiments of the present invention.

**[0042]** Unless otherwise defined, all terms (including technical and scientific terms) used herein have the same meaning as commonly understood by one of ordinary skill in the art to which the present invention belongs. It will be further understood that terms, such as those defined in commonly used dictionaries, should be interpreted as having a meaning that is consistent with their meaning in the context of the relevant art and/or the present specification, and should not be interpreted in an idealized or overly formal sense, unless expressly so defined herein.

**[0043]** FIG. 1 is a block diagram depicting a topological structure of a datacenter cluster, according to an embodiment of the present invention.

**[0044]** Referring to FIG. 1, a datacenter cluster **100** includes multiple nodes **110** arranged in a cluster. Each node **110** may be a physical host (e.g., a server) running multiple virtual machines (VMs) **120**. The VMs **120** may be run on either a type-1 hypervisor or a type-2 hypervisor. For example, the host used to run the VMs **120** in the present embodiment is a type-1 implementation (e.g., the present embodiment may use ESXi hosts for each node **110**).

**[0045]** Inside each node **110**, there may be two tiers **130** of data storage devices/memory devices **140**. The two storage tiers **130** may include a solid-state drive (SSD) tier **130a**, and a hard disk drive (HDD) tier **130b**. It should be noted that, although SSDs and HDDs are used to describe the various embodiments below, other embodiments may use different cache/tier devices and/or memory devices.

**[0046]** Each storage tier **130** includes one or more memory devices **140**. For example, the SSD tier **130a** may include one or more SSDs **140a**, and the HDD tier **130b** may include one or more HDDs **140b**. RAID mode disks (e.g., disks operating as a redundant array of independent disks) can also be adopted in each of the storage tiers **130**. VMs **120** share the SSD tier **130a** and the HDD tier **130b** in each respective node **110**. The SSD tier **130a** may be used as a cache tier, and the HDD tier **130b** may be used as a backend storage tier.

**[0047]** Inside the SSD tier **130a** there are two partitions **150**. The two partitions **150** include a cache partition **150a**, which is for storing data for the local VMs **120**, and a replica partition **150b**, which is for storing replica data from other nodes **110**. In the present embodiment, only the SSD tier **130a** is used to store replica data from other nodes **110**. This may be suitable because the access speed of a remote SSD **140a**, even when network delay is accounted for, may be faster than the access speed of a local HDD **140b** (e.g., due to high speed fiber channels associated with the SSDs **140a**, and due to slower speed associated with the more durable HDDs **140b**).



[0048] According to different embodiments of the present invention, different approaches may be used to design the structure of the topology of the datacenter cluster 100. The type of approach that is used may be chosen by determining how to select replica nodes 110, how many replicas nodes 110 are suitable for the constraints of the system, and how to assign replica data to the replica nodes 110.

[0049] FIG. 2A depicts a method of selecting and assigning replica nodes of a datacenter cluster using a ring structure, according to an embodiment of the present invention.

[0050] Referring to FIG. 2A, a first approach is a directed ring structure 200a, which can be either a user-defined structure or a system-defined structure. The ring structure 200a may be thought of as a linear structure. A system-defined ring structure 200a may be based on geographic “distance” parameters (e.g., a latency distance based on an effective distance corresponding to a total network delay). The logical ring structure 200a shown defines an order of preference between a primary node 110a and replica nodes 110b. Caching may be performed by using a local SSD 140a with a copy replicated to another replica node 110b in the ring structure 200a. Each node 110 has two neighbors (e.g., a replica node 110b on each side of each primary node 110a), and replicated data may be stored on one or both of the neighboring replica nodes 110b. The primary node 110a searches for the replica nodes 110b of the ring structure 200a in order of geographic distance/latency distance until a replica node 110b is found that can be used in the event that the primary node 110a is unsuccessful during the process of building the ring cluster 200a. Once the primary node 110a finds a suitable replica node 110b, the primary node 110a may begin write-caching independently of what the remaining replica nodes 110b are doing.

[0051] FIG. 2B depicts a method of selecting and assigning replica nodes of a datacenter cluster using a network structure, according to an embodiment of the present invention.

