DOI 10.12694/scpe.v18i4.1330
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 18, Number 4, pp. 291–311. http://www.scpe.org

AUTOMATIC AND SCALABLE DATA REPLICATION MANAGER IN DISTRIBUTED COMPUTATION AND STORAGE INFRASTRUCTURE OF CYBER-PHYSICAL SYSTEMS

ZHENGYU YANG *, JANKI BHIMANI †, JIAYIN WANG ‡, DAVID EVANS §, AND NINGFANG MI ¶

Abstract. Cyber-Physical System (CPS) is a rising technology that utilizes computation and storage resources for sensing, processing, analysis, predicting, understanding of field-data, and then uses communication resources for interaction, intervene, and interface management, and finally provides control for systems so that they can inter-operate, evolve, and run in a stable evidence-based environment. There are two major demands when building the storage infrastructure for a CPS cluster to support above-mentioned functionalities: (1) high I/O and network throughput requirements during runtime, and (2) low latency demand for disaster recovery. To address challenges brought by these demands, in this paper, we propose a complete solution called "AutoReplica" – an automatic and scalable data replication manager in distributed computation and storage infrastructure of cyber-physical systems, using tiering storage with SSD (solid state disk) and HDD (hard disk drive). Specifically, AutoReplica uses SSD to absorb hot data and to maximize I/Os, and its intelligent replication scheme further helps to recovery from disaster. To effectively balance the trade-off between I/O performance and fault tolerance, AutoReplica utilizes the SSDs of remote CPS server nodes (which are connected by high speed fibers) to replicate hot datasets cached in the SSD tier of the local CPS server node. AutoReplica has three approaches to build the replica cluster in order to support multiple SLAs. AutoReplica automatically balances loads among nodes, and can conduct seamlessly online migration operation (i.e., migrate-on-write scheme), instead of pausing the subsystem and copying the entire dataset from one node to the other. Lastly, AutoReplica supports parallel prefetching from both primary node and replica node(s) with a new dynamic optimizing streaming technique to improve I/O performance. We implemented AutoReplica on a real CPS infrastructure, and experimental results show that AutoReplica can significantly reduce the total recovery time with slight overhead compared to the no replication cluster and traditional replication clusters.

Key words: Cyber Physical Systems Infrastructure, Replication, Backup, Fault Tolerance, Device Failure Recovery, Distributed Storage System, Parallel I/O, SLA, Cache and Replacement Policy, Cluster Migration, VM Crash, Consistency, Atomicity

AMS subject classifications. 68M14, 68N30, 68P20

1. Introduction.

With the rise of Cloud Computing and Internet of Things, as an enabling technology, Cyber-Physical Systems (CPS) is increasingly reaching almost everywhere nowadays [1, 2, 3, 4]. CPS uses computing, communication, and control methods to operate intelligent and autonomous systems that can provide using cutting edge technologies. That is to say, CPS is the integration of computation, networking, and physical processes. As illustrated in Fig. 1.1, a typical CPS structure has the following three stages:

- **Data Capture Stage:** consists of relative lightweight "field devices" (also called "CPS clients") such as embedded computers, sensors, network and other mobile CPS devices.
- **Data Management Stage:** is responsible for multi-stream data collection, data storage, preliminary process, sharing control. AutoReplica is working on this stage.
- **Data Process Stage:** analyzes the streaming data, makes decisions, and sends feedbacks to CPS clients.

In modern CPS use cases, huge amount of data are needed to be stored and processed on the CPS cluster, and the corresponding I/O pressure will be mainly put on the "Data Management Stage". In real implementation of a CPS cluster, there are two challenges in this stage: The first challenge is related to the I/O speed requirement in large-scale CPS. Traditional HDDs are not efficient for high speed I/O requires [5, 6]. Therefore, high speed SSDs are often utilized in CPS storage system, as shown in Fig. 1.2 left subfigure, where a CPS server node can have multiple storage devices such as SSDs, performance-oriented HDDs and archive-oriented HDDs as shown

* Dept. of Electrical & Computer Engineering, Northeastern University, 360 Huntington Ave., Boston, MA, USA, 02115 (yangzy1988@coe.neu.edu).
† Dept. of Electrical & Computer Engineering, Northeastern University, 360 Huntington Ave., Boston, MA, USA, 02115 (bhimani@ece.neu.edu).
‡ Dept. of Computer Science, University of Massachusetts Boston, 100 Morrissey Boulevard, Boston, MA, USA 02125 (jiayin.wang001@umb.edu)
§Samsung Semiconductor Inc., Memory Solution Research Lab, Storage Software Group, San Diego, CA,USA 92121 (david.evans@samsung.com)
¶ Dept. of Electrical & Computer Engineering, Northeastern University, 360 Huntington Ave., Boston, MA, USA, 02115 (ningfang@ece.neu.edu).

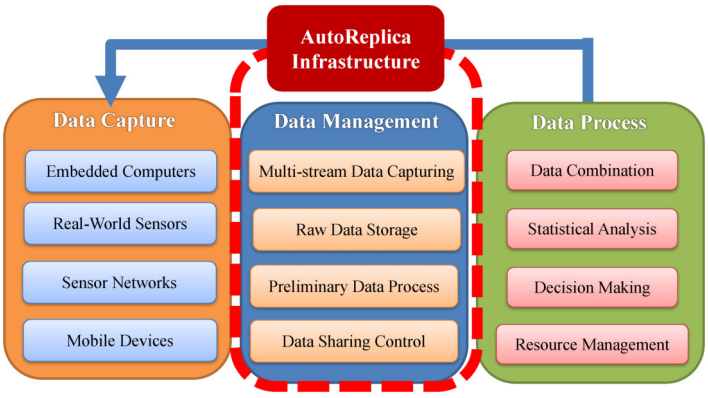

FIG. 1.1. *Three-stage data flow and components of CPS implementation.*

in the dash box. Above that, multiple virtual machines (VMs) running CPS platform applications are hosted by the hypervisor software, and all of them are sharing the storage pools. In Fig. 1.2, the right side figure further illustrates that in order to speed up the I/O performance of the storage system, SSDs are used to cache hot data, and HDDs are designed to host backend cold data.

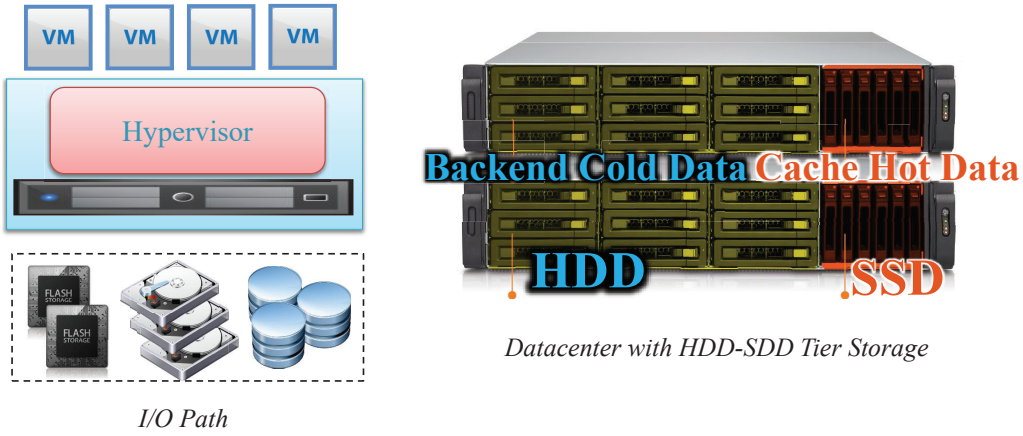

FIG. 1.2. *Storage architecture of each node.*

The second challenge is the problem of data recovery from different types of disasters. Data loss and delay caused by disasters will dramatically reduces the data availability and consistency, which are very critical for CPS applications. To address this challenge, replication technique – a process of synchronizing data across multiple storage nodes – is often used to provides redundancy and increases data availability from the loss of a single storage node [7, 8, 9, 10].

However, since redundancy brings overheads in terms of network traffic, I/O performance, storage space, and consistency maintenance, we need to balance the replication and performance [11, 12]. In piratical, SSDs are often used as the *write back* cache to improve to I/O speed, having only one up-to-date copy on SSD is not acceptable for high SLA (Service-Level Agreement) demand use cases such as bank, stock market, and military CPS networks. According to the study [13], compare to HDD, SSD is relatively not a "safe destination" though it can preserve the data after power off. Therefore, we focus on replicating only cached datasets in the SSDs, and now the main problem is *"where to store replicas of those datasets cached in the SSD while not downgrading the performance?"*

Motivated by this, we propose a complete solution called "AutoReplica", which is an automatic and scalable data replication manager designed for distributed CPS infrastructure with SSD-HDD tiering storage systems. AutoReplica maintains replicas of local SSD cache in the remote SSD(s) connected by high speed fibers, since the access speed of remote SSDs can be faster than that of local HDDs. AutoReplica can automatically build and rebuild the cross-node replica structure following three approaches designed based on different SLAs. AutoReplica can efficiently recover from different disaster scenarios (covers CPS service virtual machine crash, device failures and communication failures) with limited and controllable performance downgrades with a lazy migrate-on-write technique called "fusion cache", which can conduct seamlessly online migration to balance loads among nodes, instead of pausing the subsystem and copying the entire dataset from one node to the other. Finally, AutoReplica supports parallel prefetching from both primary node and replica node(s) with a novel dynamic optimizing streaming technique to further improve I/O performance. We implemented AutoReplica on VMware ESXi platform, and experimental results based on real CPS workloads show that AutoReplica can significantly improve the performance with slight or even less overheads compared to other solutions.

The remainder of this paper is organized as follows. Sect. 2 presents the topological structure of AutoReplica cluster. Sect. 3 introduces AutoReplica's cache and replacement policy, including the new "fusion cache" technique. Sect. 4 describes recovery policies under four different scenarios. Sect. 5 discusses the parallel prefetching scheme. Sect. 7 shows the experimental results. Finally, we summarize the paper in Sect. 8.

2. Topological Structure Of Datacenter. We first introduce topological structure of AutoReplica cluster [14, 15, 16]. As illustrated in Fig. 2.1, there are multiple nodes in the cluster, and each node is a physical host which runs multiple CPS service virtual machines (VMs). In our prototype, we use VMware's ESXi [17] to host VMs. Inside each node, there are two tiers of storage devices: SSD tier and HDD tier. The former tier is used as the cache and the latter tier is used as the backend storage. Each storage tier contains one or more SSDs or HDDs, respectively. RAID mode disks can also be adopted in each tier. SSD and HDD tiers in each node are shared by VMs and managed by the hypervisor.