[0052] Referring to FIG. 2B, a second approach is directed to a network structure 200b. The ring structure 200a shown in FIG. 2A is effectively a linear approach, because the primary node 110a has only one or two directions to find replica nodes 110b (e.g., a previous replica node 110b or a next replica node 110b). The network structure 200b shown in FIG. 2B, however, provides additional flexibility by allowing for access by the primary node 110a to a larger number of different replica nodes 110b. The additional access may be achieved by letting the different nodes 110 maintain a list of preferred replica nodes 110b. Further, it should be noted that the cluster structure in a network structure 200b is not limited to the ring-like orientation shown. For example, the network structure 200b can be either symmetric or asymmetric.

[0053] In structuring the network structure 200b, a distance matrix may be used to rank each path between nodes 110 (see Table 1, below). Accordingly, each node 110 acting as a primary node 110a may attempt to use multiple different replica nodes 110b based on an order corresponding to the distance matrix.

[0054] The distance matrix for each node 110, may be thought of as including scores or rankings that are calculated based on multiple parameters (e.g., based on network delays between respective nodes 110). These scores/rankings effectively reflect an abstract meaning of the aforementioned

geographical distance/latency distance. The distance matrix may be periodically updated through runtime measurement.

[0055] A distance matrix of an example of the present embodiment may be seen in Table 1 below. For example, a first node as a primary node 110a may first seek to use a second node as a replica node 110b, followed by a fourth node, and then a third node. However, the second node, when acting as a primary node 110a, may first seek to use the fourth node as a replica node 110b, followed by the third node, and then the first node. Furthermore, the third node, when acting as a primary node 110a, may initially seek the fourth node, then the second node, and then the first node as a replica node 110b, while the order for the fourth node, when acting as a primary node 110a, may be from the first node to the third node to the second node.

TABLE 1

From	To			
	1	2	3	4
1	—	1	3	2
2	3	—	2	1
3	3	2	—	1
4	1	3	2	—

[0056] Different processes may be used for assigning a preferred order of the replica nodes 110b for each node 110 as a primary node 110a. For example, if only one replica node 110b is sufficient for each primary node 110a, then each node 110 may select its “closest” node as its replica node 110b. As another example, if a large number of nodes 110 choose a same single node 110 (or choose a small subset of nodes 110) as the preferred replica node(s) 110b, a maximum number of replica nodes per primary node may be set.

[0057] Accordingly, the network structure 200b may be set such that each primary node 110a has only one replica node 110b, and such that all nodes 110 have the same hardware resources (e.g., a homogeneous cluster). However, the network structure 200b may also work in a heterogeneous environment.

[0058] According to another embodiment, a multiple service-level agreement (SLA) network approach may be used. The multiple-SLA network approach further protects data consistency and atomicity, and improves the previous network structure approach discussed with reference to FIG. 2B. A SLA may be a part of a standardized service contract where a service is formally defined. Each node 110 may have more than one replica node 110b, and the replica nodes 110b may be determined or assigned based on workload characteristics and the SLA (e.g., based on how much a corresponding client or customer pays, as preferential treatment may be purchased, and based on a corresponding workload temperature). Table 2 shows a decision table for different cases, which may be used to set up the cluster multiple-SLA network approach, according to an example of an embodiment of the present invention.

TABLE 2

Case #	Trace		Destination				# of Replica
	SLA Degree	Temperature	SSD <sub>p</sub>	SSD <sub>R1</sub>	SSD <sub>R2</sub>	HDD <sub>p</sub>	
1		✓	✓	✓			1
2			✓		✓		1
3	✓	✓	✓	✓	(✓)		1(2)
4	✓		✓	✓		(✓)	1(2)

[0059] A SLA degree may be related by assigning a level of importance of each node 110. The present embodiment can handle multiple SLA levels by using multiple replica approaches. The present example has only two degrees (e.g., important and not important). In other embodiments, more fine-grained degrees of SLAs may be set. Moreover, an online-changing SLA can also be adopted into embodiments of the present invention.

[0060] Additionally, “data temperature” may be used as an indicator for classifying data into two (or more) categories based on the access frequency of the data (e.g., “hot data” has a frequent access pattern, and “cold data” is occasionally queried).

[0061] A write back policy may be used such that eviction from one SSD 140a to another SSD 140a when the SSD 140a is full allows for the data to be simply removed. To avoid affecting write-backs from a SSD 140a to a HDD 140b, and to avoid affecting fetching operations from a HDD 140b to a SSD 140a, a lower priority may be established for replicas writing to the HDD. Many techniques may be adopted to improve performance of a write-to-HDD queue.

[0062] A main procedure of a cache policy may be based on the multiple-SLA network approach. As discussed above, the present embodiment may support more than one replica node 110b for each primary node 110a, and the present embodiment may be expanded to any number of replica nodes 110b to support more finely grained SLAs. Accordingly, the present embodiment is able to switch between two different cache modes, namely a runtime mode and an online migration mode.

[0063] The datacenter cluster 100 may periodically check for the existence of a “migration trigger” condition. If it is determined that the “migration trigger” condition exists, one of the two replica nodes 100b is selected as an “overheat node,” and a next available replica node 110b is sought. After that, a migration mode cache policy may be initiated. If it is determined that an old replica node 110b no longer has any “out-of-date” replica data, then the migration mode cache policy may be turned off, while the runtime mode cache policy is run in all other cases.

[0064] In case #s 3 and 4 of Table 2, above, it may be assumed that crashes/failures of VMs 120 mainly affect cached data on the SSDs 140a, and do not affect data on the HDD 140b. In the cache policy of the present embodiment, there are only three cases where the cached data is written to the HDD 140b (e.g., during eviction write back, during migration time, and when the queued replica is written to HDD 140b).

[0065] A runtime mode cache policy may be thought of as a normal mode, or a default mode, of the datacenter cluster 100. The present embodiment may first search a new I/O request in a local SSD VM partition (e.g., in the SSD 140a of the primary node 110a). If it is determined that a cache hit

is returned, the data may be fetched from the SSD 140a of the primary node 110a for a read I/O operation, or the new data may be updated to existing cached copies in the SSD 140a of the primary node 110a and to one or more replica nodes 110b.

[0066] When it is determined that there is a cache miss, victim data may initially be selected to be evicted from the primary node 110a and from the replica nodes 110b. Then, only updated (dirty) data may be written into the HDD 140b of the primary node 110a. Different replacement algorithms may be used for implementation of the victim data selection process. The selected victim data may then be written back to the HDD 140b of the primary node 110a if the victim data is determined to be dirty. That is, the victim data may be updated in the SSD 140a while not being updated in the HDD 140b. Thereafter, all copies of the victim data may be deleted from both local SSDs 140a and remote SSDs 140a. Then the new data may be inserted into both the primary node 110a and the replica node(s) 110b.

[0067] In detail, if the victim data corresponds to a read I/O operation, then the victim data may simply be fetched from the HDD 140b to the SSDs 140a of the primary node 110a and the replica nodes 110b. Additionally, the fetched cached data may be returned to a user buffer in the memory. If the victim data corresponds to a write I/O operation, the victim data may simply be written to the SSDs 140a of the primary node 110a, and also to all corresponding replica nodes 110b, while being flagged as being “dirty” data, because the data is updated new data.

[0068] FIG. 3 is a block diagram depicting an online migration cache policy, according to an embodiment of the present invention.

[0069] Referring to FIG. 3, when migrating replica data from one replica node 110b to another replica node 110b, a fusion cache technique 300 may be used. The fusion cache technique 300 is a cost-efficient, lazy migration scheme. Accordingly, new incoming data 360 may be written only on a new replica node (e.g., a second replica node) 110b2 (S301), and unchanged cached data/victim data 370 may be kept on an old replica node (e.g., a first replica node) 110b1 (S302). Accordingly, the present embodiment is able to mirror the primary node 110a by using a fusion cache 380 consisting of two nodes (e.g., the new replica node 110b2 and the old replica node 110b1). By not pausing the system while performing the copying of all existing replica data pieces from the old replica node 110b1 to the new replica node 110b2 (regardless of whether these data pieces are needed), the proposed fusion cache can save system bandwidth.