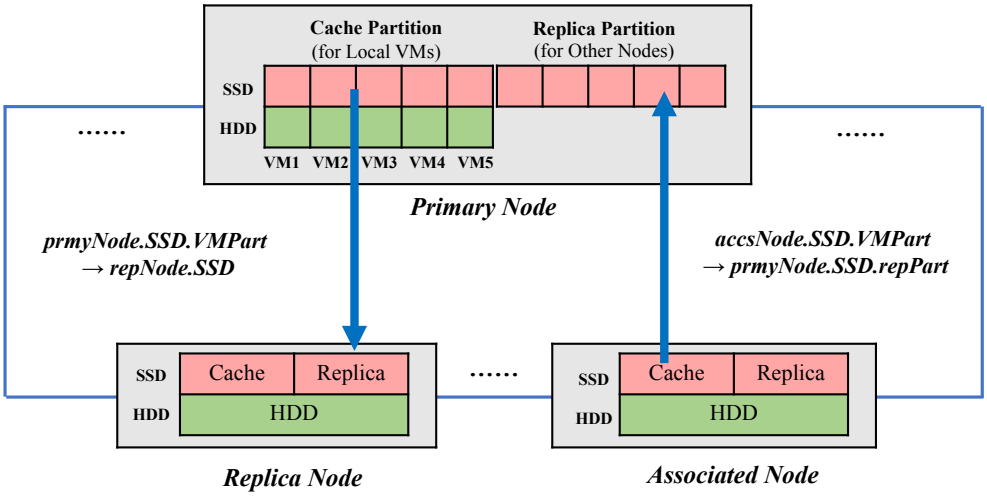

FIG. 2.1. *An example of the structure of AutoReplica's datacenter.*

Since nodes are connected by high speed fiber channels, the remote SSD access speed (including the network delay) can be even faster than local HDD access speed. Thus, to utilize remote SSDs as replica destination, inside the SSD tier, we set two partitions: "Cache Partition" (for the local VMs), and "Replica Partition" (for storing replica datasets from other nodes SSD cache). AutoReplica uses *write back cache policy* to maximize I/O performance, since writing through to HDD will slow down the I/O path. However, as mentioned, SSD is relatively vulnerable and not cannot be equally trusted as a "safe destination" like HDD, though SSD can preserve the data after power off. Therefore, AutoReplica maintains additional replicas in the remote SSDs

to prepare for recoveries for failures. In fact, we still can use local HDD as the second replica device for those extremely high SLA nodes, which will be discussed in Sect. 2.1. Based on these facts, we propose three approaches to setup the topological structure of the datacenter clusters, focusing on *"how to select replica nodes?"*, *"how many replicas nodes do we need?"*, and *"how to assign replicas?"*.

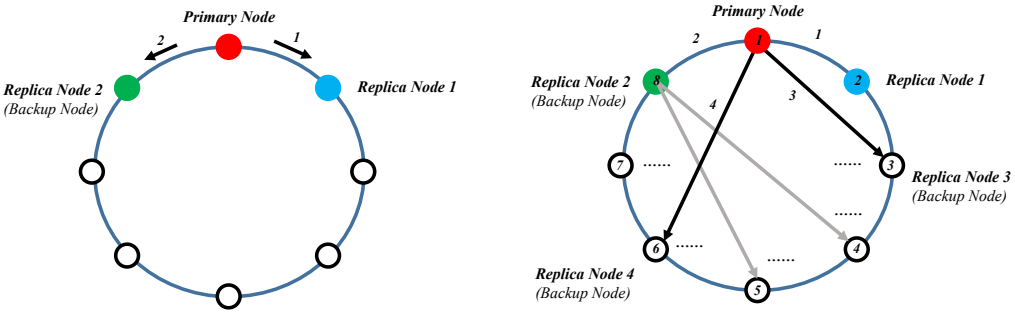

FIG. 2.2. *Examples of (a) Ring and (b) Network approaches.*

2.1. Ring Approach. Our first approach is a directed logical "Ring" structure, which can be either user-defined or system-defined. A system-defined ring is based on geographic distance parameters (e.g., I/O latency and network delay). As shown in Fig. 2.2(a), this logical ring defines an order of preference between the primary and replica nodes. Caching is performed using the local SSD with a copy replicated to another node in the cluster. Each node consists of two neighbors, storing replicas on both/one of them. The node walks in the ring until it can find a replica to use if unsuccessful during the process of building the ring cluster. Once it has a replica, it can begin to write caching independently of what the other nodes are doing.

2.2. Network Approach. As a "linear" approach, the "Ring" structure has a drawback during searching and building replicas, since it has only one or two directions (e.g., previous and next neighbors). In order to improve system robustness and flexibility, we further proposed the "network" approach – a symmetric or asymmetric network, see Fig. 2.2(b), which is based on each node's preference ranking list of all its connected nodes (i.e., not limited to two nodes).

TABLE 2.1
Example of the "distance matrix" used in Network approach, which shows the ranking of each path.

From \ To	1	2	3	4	5	6	7	8
1	-	1	3	-	-	4	-	2
2	...	-	1	...	2	...	...	...
3	3	...	-	...	1	...	2	...
...	...	...	...	...	...	...	...	...
8	...	...	...	2	...	...	1	-

In real implementation, we introduce a "distance matrix" (an example is shown in Table 2.1) to maintain each node's preference list ranked by a customized "score" calculated based on multiple parameters such as network delay, I/O access speed, space/throughput utilization ratio, etc. This matrix is periodically updated through runtime measurement (e.g., heartbeat). For example, in Table 2.1, node 1's first neighbor is node 2, and its second neighbor is node 8, etc.

The main procedure of how to assign the replica nodes for each node is as following: Each node searches the matrix and selects its "closest" node as its replica node if possible. To avoid the "starvation" case where lots of nodes are choosing one single node or a small range of nodes as their replica nodes, AutoReplica also limits the maximum replica number per node. Lastly, each node can also have more than one replica node.

2.3. Multiple-SLA Network Approach. In real environment, rather than treating different nodes equally, the CPS system administrator is often required to differentiate the quality of service based CPS network SLAs and even workload characteristics. To support this requirement, we further develop the "multiple-SLA network" approach to allow each node to have more than one replica node with different configurations based on a replica configuration decision table.

TABLE 2.2
Replica configuration table for multiple SLAs.

Case	Workload		Destination				#Reps.
	SLA	Temp.	SSD_P	SSD_{R1}	SSD_{R2}	HDD_P	
1		✓	✓	✓			1
2			✓		✓		1
3	✓	✓	✓	✓	(✓)		1(2)
4	✓		✓	✓		(✓)	1(2)

An example of replica configuration table is shown in Table 2.2, where SSD_P, SSD_{R1}, SSD_{R2} and HDD_P stand for the SSD tier of the primary node, the SSD tier of the first replica node, the SSD tier of the second replica node, and the HDD tier of the primary node, respectively. It also considers:

- **SLA:** Related with importance of each node. Multiple SLAs is supported by utilizing multiple replica configurations. Although our example has only two degrees: "important" and "not important", AutoReplica supports more fine-grained degrees (even online-varying) SLAs.
- **Temperature:** Similar to [18], we use "data temperature" as an indicator to classify data into two categories according to their access frequency: "hot data" has a frequent access pattern, and "cold data" is occasionally queried.

Local HDD ($prmyNode.HDD$) can also be used as a replica destination (case 4 in table 2.2), and AutoReplica needs reduce the priorities of those write-to-HDD replica operations in order not to affect those SSD-to-HDD write back and HDD-to-SDD fetch operations in the I/O path. Additionally, techniques [19, 20, 21] are adopted to further improve the performance of the write-to-HDD queue in the I/O path considering multiple SLAs.

3. Cache and Replacement Policies. To maximize the I/O performance, AutoReplica uses *write back cache policy*. In detail, when the SSD tier (i.e., cache) is full, SSD-to-HDD eviction operations will be triggered in the (local) primary node ($prmyNode$); while in the replica node ($repNode$), the corresponding dataset will simply be removed from the $repNode.SSD$ without any additional I/O operations to the $repNode.HDD$. Alg. 3.1 shows a two-replica-node implementation. In fact, it can have any number of SSD replica nodes to support more fine-grained SLAs. Specifically, AutoReplica's cache and replacement policy is basically switching between two modes, namely "runtime mode" (line 19) and "online migration mode" (line 3 to 11) by periodically checks the $migTrigger$ condition (which considers runtime states such as load balancing and bandwidth utilization). If $migTrigger$ returns true, AutoReplica will select the "$overheat$" replica node (line 6) and is replaced with the next available replica node (line 7). After that, AutoReplica begins to run under the "migration mode" (line 14). If the "migrate out" replica node ($repNodeOut$) has no more "out-of-date" replica datasets (i.e., the migration is finished), AutoReplica then stops the migration by setting $migModeFlag$ to $false$ (line 16), and goes back to the runtime mode (line 19). We describe the details of the runtime mode and the migration mode cache policy in Sect. 3.1 and 3.2.

3.1. Runtime Mode Cache Policy. Under the runtime mode, AutoReplica searches the new I/O request in the local SSD cache partition (i.e., $prmyNode.SSD$). If it returns a cache hit, then AutoReplica either fetches it from the $prmyNode.SSD$ for a read I/O, or updates the new data to its existing cached copies in $prmyNode.SSD$ and the SSD replica partition in corresponding replica node(s) for a write I/O. For the cache miss case, AutoReplica first selects a victim to evict from the $prmyNode.SSD$ and all the victim's copies from $repNode.SSD$(s), and then AutoReplica only writes those unsync (with "dirty" flag) evicted datasets into $prmyNode.HDD$. AutoReplica then inserts the new dataset into both $prmyNode$ and all its $repNode.SSD$(s). If it is a read I/O, AutoReplica fetches it from $prmyNode.HDD$ to SSDs of the $prmyNode.HDD$ and also

Main Procedure of AutoReplica's Cache Policy

Note: (1) $prmyNode$ is the primary node. $repNodeRem$ and $repNodeOut$ are the original two replica node. $repNodeIn$ is the destination of migration which replaces $replicaNodeOut$.
(2) $write(inputData, inputDirtyFlag)$: A function that writes the *inputData* into the device. If the device is "SSD", then this function sets *dirtyFlag* of the *inputData* as *True*. If the device is "HDD", then $inputDirtyFlag$ can be ignored.
(3) $migrateTrigger()$: A function returns *True* if the subsystem needs to migrate due to imbalance load.
(4) T_W: window size (i.e., frequency) of migration condition checking.

```
Procedure cache (prmyNode, repNode1, repNode2)
1   migModeFlag = Flase
2   for each new I/O request newData ∈ IOStream on prmyNode do
3       /* check load balance */
4       if currTime mod T_W == 0 and migModeFlag ≠ True and migTrigger() ≠ False then
5           migModeFlag = True
6           repNodeOut = selectOverheatNode(repNode1, repNode2)
7           repNodeIn = selectNextReplica()
8           if repNodeIn == repNode1 then
9               repNodeRem = repNode2
10          else
11              repNodeRem = repNode1
12      /* online migrate mode cache policy */
13      if migModeFlag == True then
14          cacheOnlineMigrateMode(newData, prmyNode, repNodeRem, repNodeOut, repNodeIn)
15          if repNodeOut.SSD.sizeOfRepForPrmy() == 0 then
16              migModeFlag = False
17      /* runtime mode cache policy */
18      else
19          cacheRuntimeMode(newData, prmyNode, repNode1, repNode2)
20  return
```

FIG. 3.1. *Main procedure of AutoReplica's cache policy.*

send it to all its $repNode(s)$. It finally returns the fetched $cacheData$ to the user buffer in the memory. If it is a write I/O, AutoReplica simply writes it to SSDs of $prmyNode$ and all its $repNode$(s) with $dirtyFlag$ as "dirty", since it is unsync new data. An example is shown in Fig. 3.2, where for the write I/O data "F", AutoReplica first writes back a victim "E" from $prmyNode.SSD$ to $prmyNode.HDD$, and deletes "E" from both $prmyNode.SSD$ and $reNode.SSD$. Lastly, it writes "F" to both $prmyNode.SSD$ and $reNode.SSD$. Our implementation is also compatible for other replacement algorithms to implement the victim selection function, such as Glb-VFRM [18] and GREM [22].