[0070] FIG. 3 shows that only one replica node (e.g., the old replica node 110b1) is to be “migrated out” with the new replica node 110b2 receiving the data 370 that is migrated out, although the present embodiment may be applied to a system/datacenter cluster 100 in which more replica nodes 110b exist, including use of a HDD 140b in a replica node 110b. The primary node 110a may migrate its existing data piece/victim data 370 to a new replica node 110b2. When new data 360 comes to the primary node 110a (S303), victim data 370 may first be evicted from the SSD 140a of the primary node 110a (S304), and the new data 360 may be written to the HDD 140b of the primary node 110a (S305). Meanwhile, the new data 360 may also be written to the new replica node 110b2 (S301), and the old replica node 110b1 may be instructed to simply remove the victim data 370 from

its SSD **140a** (S306). The fusion cache **380** is a unibody of the new replica node **110b1**, and the old replica node **110b2** is a mirror of the primary node **110a**, where the new replica node **110b1** eventually replaces the old replica node **110b2**.

[0071] By using the online migration mode cache policy **300**, data is able to be migrated from the old replica node **110b2** to the new replica node **110b1**, while a remaining replica node **110b** may remain as a replica node **110b** that is used by the primary node **110a** (along with the new replica node **110b1**). For a two-SSD-replica node prototype, both replica nodes **110b1** and **110b2** can be the migration sources.

[0072] FIGS. 4A-4D are block diagrams depicting data recovery for different scenarios, according to one or more embodiments of the present invention.

[0073] Embodiments of the present invention may use a write back caching policy to maximize I/O performance, as writing through to the HDD **140b** would otherwise slow down the fetching I/O operations in the queue. However, storage in the SSD **140a** may be relatively unsafe, even though SSD storage preserves stored data after losing power. Such a lack of safety may not acceptable in some situations (e.g., bank databases, stock market databases, and military databases). Accordingly, data may be replicated in remote SSDs **140a** to prepare for recoveries for different failures. The possible types of failures may be categorized into the four scenarios discussed with respect to FIGS. 4A-4D, respectively.

[0074] Referring to FIG. 4A, a first recovery scenario may be referred to as a “VM crash on the primary node” scenario **400a**. In the present scenario **400a**, the VMDK may simply be closed out. Then, the “dirty” data may be written back from the SSD **140a** of the primary node **110a** to the HDD **140b** of the primary node **110a** (S1401). The dirty data may be kept in the SSD **140a** of the replica node **110b** while being flagged as “nondirty” (S1402). The crashed VM may then be restarted on the primary node **110a**, and incoming I/O requests may continue to be forwarded to both the SSD **140a** of the primary node **110a** and the SSD **140a** of the replica node **110b** (S1403).

[0075] Referring to FIG. 4B, a second recovery scenario may be referred to as a “primary node cache storage device failure” scenario **400b**, which will result in an inability to continue write caching. In the present scenario **400b**, the “dirty” data is first written back from the SSD **140a** of the replica node **110b** to the HDD **140b** of the primary node **110a** (S2401), and the data is kept in the SSD **140a** of the replica node **110b** while being flagged as “nondirty.” This “unavailable” information is then broadcast to the datacenter cluster **100** to let those nodes **110** having replicas in this failed primary node **110a** (e.g., associated nodes **110c**) to write back “dirty” data from their own SSD **140a** to HDD **140b** while keeping the data in their own SSD **140a** with a “nondirty” flag (S2402). The SSD **140a** on primary node **110** is then found and replaced (S2403). After that, incoming I/O requests continue to be written on both of the SSD **140a** of the primary node **110a** and the

[0076] SSD **140a** of the replica node **110b** (S2404), while the associated nodes **110c** write new replicas to the SSD **140a** of the primary node **110a** (S2405).

[0077] Referring to FIG. 4C, a third recovery scenario **400c** corresponds to when an old replica node **110b1** detects a cache device/SSD **140a** failure (S3401). In the present example, the primary node **110a** is disconnected, and any future connection attempts from the primary node **110a** are

rejected by an error response. First, the “dirty” data is written back from the SSD **140a** of the primary node **110a** to the HDD **140b** of the primary node **110a** (S3402). That data is also stored, while being flagged as “nondirty” flag data, in the SSD **140a** of a remote new replica node **110b2**, which may be found using a dynamic evaluation process (S3403). Thereafter, incoming I/O requests continue to be written on both the SSD **140a** of the primary node **110a** and the SSD **140a** of the new replica node **110b2** (S3404).