3.2. Online Migration Mode Cache Policy. AutoReplica uses a cost-efficient migrate-on-write scheme called "fusion cache" to migrate replicas from one $repNode$ to the other. The main idea is that instead of pausing the subsystem and copying all existing replicas from the old node ($repNodeOut$) to the new node ($repNodeIn$), regardless of whether these data pieces are necessary or not, "fusion cache" keeps the subsystem alive and only writes new incoming datasets to $repNodeIn$ and keeps those "unchanged" cached data on $repNodeOut$. Eventually, $repNodeIn$ will replace $repNodeOut$. In other words, AutoReplica mirrors the $prmyNode.SSD$ by using the *unibody* of $repNodeOut$ and $repNodeIn$ to save lots of bandwidth.

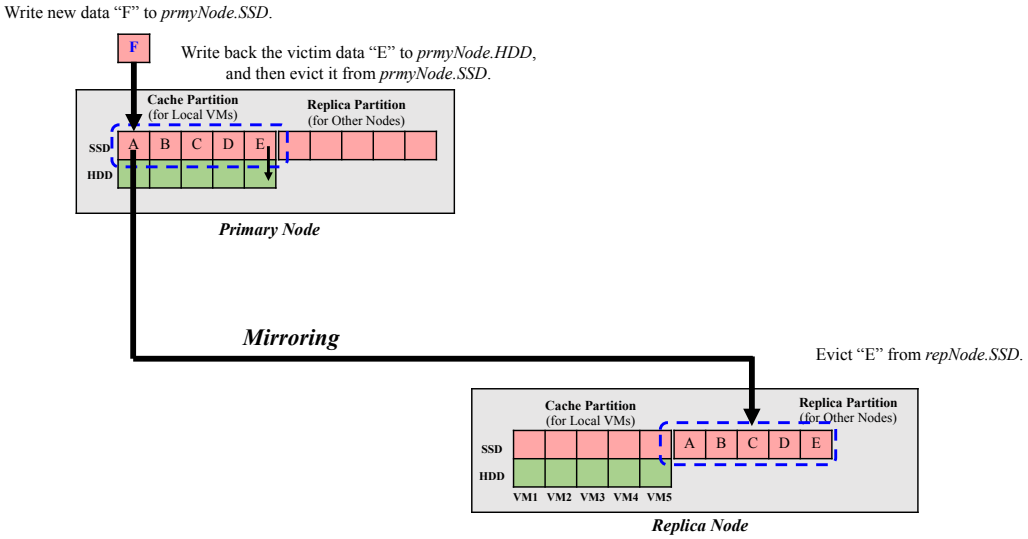

FIG. 3.2. *Example of runtime cache policy.*

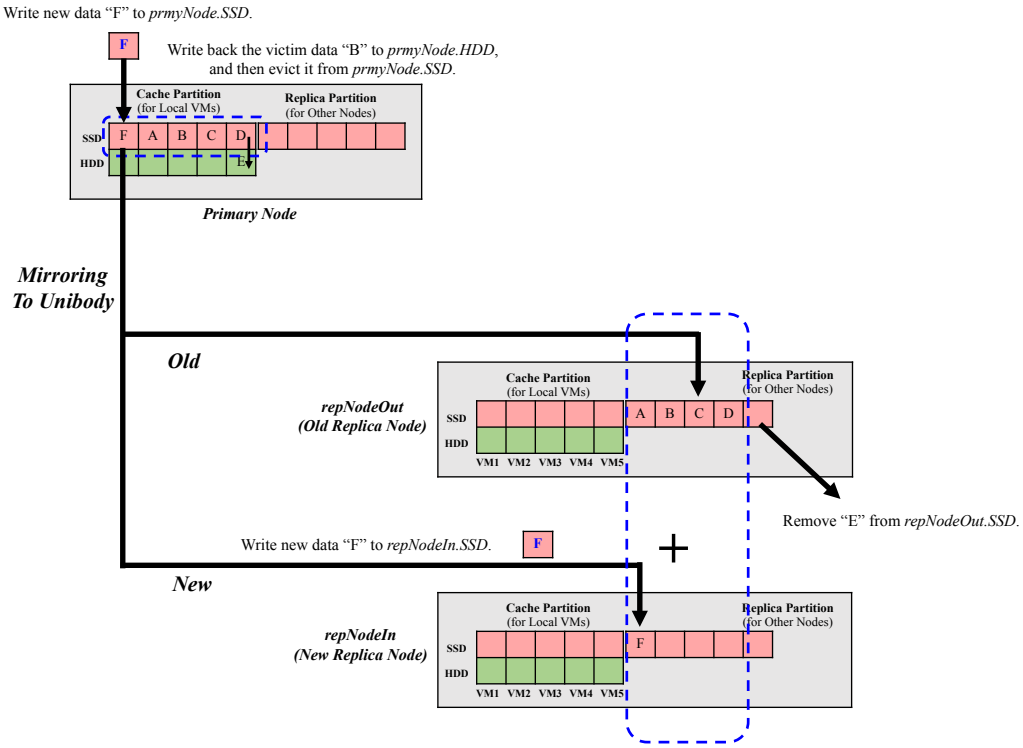

FIG. 3.3. *Example of online migration cache policy.*

An example is depicted in Fig. 3.3, where only one replica node is needed to be "migrated out" and one new replica node is needed to take over those cached datasets. When a new write I/O data "F" comes to the *prmyNode*, similarly, AutoReplica first writes back the victim "E" from $prmyNode.SSD$ to $prmyNode.HDD$ since the cache is full. Then, "E" on the old replica node is directly deleted. After that, the new write I/O "F" will be inserted to the $prmyNode.SSD$ and its new $repNodeIn.SSD$. As mentioned, the *prmyNode* (e.g., the

blue dash box on top of Fig. 3.3) is mirrored in the "fusion cache" across the old and new *repNodes* (e.g., the blue dash box in on the right side of Fig. 3.3). Notice that if "F" is a read I/O, AutoReplica will not migrate it from the old to the new replica node.

To sum up, unlike the traditional pro-active migration, AutoReplica is following this "migrate on write" scheme which is lazy and cost-efficient.

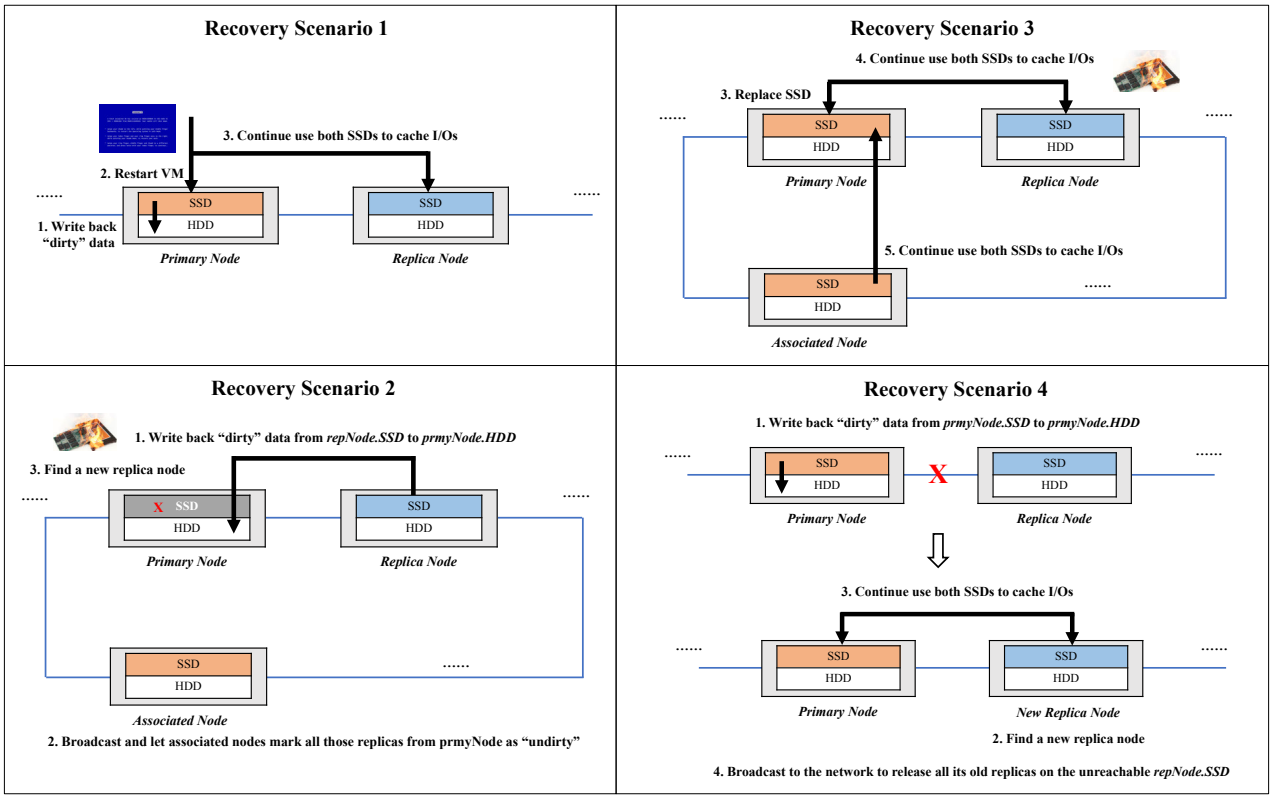

FIG. 4.1. *Example of AutoReplicas recovery scenario.*

4. Recovery Policy. AutoReplica maintains additional replicas in the remote SSDs to prepare for recoveries for different failures. Specifically, it has different procedures to recover from failures covering the following four scenarios:

4.1. VM Crash on Primary Node. A very common failure is that a CPS service virtual machine crashes on the primary node. As shown Fig. 4.1(1) and Fig. 4.2, AutoReplica first closes out the VMDK. It then writes back "dirty" datasets from $prmyNode.SSD$ to $prmyNode.HDD$, and marks all $prmyNode$'s replicas in $repNode.SSD(s)$ with "nondirty" flag. After that, it restarts crashed VM on $prmyNode$, and continues to forward incoming I/O requests on both $prmyNode.SSD$ and $repNode.SSD$.

4.2. Primary Node Cache Device Failure. A primary node cache storage device failure will result in its inability to continue to write caching. As shown in Fig. 4.1(2) and Fig. 4.3, AutoReplica first writes back "dirty" datasets from $repNode.SSD$ to $prmyNode.HDD$, and marks all $prmyNode.SSD$'s replicas on $repNode.SSD(s)$ with "nondirty" flag. It then broadcasts this "unavailable" information to notify those nodes having replicas of this failure $prmyNode$ (called "associated nodes") to write back "dirty" datasets from their own SSD to HDD. Notice that replicated datasets with "nondirty" flags are still kept in the $repNode.SSD$(s). AutoReplica further finds and replaces the SSD on $prmyNode$. After that, it continues to write incoming I/O requests on both $prmyNode.SSD$ and $repNode.SSD$. It also continues to let those "associated nodes" to write new replicas to $prmyNode.SSD$.

[Recovery Scenario 1] VM Crash on Primary Node	
	Procedure $recoverFromPrmyNodeVMCrash()$
1	Write back "dirty" data from $prmyNode.SSD$ to $prmyNode.HDD$, and mark their copies in $repNode.SSD$ with "nondirty" flag.
2	Restart crashed VM on $prmyNode$.
3	Continue to forward incoming I/O requests on both $prmyNode.SSD$ and $repNode.SSD$.
4	**return**

FIG. 4.2. *Main procedure of recovery scenario 1: VM crash on primary node.*

[Recovery Scenario 2] Primary Node Cache Device Failure	
	Procedure $recoverFromPrmyNodeSSDFailure()$
1	Write back "dirty" data from $repNode.SSD$ to $prmyNode.HDD$, and mark their copies in $repNode.SSD$ with "nondirty" flag.
2	Broadcast this "unavailable" information to the network to let those nodes having replicas in this failure $prmyNode$ ("associated nodes") to write back "dirty" data from their own SSD to HDD, and keep copies with "nondirty" flag in those associated nodes.
3	Replace SSD on $prmyNode$.
4	Continue to write incoming I/O requests on both $prmyNode.SSD$ and $repNode.SSD$.
5	Continue to let those "associated nodes" to write new replicas to $prmyNode.SSD$.
6	**return**

FIG. 4.3. *Main procedure of recovery scenario 2: Primary node cache device failure.*

4.3. Replica Node Cache Device Failure. When a replica node detects a cache device (i.e., SSD) failure, it will disconnect from the primary node and reject any future connection attempts from that node with an error response. As shown in Fig. 4.1(3) and Fig. 4.4, AutoReplica then writes back "dirty" dataset from $prmyNode.SSD$ to $prmyNode.HDD$, but still marks and keeps them in $prmyNode.SSD$ with "nondirty" flag. After a new replica node is found by using dynamic evaluation process, AutoReplica continues to write incoming I/O requests on both $prmyNode.SSD$ and new $repNode.SSD$. Notice that policy in Sect. 4.2 takes responsibility for recovering this failure device.

[Recovery Scenario 3] Replica Node Cache Device Failure	
	Procedure $recoverFromRepSSDFailure()$
1	Write back "dirty" data from $prmyNode.SSD$ to $prmyNode.HDD$, and mark their copies in $prmyNode.SSD$ with "nondirty" flag.
2	Find a new replica node by using dynamic evaluation process.
3	Continue to write incoming I/O requests on both $prmyNode.SSD$ and new $repNode.SSD$.
4	**return**

FIG. 4.4. *Main procedure of recovery scenario 3: Replica node cache device failure.*

4.4. Communication Failure Between Primary & Replica Node. When the primary node detects a non-recoverable communication failure between the primary and replica hosts, AutoReplica will be unable to continue to write caching. To recover from this failure, as shown in Fig. 4.1(4) and Fig. 4.5, AutoReplica daemon will write back "dirty" data from $prmyNode.SSD$ to $prmyNode.HDD$ to ensure all cached data are

updated to the backend HDD. Next, it will start the dynamic evaluation process to find a new replica node to replace the unreachable replica node. It will then continue to use both SSDs to cache I/Os following the "fusion cache" design in migration policy. Finally, it will broadcast to the network to release all its old replicas on the unreachable $repNode.SSD$(s).

[Recovery Scenario 4] Communication Failure between Primary and Replica Node	
	Procedure *recoverFromCommFailure*()
1	Write back "dirty" data from *prmyNode.SSD* to *prmyNode.HDD*, and mark their copies in *prmyNode.SSD* with "nondirty" flag.
2	Find a new replica node by using dynamic evaluation process.
3	Continue to write incoming I/O requests on both primary SSD and new replica SSD.
4	Broadcast to the network to release all its old replicas on the unreachable *repNode.SSD*.
5	**return**

FIG. 4.5. *Main procedure of recovery scenario 4: Communication failure between primary and replica node.*

5. Parallel Prefetching. Lastly, replicates can also be used to enable parallel prefetching from multiple nodes (similar like parallel stripping in RAID [23, 24]), especially for read operations. An example is shown in Fig. 5.1, where we split the dataset (with size of C) to prefetch (e.g., a file) into two parts (with sizes of αC and C), and load each part from the primary and replica node. Assume the access speed of $prmyNode.SSD$ is λ_1 (GB/Sec) and the access speed of $repNode.SSD$ (including the network delay) is λ_2 (GB/Sec). Since the main target for parallel prefetching is to reduce the total I/O time, i.e., makespan of each I/O request, we can convert our problem into the following optimization framework:

Optimize:

$$\max\left(\frac{\alpha C}{\lambda_1}, \frac{(1-\alpha)C}{\lambda_2}\right) \tag{5.1}$$

Subject to:

$$\alpha \in [0,1] \tag{5.2}$$

$$\lambda_1 \geq \lambda_2 > 0 \tag{5.3}$$

$$\frac{C}{\lambda_1} \geq \max\left(\frac{\alpha C}{\lambda_1}, \frac{(1-\alpha)C}{\lambda_2}\right) \tag{5.4}$$

Eq. 5.1 is the objective function which minimizes the overall makespan of an I/O request. The makespan is determined by the maximum value of the I/O operating time of each side (i.e., $prmyNode.SSD$ and $repNode.SSD$). Eq. 5.2 ensures that the branching ratio of two streams should be meaningful. Eq. 5.3 reflects that the local SSD I/O speed (i.e., from $prmyNode.SSD$) is usually greater than remote (i.e., $repNode$) I/O speed including network delay. Notice that this constraint can be relaxed as "$\lambda_1 > 0$ and $\lambda_2 > 0$", if the remote I/O speed is higher (which is true in some rare cases), but the optimization framework remains the same. Eq. 5.4 further ensures that the parallel prefetching operation should only be triggered when it *can* help to reduce the I/O makespan. Based on this result, Fig. 5.3 further shows the example of the decision maker workflow for parallel prefetch, where the "StatusMonitor" reports to "PararllelPrefectDeamon" and the latter component switches between the "ParallelPrefetchMode" and "LocalPrefetchMode".

We then plot these functions and constraints into Fig. 5.2, where the red line is the objective function curve, and the blue line is the constraint of Eq. 5.4. We can see that there exists a minimum point at the cross point of $f(\alpha) = \frac{(1-\alpha)C}{\lambda_2}$ and $g(\alpha) = \frac{\alpha C}{\lambda_1}$. In order to calculate this sweet spot, we let:

$$\frac{\alpha C}{\lambda_1} = \frac{(1-\alpha)C}{\lambda_2} \tag{5.5}$$

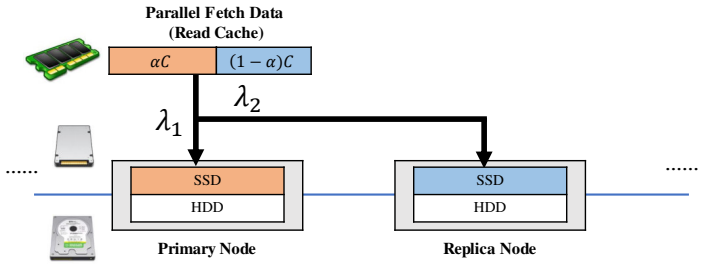

FIG. 5.1. *Example of parallel prefetch enabled read cache.*

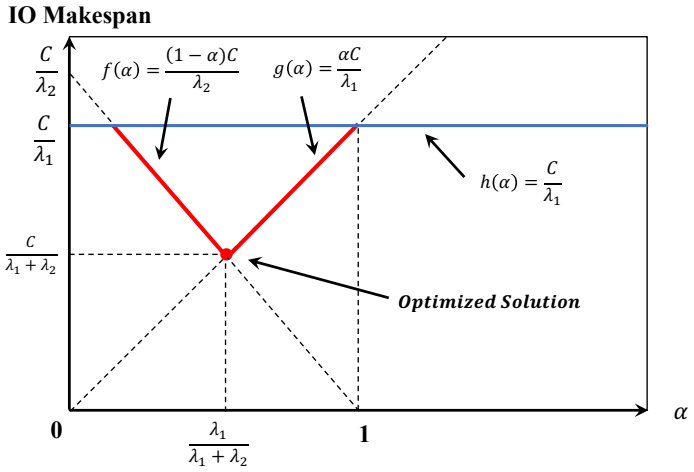

FIG. 5.2. *Finding the optimized solution of prefetching stream division.*

Then, we can get the minimum makespan ($\frac{\alpha C}{\lambda_1+\lambda_2}$) when:

$$\alpha = \frac{\lambda_1}{\lambda_1 + \lambda_2} \tag{5.6}$$

We can easily extend it to support any number of replica nodes. Alg. 5.4 first describes the main procedure of parallel fetching daemon, which triggers the parallel prefetching by periodically checking whether the access speed of its all $repNode.SSD(s)$ (including the network delay) is close enough to the access speed of local $prmyNode.SSD$ (by comparing their difference with a preset threshold ε), and the current utilization ratio of throughput of the $repNode.SSD(s)$ is less than a threshold Thr. Notice that the time window T_W does not necessarily to be same as the sliding window previously mentioned in Alg. 3.1. The "ParallelPrefetchMode" will be triggered if all these conditions are satisfied, and the parallel fetching policy then calculates and assigns the branching ratio of dataset to be loaded from each node. Otherwise, AutoReplica will keep running "LocalPrefetchMode", which only fetches data from the local $prmyNode.SSD$.

6. Evaluation. In this section, we investigate the performance of our proposed AutoReplica solution under different CPS use cases.