[0078] Referring to FIG. 4D, a fourth recovery scenario **400d** corresponds to a communication failure between a primary node **110a** and an old replica node **110b1**. When the primary node **110a** detects a non-recoverable communication failure between the primary host of the primary node **110a** and the replica host of the replica node **110b1** (S4401), it may not be possible to continue write caching. To recover from such failure, a daemon of the datacenter cluster **100** may write back “dirty” data from the SSD **140a** of the primary node **110a** to the HDD **140b** of the primary node **110a** to ensure all cached data is updated to the back end HDD **140b** (S4402). Then the daemon may start a dynamic evaluation process to find a new replica host, and to find a new replica node **110b2** to replace the unreachable old replica node **110b1** (S4403). The daemon then continues to use the SSDs **140a** of both the primary node **110a** and the new replica node **110b2** to cache I/Os following the “fusion cache” design in migration policy (S4404).

[0079] Finally, the daemon may broadcast to the network to release all old replicas on the unreachable the SSD **140a** of the replica node **110b1**.

[0080] FIG. 5 is a block diagram depicting an example of using replicate data to enable parallel prefetching from multiple nodes, according to an embodiment of the present invention.

[0081] Referring to FIG. 5, replicated data can also be used to enable parallel prefetching **500** from multiple nodes **110**, such as for read operations. In the present example, the data (e.g., data of a dataset) **590** is split into two parts **590a** and **590b**, and each part of the data **590** is loaded from its corresponding primary node **110a** and replica node **110b**.

[0082] For example, for a two-node case, it may be useful to reduce the total I/O time (e.g., the makespan of each I/O request). The makespan of parallel prefetching **500** is less than simply reading an entire piece of data from the primary node **110a**. In the present example, a total size of the data **590** is “C,” the access speed of the SSD **140a** of the primary node **110a** is “ $\lambda_1$ ” (e.g., in GB/sec), and the access speed of the SSD **140a** of the replica node **110b**, while accounting for network delays, is “ $\lambda_2$ ” (e.g., in GB/sec). Assuming data of the SSD **140a** of the primary node **110a** having a size of “ $\alpha C$ ” is loaded, the problem may be solved by determining a maximum between  $\alpha C/\lambda_1$  and  $(1-\alpha)C/\lambda_2$ , where  $\alpha$  is between 0 and 1, and where  $\lambda_1 \geq \lambda_2 > 0$ , and where  $C/\lambda_1 \geq$  the maximum between  $\alpha C/\lambda_1$  and  $(1-\alpha)C/\lambda_2$ .

[0083] Determining the maximum between  $\alpha C/\lambda_1$  and  $(1-\alpha)C/\lambda_2$  shows an objective function for reducing, or minimizing, an overall makespan of an I/O request. This makespan may be determined by the maximum value of the I/O operating time of each node involved in the parallel prefetching (e.g., the primary node and the replica node). Furthermore, by ensuring that  $\lambda_1 \geq \lambda_2 > 0$ , the local I/O speed (e.g., the speed associated with the primary node) is greater than the remote I/O speed (e.g., the speed associated with the replica node). The speeds associated with the respective

nodes may account for, or may include, network delay. Alternatively, a constraint may simply be that  $\lambda_1 > 0$  and that  $\lambda_2 > 0$  (e.g., if the remote I/O speed associated with the replica node is higher than the I/O speed associated with the primary node, although an optimization result may remain the same. Additionally, by having  $C/\lambda_1 \geq a$  maximum of  $\alpha C/\lambda_1$  and  $(1-\alpha)C/\lambda_2$ , parallel prefetching may only be triggered when resulting in a reduction of the makespan.

**[0084]** FIG. 6 is a graphical depiction of mathematical equations for finding an optimized solution of stream division, according to an embodiment of the present invention.

**[0085]** Referring to FIG. 6, the abovementioned functions, constraints, and equations may be graphically represented, where lines 601 and 602 represent an objective function curve, and where line 605 represents  $C/\lambda_1$ . As shown in FIG. 6, a minimum point occurs at a cross point of the line 601 and the line 602. Accordingly, the minimum point may be determined by setting  $\alpha C/\lambda_1$  equal to  $(1-\alpha)C/\lambda_2$ . Thereafter, a minimal makespan may be determined as  $C/(\lambda_1\lambda_2)$  (i.e., line 603) when  $a$  is equal to  $\lambda_1/(\lambda_1+\lambda_2)$  (i.e., line 604).