6.1. Implementation Configurations. Table 6.1 summarizes the configuration of our CPS server testbed. Figure 6.1 also illustrates the architecture of our VMware-based implementation. Specifically, there are multiple CPS server nodes in the cluster. Each of these CPS server nodes runs the VMware ESXi hypervisor [17] to host multiple VMs that are working for nearby CPS devices based on location distribution. AutoReplica works on the VMware's ESXi in the "user mode". More accurately, with some internal customization, AutoReplica is

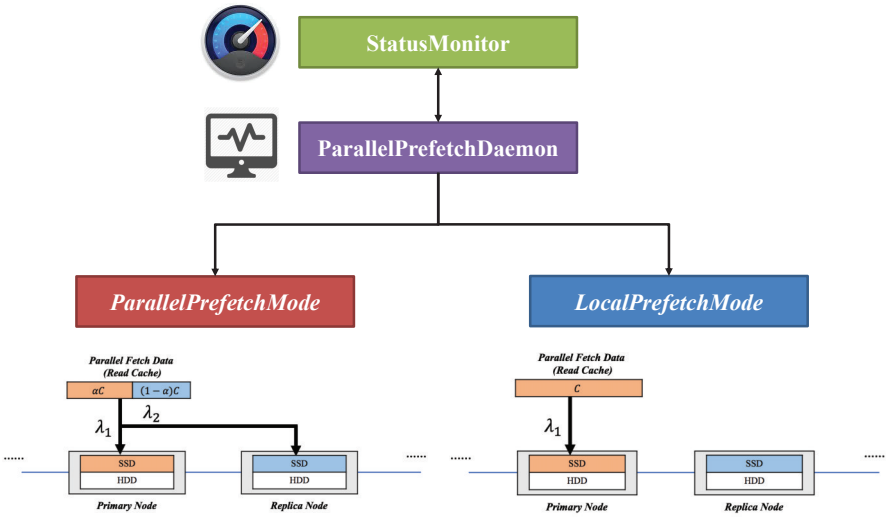

FIG. 5.3. *Decision maker for parallel prefetch.*

very close to a "pseudo-kernel mode" application. AutoReplica consists of 4 components: a vSphere web client plug-in, a CIM provider, multiple I/O filter library instances and a daemon process on each VM. CPS server nodes are connected by high-speed fiber channels, and due to the advantage of SSD and fiber channels, remote SSDs access speed can be faster than local HDDs speed. AutoReplica strives to cache hot datasets onto the SSD tier and bypass those cold datasets onto the HDD tier.

TABLE 6.1
Configuration of a single CPS server.

Component	Specs
Server	PowerEdge R630 Server
Processor	Intel(R) Xeon(R) CPU E5-2660 v4
Processor Speed	2.00GHz
Processor Cores	56 Cores
Processor Cache Size	35M
Memory Capacity	64GB RDIMM
Memory Data Rate	2400 MT/s
Operating System	Ubuntu 14.04 LTS
Docker Version	17.03
VM Hypervisor	VMware Workstation 12.5

6.2. Performance Metrics. In our evaluation, we mainly focus on the following four metrics:

- **Total recovery time:** the cumulative recovery time of all CPS server nodes.
- **Coefficient variation (C.V.) of total recovery time:** balance degree of the cumulative recovery time across all CPS server nodes. The less C.V. is, the better balance is achieved.
- **Overhead:** Total extra I/O and network traffic for recovery.
- **Coefficient variation (C.V.) of overhead:** balance degree of the overhead across all CPS server nodes. The less C.V. is, the better balance is achieved.

We use the open source measurement tools (e.g., `dstat` [25], `iostat` [26], `blktrace` [27]) to measure performance metrics such as total recovery time, disk I/O rate, and network traffic. We evaluate the recovery correctness and efficiency of different failure scenarios that are manually triggered and drawn from well-tuned temporal interval distributions. We also evaluate the total recovery time and extra I/O overhead under different cluster sizes. Different SLA configurations are further considered to evaluate the benefit and overhead of multiple

Main Procedure of Parallel Fetching Daemon

Note: (1) $currUtil_{IOPS}$: current throughput utilization rate of a node.
(2) ϵ: preset threshold to compare the difference between local and remote access speed.
(3) Thr_{IOPS}: preset threshold of upper bound of throughput utilization rate of a node to be perfected from.

Procedure $parallelFetchDaemon()$
1 $parallelReadFlag = False$
2 **while** $True$ **do**
3 **if** $(currTime \bmod T_W == 0)$ **and** $(\lambda(prmyNode.SSD) - \sum_{i \in RepNodes} \lambda(repNode(i).SSD) \leq \epsilon)$ **and** $(repNode(s).currUtil_{IOPS} \leq Thr_{IOPS})$ **then**
4 $parallelReadFlag = True$
5 **else**
6 $parallelReadFlag = False$
7 **return**

Parallel Fetching Policy

Note: (1) $\lambda(prmyNode.SSD)$: the IO speed of $prmyNode.SSD$.
(2) $\lambda(repNode(i).SSD)$: the end-to-end IO speed of $repNode(i).SSD$ (including network delay).

Procedure $parallelFetch(data)$
1 **if** $parallelReadFlag == True$ **then**
2 fetch size of $\left[\frac{\lambda(prmyNode.SSD)}{\lambda(prmyNode.SSD)+\sum_{i \in RepNodes} \lambda(repNode(i).SSD)} \cdot |data|\right]$ data from $prmyNode.SSD$
3 fetch size of $\left[\frac{\lambda(prmyNode(i).SSD)}{\lambda(prmyNode.SSD)+\sum_{i \in RepNodes} \lambda(repNode(i).SSD)} \cdot |data|\right]$ data from $repNode(i).SSD$
4 **return**

FIG. 5.4. *Parallel prefetching procedure.*

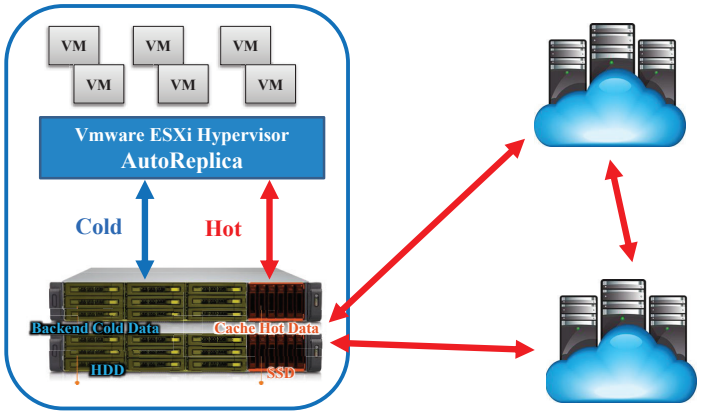

FIG. 6.1. *Architecture of AutoReplica implementation.*

replicas.

6.3. Comparison Solutions. To conduct fair comparison, we implemented the following three representative solutions for the CPS cluster:

- `NoReplication`: No replications will be generated and maintained in the CPS cluster. Once a CPS server node is failed, data cached in the SSD cannot be recovered from other nodes or local HDD tier. As a result, re-collection and re-computation from CPS devices are triggered.

- `Replication(IM)`: A pro-active replication solution, which conducts "Immediate Migration (IM)" on recovery. To conduct fair comparison, we maintain the same replication assignment as AutoReplica [28] for this solution.
- `AutoReplica(MOW&PF)`: Our proposed AutoReplica solution which has fusion cache with migrate-on-write (MOW), and parallel prefetching (PF) features enabled.

Meanwhile, to consider more scenarios in reality, we also evaluate both homogeneous and heterogeneous CPS VM servers and hardware. Details will be discussed in the following two subsections.

6.4. Study on Homogeneous CPS Clusters. We first investigate the homogeneous CPS cluster use case, which is often referred to as a "symmetrical" CPS structure where same CPS devices are widely deployed in the cluster. This use case is popular due to its simplicity and system consistency [29].

Table 6.2 shows the statistics of the experiment configurations. In detail, $clusterSize$ is iterated from 10 to 100 CPS server nodes. We use $avgRepNode\#$ and $avgUsrNode\#$ to demote the average number of replication nodes that each node has and the average number of replicated remote node copies that each node is hosting, respectively. Additionally, $connectRatio$ shows that the connectivity of the cluster, which equals the ratio of the number of paths between two nodes to the upper bound of all possible paths (e.g., for N-node cluster, at most, we can have $\frac{N^2}{2}$ paths). The larger $connectRatio$ is, the higher chance a node will be recovered from others. We further use $avgSSDBW$ and $avgHDDBW$ to denote the average bandwidth of each node at the SSD tier and the HDD tier, respectively. Finally, $readIO\%$ gives the read I/O ratio of each CPS server node. Notice that for the homogeneous use case, both hardware-related (e.g., $avgSSDBW$ and $avgHDDBW$) and workload-related (e.g., $readIO\%$) factors are the same among different cluster size cases.

Table 6.2
Statistics of configurations of homogeneous CPS cluster.

clusterSize	10	20	30	40	50	60	70	80	90	100
avgRepNode#	2.20	2.45	2.63	2.33	2.62	2.45	2.41	2.39	2.43	2.45
avgUsrNode#	1.70	2.00	2.03	1.95	2.04	1.98	2.00	1.93	1.91	1.91
connectRatio(%)	48.00	37.00	48.67	47.13	47.60	48.22	51.06	50.13	49.04	49.96
avgSSDBW(TB/hour)	36.00	36.00	36.00	36.00	36.00	36.00	36.00	36.00	36.00	36.00
avgHDDBW(TB/hour)	0.58	0.58	0.58	0.58	0.58	0.58	0.58	0.58	0.58	0.58
readIO(%)	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00

Figs. 6.2 to 6.5 depict the results of 10 homogeneous CPS cluster use cases, where both device failure and communication failure rates are following the uniform distribution. As shown in Fig. 6.2, `NoReplication` has long total recovery time, because that once a device failure occurs, `NoReplication` has no backups and has to request CPS devices to collect and send data to it again, and it further needs to re-compute the lost data. Meanwhile, `AutoReplica` saves 9.28% of total recovery time from the `Replication` case. The reason is that `AutoReplica` lazily recoveries from its neighbors using the migration-on-write technique and maximizes I/O bandwidth using the parallel prefetching technique. In contrast, `Replication` has to pause and migrate everything immediately once a device or communication failure happens.

We further evaluate the coefficient variation of total recovery time in Fig. 6.3, where the recovery time of `Replication` has a higher chance to be unbalanced among nodes, while our `AutoReplica` has similar balancing degree to `NoReplication`. This result is promising since in a homogeneous use case, more balanced total recovery distribution helps to make the cluster more robust and stable.

We next investigate the extra I/O and network traffic for recovery. As shown in Fig. 6.4, `Replication` has very large overhead, because it has to migrate everything immediately once a failure happens. Meanwhile, even with the need for maintaining additional replications, `AutoReplica` still has lower overhead than all `NoReplication` cases. The main reason behind it is that once a device failure occurs, `NoReplication` has no backups and has to request CPS devices to send data to it again, and it also has to recompute the lost data. These operations trigger a huge amount of extra I/O and network bandwidth consumption.

We further check the coefficient variation of overhead in Fig. 6.5. Neither `Replication` nor `AutoReplica` are load balanced as the `NoReplication` case. Additionally, we observe that although `AutoReplica` saves lots

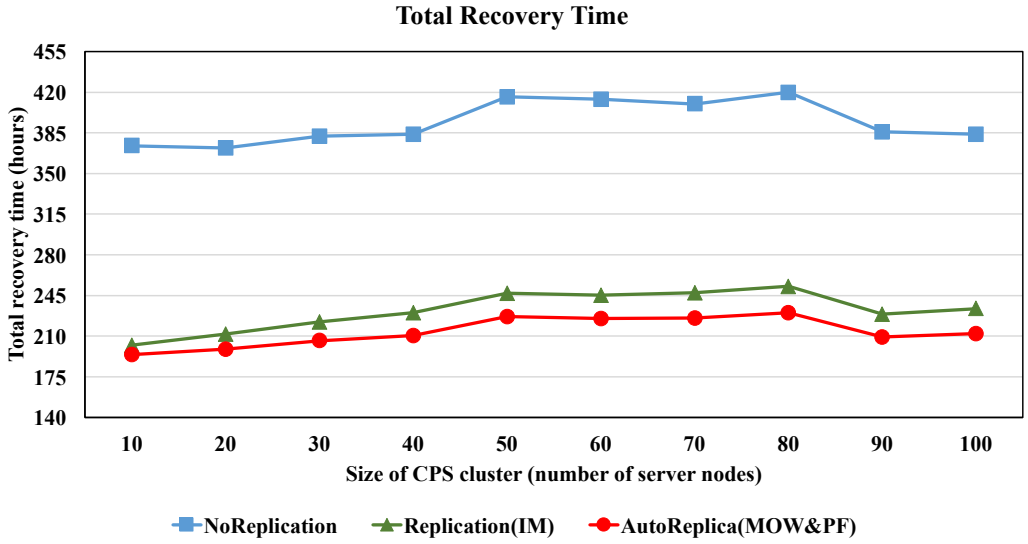

FIG. 6.2. *Total recovery time under different homogeneous CPS cluster sizes.*

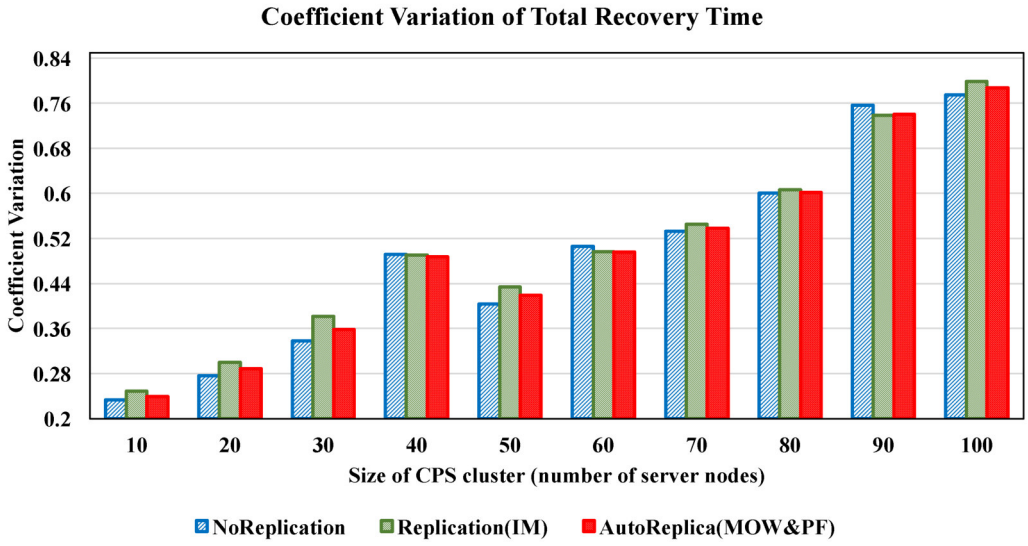

FIG. 6.3. *Coefficient variation of total recovery time under different homogeneous CPS cluster sizes.*

of I/O and network bandwidth, the difference of the overhead balancing degree between `AutoReplica` and `Replication` cases is very slight. The reason is mainly that we deployed the same replication assignment and failure distribution in these two cases in order to conduct fair comparison.

6.5. Study on Heterogeneous CPS Clusters. We further conduct a set of heterogeneous CPS experiments. Table 6.3 shows the configuration of heterogeneous CPS clusters. Hardware factors (such as $avgSSDBW$ and $avgHDDBW$) and workload factors (such as $readIO\%$) are varying among CPS server nodes. Notice that in terms of I/O pattern, CPS workloads usually are write-intensive [30, 31, 32, 33].

Similarly, as we can see from Fig. 6.6, `NoReplication` has the highest total recovery time due to its recomputation cost, and `AutoReplica` still performs better than `Replicaiton` for all cluster sizes. Notice that for cluster size of 20 and 60, we found the failure interval is relatively larger than that for other cluster sizes. As a result, the scenarios with cluster size of 20 and 60 have lower chances to have multiple failures happen

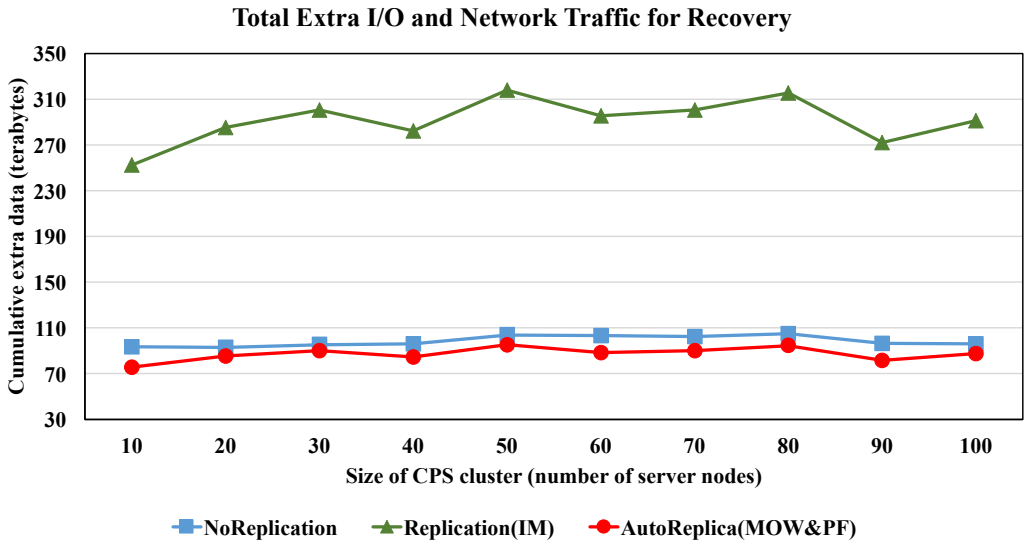

FIG. 6.4. *Total extra I/O and network traffic for recovery under different homogeneous CPS cluster sizes.*

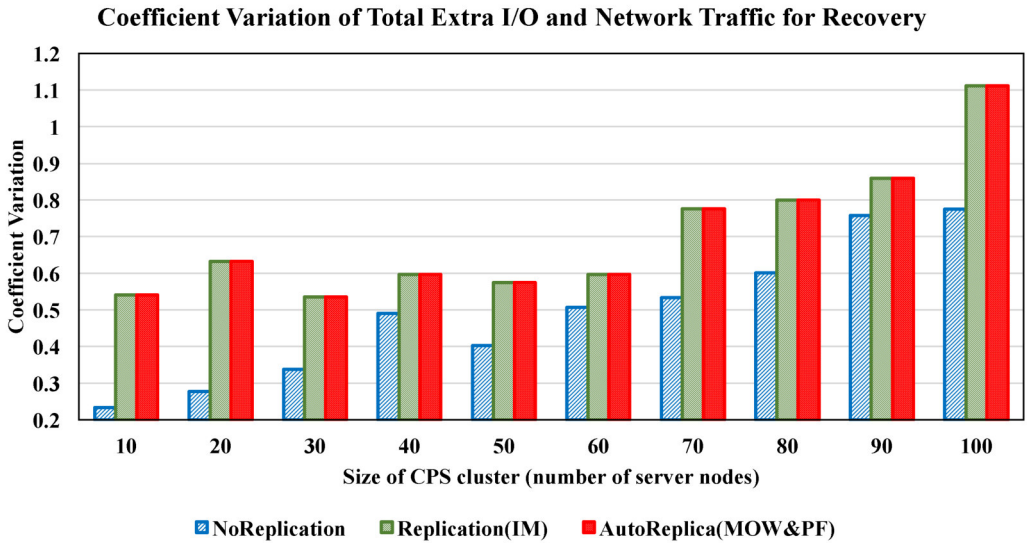

FIG. 6.5. *Coefficient variation of total extra I/O and network traffic for recovery under different homogeneous CPS cluster sizes.*

simultaneously, which then reduces the recovery congestion.

Fig. 6.7 reflects the total recovery time balancing degree. Unlike the homogeneous use case, `Replicaiton` and `AutoReplica` do not have the total recovery time as balanced as `NoReplication` under the heterogeneous use case. The reason is that `NoReplicaiton`'s recovery time is mainly dominated by the CPS device re-collecting and re-sending data. Hence, `NoReplicaiton`'s recovery time is highly coupled with the failure distribution. On the other hand, the recovery time of `Replicaiton` and `AutoReplica` is depending on both failure distribution and replication network assignment.

Similar to the homogeneous use case (e.g., Fig. 6.4), Fig. 6.8 shows that in the heterogeneous use case, `Replication` has the highest overhead. While, `AutoReplica` achieves the shortest total recovery time with a slightly higher overhead compared to `NoReplication`. Fig. 6.9 also show the results of overhead balancing as

TABLE 6.3
Statistics of configurations of heterogeneous CPS cluster.

clusterSize	10	20	30	40	50	60	70	80	90	100
avgRepNode#	2.40	2.45	2.77	2.58	2.60	2.45	2.39	2.38	2.44	2.58
avgUsrNode#	1.80	1.95	2.17	2.05	2.10	1.95	1.83	1.95	1.84	2.08
connectRatio(%)	52.00	46.00	49.56	48.88	49.04	48.00	47.47	49.59	48.62	49.66
avgSSDBW(TB/hour)	37.60	41.15	39.27	39.13	39.34	40.72	41.04	40.73	40.68	40.71
avgHDDBW(TB/hour)	0.50	0.50	0.51	0.51	0.50	0.51	0.50	0.50	0.49	0.50
readIO(%)	35.60	36.70	37.57	37.43	38.08	36.92	38.03	36.91	37.80	37.34

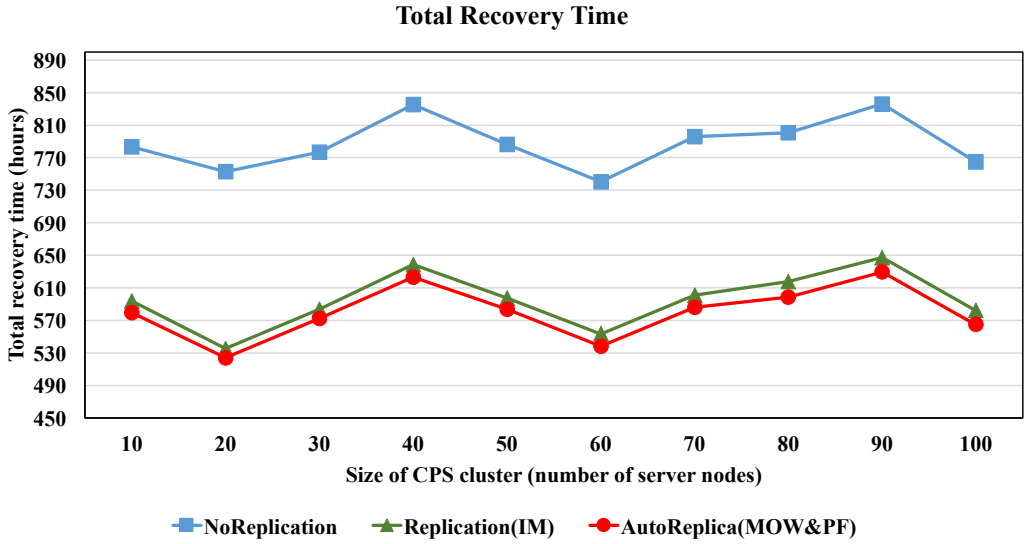

FIG. 6.6. *Total recovery time under different heterogeneous CPS cluster sizes.*

similar as the homogeneous case.