**[0086]** Accordingly, a parallel prefetching policy may be determined. In some embodiments of the present invention, a parallel fetching daemon may be used to trigger the parallel prefetching. The parallel fetching daemon may periodically check whether the access speed of the SSD of the replica node (including network delay) is approximately equal to the access speed of the local SSD of the primary node by comparing their difference with a preset threshold. The parallel fetching daemon may also check whether a current utilization ratio of throughput of the SSD of the replica node is less than a given operating threshold, and may approve/trigger parallel prefetching if all of these conditions are satisfied.

**[0087]** Furthermore, the parallel fetching daemon may calculate a branching ratio of data to be loaded from each node of a datacenter cluster, and may then assign the branching ratio to each node before beginning to read from the nodes. Accordingly, embodiments of the present invention may be further extended to a parallel fetching case involving more than two source nodes. Lastly, embodiments of the present invention may also work for parallel write operations with additional synchronization schemes.

**[0088]** Accordingly, the solutions provided by embodiments of the invention described above provide for a data replica manager designed for distributed caching, and data processing systems using SSD-HDD tier storage systems. The embodiments improve the ability of fault tolerance by storing caches in replica nodes to effectively recover from disasters while enhancing performance in the SSD space. Furthermore, the embodiments provide at least three approaches for selecting replica nodes and building a replica cluster architecture to support multiple SLAs, based on importance degree, workload temperature, and an abstract distance matrix, which considers network delay and storage access latency.

**[0089]** Embodiments of the present invention can also automatically perform load-balancing among nodes, and can conduct seamlessly online migration operation, as opposed to pausing the subsystem and copying the entire dataset from one node to the other. The described embodiments therefore improve fault tolerance ability for cluster caching systems, and enable successful recovery from four different failure scenarios. Embodiments of the present invention also allow

for parallel prefetching from both primary node and replica node(s) with an optimized prefetching solution for each I/O path.

What is claimed is:

1. A method of data storage, the method comprising:  
determining a latency distance from a primary node to each of two or more replica nodes;  
choosing a preferred replica node of the two or more replica nodes based on the determined latency distances; and  
write-caching data into the preferred replica node.

2. The method of claim 1, wherein the two or more replica nodes each have a smaller latency distance to the primary node than all other available nodes of a datacenter cluster comprising the primary node and the two or more replica nodes.

3. The method of claim 1, wherein each node of a datacenter cluster as a primary node has a ranking of preferred replica nodes that are ranked based on network delays associated with paths between the primary node and respective ones of the replica nodes, and  
wherein each node of the datacenter cluster comprises:

a solid-state drive tier as a cache tier comprising a cache partition for storing data of local virtual machines and a replica partition for storing replica data from other nodes; and  
a hard disk drive tier as a back end storage tier.

4. The method of claim 3, wherein each node comprises a physical host for running multiple virtual machines, each of the virtual machines sharing the solid-state drive tier and the hard disk drive tier in a respective node of the datacenter cluster, the method further comprising periodically updating the rankings of the preferred replica nodes to account for changes in network delays.

5. The method of claim 3, wherein the rankings of the preferred replica nodes are further based on workload characteristics comprising access frequency of data associated with the nodes and service level agreements (SLAs) associated with the nodes.

6. A method for enabling fault tolerance of a datacenter cluster caching system, the method comprising:

detecting a failure of the datacenter cluster; and  
performing load balancing among nodes of the datacenter cluster in accordance with the detected failure by moving data among storage tiers of the nodes,  
wherein the load balancing is performed without pausing a subsystem corresponding to the detected failure, and without copying an entirety of a dataset of a node of the datacenter cluster corresponding to the detected failure.

7. The method of claim 6, the method further comprising: migrating replica data from a first replica node in the datacenter cluster to a second replica node in a datacenter cluster by:

writing new incoming data on the second replica node; and  
maintaining unchanged cached data on the first replica node.