7. Related Work. Replications are widely used in the big data and cloud computing era [4, 34, 35, 36]. Replica's are useful in an incident of data loss for recovery [37, 7] and also for improving performance [38, 33]. Data loss can be incurred by many factors such as software-hardware failure, natural disaster at data center location, power surge etc. Moreover, when there are more highly-visited file gathered in some nodes with poor storage capacity, it will cause a hot issue which may reduce the overall performance of the system, so replication is used to guarantee performance in such critical situations.

Facebook's proprietary HDFS implementation [39] constrains the placement of replicas to smaller groups in order to protect against concurrent failures. MongoDB [28] is a NoSQL database system that uses replicas to protect data. Its recovery scheme is based on the election among live nodes. Copyset [40] is a general-purpose replication technique that reduces the frequency of data loss. [41] designed a novel distributed layered cache system built on the top of the Hadoop Distributed File System. Studies [42, 43, 44, 45, 46, 47, 48] investigated SSD and NVMe storage-related resource management problems, in order to reduce the total cost of ownership and increase the Flash device utilization to improve the overall I/O performance.

Replication creation strategies of based on the frequency of data operation [15, 49, 16] and file heat [50], is proposed to solve the problem of uneven distribution of data in auto-sharing and hybrid clouds. [51] focuses on the dynamic replica placement and selection strategies in the data grid environment. [52] proposed a replication strategy based on the access pattern of file in order to optimize load balancing for large-scale user access in cloud-based WebGISs. [53] highlighted the challenges involved in making a replica selection scheme explicitly cope with performance fluctuations in the system and environment. Replication can also help in reducing communication overhead among different nodes in cloud [54, 55, 56, 57, 58, 14].

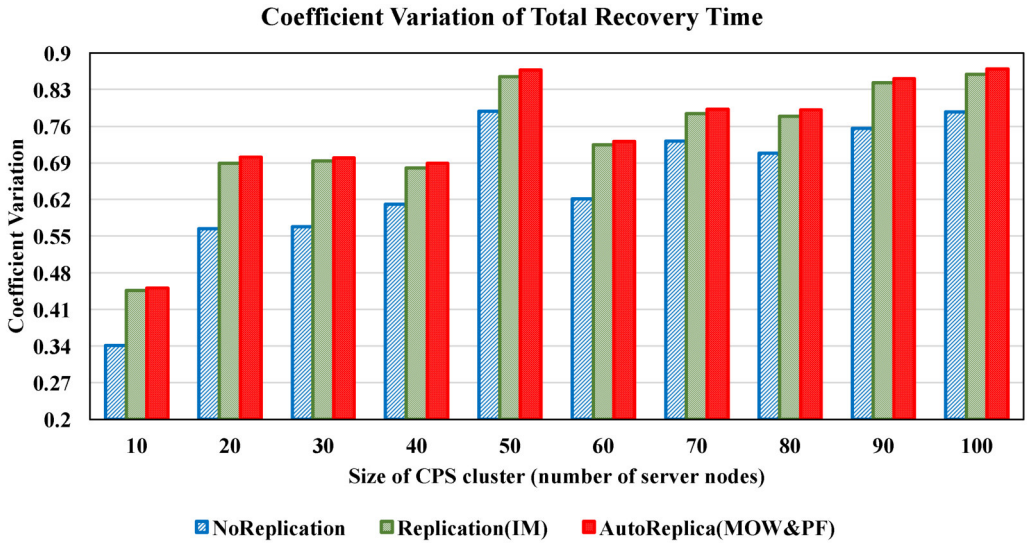

FIG. 6.7. *Coefficient variation of total recovery time for recovery under different heterogeneous CPS cluster sizes.*

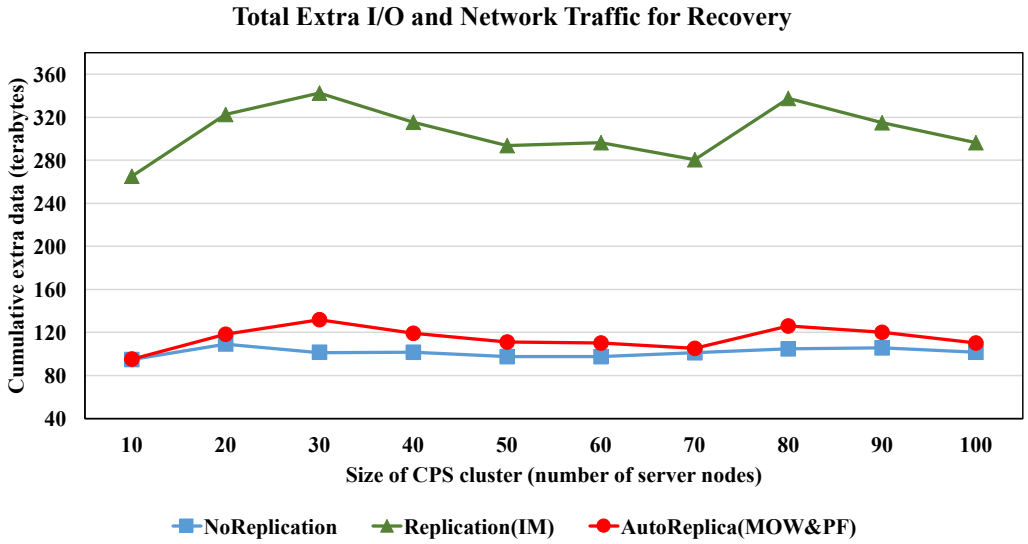

FIG. 6.8. *Total extra I/O and network traffic for recovery under different heterogeneous CPS cluster sizes.*

8. Conclusion. We proposed a complete data replica manager solution called "AutoReplica', working in distributed caching and data processing systems using SSD-HDD tier storages. AutoReplica balances the trade-off between the performance and fault tolerance by storing caches in replica nodes' SSDs. It has three approaches to build the replica cluster in order to support multiple SLAs, based on an abstract "distance matrix" which considers preset priorities, workload temperature, network delay, storage access latency, and etc. AutoReplica can automatically balance loads among nodes, and can conduct seamlessly online migration operation (i.e., migrate-on-write scheme), instead of pausing the subsystem and copying the entire dataset from one node to the other. AutoReplica further supports parallel prefetching from both primary node and replica node(s) with a new dynamic optimizing streaming technique to improve I/O performance. In the future, we plan to work on AutoReplica's compatibility with other hypervisors such as KVM/Xen and Virtual Box.

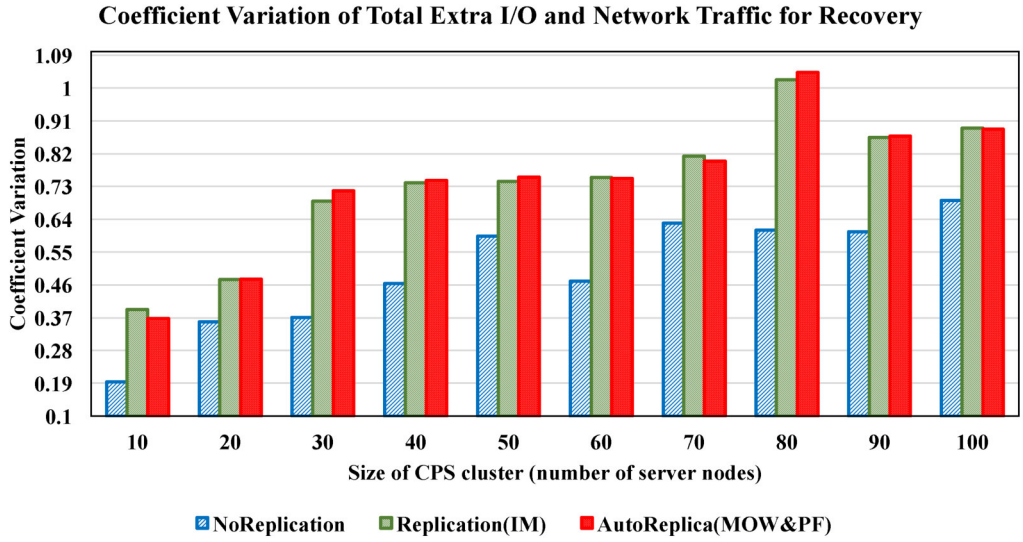

FIG. 6.9. *Coefficient variation of total extra I/O and network traffic for recovery under different heterogeneous CPS cluster sizes.*

Acknowledgment. This work was completed during Zhengyu Yang, Janki Bhimani and Jiayin Wang's internship at Storage Software Group and Performance and Datacenter Team, Memory Solution Lab, Samsung Semiconductor Inc. (CA, USA), and was partially supported by NSF grant CNS-1452751.

REFERENCES

[1] A. A. OMAR, A. GAWANMEH, AND A. APRIL, On the analysis of cyber physical systems, IN *Leadership, Innovation and Entrepreneurship as Driving Forces of the Global Economy*. SPRINGER, 2017, PP. 297–302.

[2] B. CHEN, Z. YANG, S. HUANG, X. DU, Z. CUI, J. BHIMANI, AND N. MI, Cyber-Physical System Enabled Nearby Traffic Flow Modelling for Autonomous Vehicles, IN *36th IEEE International Performance Computing and Communications Conference, Special Session on Cyber Physical Systems: Security, Computing, and Performance (IPCCC-CPS)*. IEEE, 2017.

[3] A. GAWANMEH AND A. ALOMARI, Challenges in formal methods for testing and verification of cloud computing systems, *Scalable Computing: Practice and Experience*, VOL. 16, NO. 3, PP. 321–332, 2015.

[4] B. A. MILANI AND N. J. NAVIMIPOUR, A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions, *Journal of Network and Computer Applications*, VOL. 64, PP. 229–238, 2016.

[5] Z. YANG AND D. EVANS, Automatic Data Placement Manager in Multi-Tier All-Flash Datacenter, PATENT US62/534 647, 2017.

[6] Z. YANG, M. HOSEINZADEH, A. ANDREWS, C. MAYERS, D. T. EVANS, R. T. BOLT, J. BHIMANI, N. MI, AND S. SWANSON, AutoTiering: Automatic Data Placement Manager in Multi-Tier All-Flash Datacenter, IN *36th IEEE International Performance Computing and Communications Conference*. IEEE, 2017.

[7] S. M. TONNI, M. Z. RAHMAN, S. PARVIN, AND A. GAWANMEH, Securing big data efficiently through microaggregation technique, IN *Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference on*. IEEE, 2017, PP. 125–130.

[8] J. ROEMER, M. GROMAN, Z. YANG, Y. WANG, C. C. TAN, AND N. MI, Improving Virtual Machine Migration via Deduplication, IN *11th IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS 2014)*. IEEE, 2014, PP. 702–707.

[9] Z. YANG, M. HOSEINZADEH, P. WONG, J. ARTOUX, C. MAYERS, D. T. EVANS, R. T. BOLT, J. BHIMANI, N. MI, AND S. SWANSON, H-NVMe: A Hybrid Framework of NVMe-based Storage System in Cloud Computing Environment, IN *36th IEEE International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2017.

[10] Z. YANG, M. HOSEINZADEH, P. WONG, J. ARTOUX, AND D. EVANS, A Hybrid Framework Design of NVMe-based Storage System in Cloud Computing Storage System, PATENT US62/540 555, 2017.

[11] J. BHIMANI, N. MI, M. LEESER, AND Z. YANG, FiM: Performance Prediction Model for Parallel Computation in Iterative Data Processing Applications, IN *10th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 2017.

[12] J. BHIMANI, Z. YANG, M. LEESER, AND N. MI, Accelerating Big Data Applications Using Lightweight Virtualization Framework on Enterprise Cloud, IN *21st IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017.

[13] T. Hatanaka, R. Yajima, T. Horiuchi, S. Wang, X. Zhang, M. Takahashi, S. Sakai, and K. Takeuchi, Ferroelectric (fe)-nand flash memory with non-volatile page buffer for data center application enterprise solid-state drives (ssd), in *2009 Symposium on VLSI Circuits*. IEEE, 2009, pp. 78–79.
[14] A. Gawanmeh, Optimizing lifetime of homogeneous wireless sensor networks for vehicular monitoring, in *Connected Vehicles and Expo (ICCVE), 2014 International Conference on*. IEEE, 2014, pp. 980–985.
[15] Z. Yang, J. Wang, D. Evans, and N. Mi, AutoReplica: Automatic Data Replica Manager in Distributed Caching and Data Processing Systems, in *1st International workshop on Communication, Computing, and Networking in Cyber Physical Systems (CCNCPS)*. IEEE, 2016.
[16] Z. Yang, J. Wang, and D. Evans, Automatic Data Replica Manager in Distributed Caching and Data Processing Systems, Patent US15/408 328, 2017.
[17] vSphere Hypervisor, www.vmware.com/products/vsphere-hypervisor.html.
[18] J. Tai, D. Liu, Z. Yang, X. Zhu, J. Lo, and N. Mi, Improving Flash Resource Utilization at Minimal Management Cost in Virtualized Flash-based Storage Systems, *Cloud Computing, IEEE Transactions on*, no. 99, p. 1, 2015.
[19] T. Wang, J. Wang, N. Nguyen, Z. Yang, N. Mi, and B. Sheng, EA2S2: An Efficient Application-Aware Storage System for Big Data Processing in Heterogeneous Clusters, in *26th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2017.
[20] Z. Yang, J. Wang, and D. Evans, A Duplicate In-memory Shared-intermediate Data Detection and Reuse Module in Spark Framework, Patent US15/404 100, 2017.
[21] J. Wang, Z. Yang, and D. Evans, Efficient Data Caching Management in Scalable Multi-stage Data Processing Systems, Patent US15/423 384, 2017.
[22] Z. Yang, J. Tai, J. Bhimani, J. Wang, N. Mi, and B. Sheng, GREM: Dynamic SSD Resource Allocation In Virtualized Storage Systems With Heterogeneous IO Workloads, in *35th IEEE International Performance Computing and Communications Conference*. IEEE, 2016.
[23] Understanding Penalty of Utilizing RAID, theithollow.com/2012/03/21/understanding-raid-penalty/.
[24] Z. Yang and M. Awasthi, I/O Workload Scheduling Manager for RAID/non-RAID Flash Based Storage Systems for TCO and WAF Optimizations, Patent US15/396 186, 2017.
[25] dstat, https://dag.wiee.rs/home-made/dstat.
[26] iostat, https://linux.die.net/man/1/iostat.
[27] blktrace, https://linux.die.net/man/8/blktrace.
[28] K. Chodorow, *MongoDB: the definitive guide*. O'Reilly Media, Inc., 2013.
[29] J. Shi, J. Wan, H. Yan, and H. Suo, A survey of cyber-physical systems, in *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*. IEEE, 2011, pp. 1–6.
[30] K.-D. Kang and S. H. Son, Real-time data services for cyber physical systems, in *Distributed Computing Systems Workshops, 2008. ICDCS'08. 28th International Conference on*. IEEE, 2008, pp. 483–488.
[31] L. Li, J. C. Snyder, I. M. Pelaschier, J. Huang, U.-N. Niranjan, P. Duncan, M. Rupp, K.-R. Müller, and K. Burke, Understanding machine-learned density functionals, *International Journal of Quantum Chemistry*, vol. 116, no. 11, pp. 819–833, 2016.
[32] M. Wojnowicz, D. Nguyen, L. Li, and X. Zhao, Lazy stochastic principal component analysis, in *IEEE International Conference on Data Mining Workshop*, 2017.
[33] L. Li, T. E. Baker, S. R. White, and K. Burke, Pure density functional for strong correlations and the thermodynamic limit from machine learning, *Phys. Rev. B*, vol. 94, no. 24, p. 245129, 2016.
[34] J. Wang, T. Wang, Z. Yang, N. Mi, and S. Bo, eSplash: Efficient Speculation in Large Scale Heterogeneous Computing Systems, in *35th IEEE International Performance Computing and Communications Conference*. IEEE, 2016.
[35] J. Wang, T. Wang, Z. Yang, Y. Mao, N. Mi, and B. Sheng, SEINA: A Stealthy and Effective Internal Attack in Hadoop Systems, in *International Conference on Computing, Networking and Communications (ICNC 2017)*. IEEE, 2017.
[36] H. Gao, Z. Yang, J. Bhimani, T. Wang, J. Wang, B. Sheng, and N. Mi, AutoPath: Harnessing Parallel Execution Paths for Efficient Resource Allocation in Multi-Stage Big Data Frameworks, in *26th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2017.
[37] I. Iliadis, E. K. Kolodner, D. Sotnikov, P. K. Ta-Shma, and V. Venkatesan, Enhancing reliability of a storage system by strategic replica placement and migration, Apr. 25 2017, uS Patent 9,635,109.
[38] L. Cheng, S. Kotoulas, T. E. Ward, and G. Theodoropoulos, Robust and skew-resistant parallel joins in shared-nothing systems, in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, 2014, pp. 1399–1408.
[39] T. Harter, D. Borthakur, S. Dong, A. Aiyer, L. Tang, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, Analysis of HDFS under HBase: A Facebook messages case study, in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*, 2014, pp. 199–212.
[40] A. Cidon, S. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum, Copysets: Reducing the frequency of data loss in cloud storage, in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, 2013, pp. 37–48.
[41] J. Zhang, G. Wu, X. Hu, and X. Wu, A distributed cache for hadoop distributed file system in real-time cloud services, in *2012 ACM/IEEE 13th International Conference on Grid Computing*. IEEE, 2012, pp. 12–21.
[42] Z. Yang, M. Awasthi, M. Ghosh, and N. Mi, A Fresh Perspective on Total Cost of Ownership Models for Flash Storage in Datacenters, in *2016 IEEE 8th International Conference on Cloud Computing Technology and Science*. IEEE, 2016.
[43] J. Bhimani, J. Yang, Z. Yang, N. Mi, N. K. Giri, R. Pandurangan, C. Choi, and V. Balakrishnan, Enhancing SSDs with Multi-Stream: What? Why? How?, in *36th IEEE International Performance Computing and Communications*

Conference (IPCCC), Poster Paper. IEEE, 2017.
[44] Z. Yang, M. Ghosh, M. Awasthi, and V. Balakrishnan, Online Flash Resource Migration, Allocation, Retire and Replacement Manager Based on a Cost of Ownership Model, Patent US15/094 971, US20 170 046 098A1, 2016.
[45] J. Bhimani, J. Yang, Z. Yang, N. Mi, Q. Xu, M. Awasthi, R. Pandurangan, and V. Balakrishnan, Understanding Performance of I/O Intensive Containerized Applications for NVMe SSDs, in *35th IEEE International Performance Computing and Communications Conference*. IEEE, 2016.
[46] Z. Yang, S. Hassani, and M. Awasthi, Memory Device Having a Translation Layer with Multiple Associative Sectors, Patent US15/093 682, US20 170 242 583A1, 2015.
[47] Z. Yang, M. Ghosh, M. Awasthi, and V. Balakrishnan, Online Flash Resource Allocation Manager Based on TCO Model, Patent US15/092 156, US20 170 046 089A1, 2016.
[48] Z. Yang, J. Wang, and D. Evans, Adaptive Caching Replacement Manager with Dynamic Updating Granulates and Partitions for Shared Flash-Based Storage System, Patent US15/400 835, 2017.
[49] L. Cheng, Y. Wang, Y. Pei, and D. Epema, A coflow-based co-optimization framework for high-performance data analytics, in *Parallel Processing (ICPP), 2017 46th International Conference on*. IEEE, 2017, pp. 392–401.
[50] Y. Zhao, C. Li, L. Li, and P. Zhang, Dynamic replica creation strategy based on file heat and node load in hybrid cloud, in *Advanced Communication Technology (ICACT), 2017 19th International Conference on*. IEEE, 2017, pp. 213–220.
[51] R. K. Grace and R. Manimegalai, Dynamic replica placement and selection strategies in data gridsa comprehensive survey, *Journal of Parallel and Distributed Computing*, vol. 74, no. 2, pp. 2099–2108, 2014.
[52] R. Li, W. Feng, H. Wu, and Q. Huang, A replication strategy for a distributed high-speed caching system based on spatiotemporal access patterns of geospatial data, *Computers, Environment and Urban Systems*, 2014.
[53] P. L. Suresh, M. Canini, S. Schmid, and A. Feldmann, C3: Cutting Tail Latency in Cloud Data Stores via Adaptive Replica Selection, in *NSDI*, 2015, pp. 513–527.
[54] C. Liu, R. Ranjan, C. Yang, X. Zhang, L. Wang, and J. Chen, MUR-DPA: top-down levelled multi-replica merkle hash tree based secure public auditing for dynamic big data storage on cloud, *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2609–2622, 2015.
[55] J.-Y. Zhao, M. Tang, and R.-F. Tong, Connectivity-based segmentation for gpu-accelerated mesh decompression, *Journal of Computer Science and Technology*, vol. 27, no. 6, p. 1110, 2012.
[56] J. Zhao, M. Tang, and R. Tong, Mesh segmentation for parallel decompression on gpu, *Computational Visual Media*, pp. 83–90, 2012.
[57] M. Tang, J.-Y. Zhao, R.-f. Tong, and D. Manocha, Gpu accelerated convex hull computation, *Computers & Graphics*, vol. 36, no. 5, pp. 498–506, 2012.
[58] W. Cai, X. Zhou, and X. Cui, Optimization of a gpu implementation of multi-dimensional rf pulse design algorithm, in *Bioinformatics and Biomedical Engineering,(iCBBE) 2011 5th International Conference on*. IEEE, 2011, pp. 1–4.

Edited by: Amjad Gawanmeh
Received: Jun 1, 2017
Accepted: Oct 27, 2017