8. The method of claim 7, further comprising:  
evicting victim data from a solid-state drive of a primary node;  
receiving the new incoming data at the primary node;  
writing the new incoming data to a hard disk drive of the primary node; and

removing the victim data from a solid-state drive of the first replica node.

9. The method of claim 6, wherein the detected failure comprises a virtual machine crash on a primary node of the datacenter cluster, the method further comprising:  
 writing dirty data from a solid-state drive of the primary node to a hard disk drive of the primary node;  
 maintaining the dirty data in a solid-state drive of a replica node;  
 flagging the dirty data in the solid-state drive of the replica node as nondirty;  
 restarting the crashed virtual machine on the primary node; and  
 forwarding incoming I/O requests to the solid-state drive of the primary node and to the solid-state drive of the replica node.

10. The method of claim 6, wherein the detected failure comprises a cache device failure on a primary node of the datacenter cluster such that information cannot be written to the primary node, the method further comprising:  
 writing dirty data from a solid-state drive of a replica node of the datacenter cluster to a hard disk drive of the primary node;  
 maintaining the dirty data in the solid-state drive of the replica node;  
 flagging the dirty data as nondirty;  
 broadcasting the information to associated nodes of the datacenter cluster; and  
 writing back dirty data from respective solid-state drives of the associated nodes of the datacenter cluster to respective hard disk drives of the associated nodes.

11. The method of claim 10, further comprising:  
 flagging data in the solid-state drives of the associated nodes with a nondirty flag;  
 replacing the solid-state drive on the primary node;  
 continuing to write incoming I/O requests to the solid-state drive of the primary node and the solid-state drive of the replica node; and  
 writing new replica data from the associated nodes to the solid-state drive of the primary node.

12. The method of claim 6, wherein the detected failure comprises a cache device failure on a first replica node of the datacenter cluster, the method further comprising:  
 disconnecting a primary node of the datacenter cluster;  
 writing back dirty data from a solid-state drive of the primary node to a hard disk drive of the primary node;  
 finding a remote second replica node of the datacenter cluster;  
 storing the dirty data in a solid-state drive of the second replica node;  
 flagging the dirty data in the solid-state drive of the second replica node as nondirty; and

continuing to write incoming I/O requests to the solid-state drive of the primary node and the solid-state drive of the second replica node.

13. The method of claim 12, wherein finding a remote second replica node of the datacenter cluster comprises using a dynamic evaluation process.

14. The method of claim 6, wherein the detected failure comprises a communication failure between a primary node of the datacenter cluster and a first replica node of the datacenter cluster, the method further comprising:  
 writing back dirty data from a solid-state drive of the primary node to a hard disk drive of the primary node;  
 finding a second replica node to replace the first replica node;  
 writing I/O requests to a solid-state drive of the primary node and a solid-state drive of the second replica node; and  
 broadcasting to all nodes of the datacenter cluster a request to release all old replica data corresponding to the first replica node.

15. The method of claim 14, wherein finding the second replica node comprises using a dynamic evaluation process.

16. A method of using replicated data to enable parallel prefetching from multiple nodes of a datacenter cluster, the method comprising:  
 splitting a dataset into two parts; and  
 loading each of the two parts of the dataset from a corresponding node.

17. The method of claim 16, wherein the corresponding nodes for each of the two parts of the dataset comprise:  
 a primary node of the datacenter cluster; and  
 a replica node of the datacenter cluster.

18. The method of claim 17, wherein:  
 C is a total size of the dataset;  
 $\alpha C$  is a size of the one of the two parts from a solid-state drive of the primary node;  
 $\lambda_1$  is an access speed of the solid-state drive of the primary node; and  
 $\lambda_2$  is an access speed of a solid-state drive of the replica node, the method further comprising triggering parallel prefetching when  $C/\lambda_1$  is greater than or equal to a maximum of  $\alpha C/\lambda_1$  and  $(1-\alpha)C/\lambda_2$ .

19. The method of claim 17, wherein  $\lambda_1$  and  $\lambda_2$  each account for corresponding network delays.

20. The method of claim 17, further comprising achieving a network makespan of

$$\frac{C}{\lambda_1 + \lambda_2} \text{ when } \alpha = \frac{\lambda_1}{\lambda_1 + \lambda_2}.$$

\* \* \* \* \*