This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2025.3592275

ELICA: Efficient and Load Balanced I/O Cache Architecture for Hyperconverged Infrastructures

Mostafa Kishani, Sina Ahmadi, Saba Ahmadian, Reza Salkhordeh, Zdenek Becvar, Onur Mutlu, André Brinkmann, and Hossein Asadi

Abstract—*Hyperconverged Infrastructures* (HCIs) combine processing and storage elements to meet the requirements of data-intensive applications in performance, scalability, and quality of service. As an emerging paradigm, HCI should couple with a variety of traditional performance improvement approaches such as I/O caching in virtualized platforms. Contemporary I/O caching schemes are optimized for traditional single-node storage architectures and suffer from two major shortcomings for multi-node architectures: a) imbalanced cache space requirement and b) imbalanced I/O traffic and load. This makes existing schemes inefficient in distributing cache resources over an array of separate physical nodes. In this paper, we propose an *Efficient and Load Balanced I/O Cache Architecture* (ELICA), managing the *solid-state drive* (SSD) cache resources across HCI nodes to enhance I/O performance. ELICA dynamically reconfigures and distributes the SSD cache resources throughout the array of HCI nodes and also balances the network traffic and I/O cache load by dynamic reallocation of cache resources. To maximize the performance, we further present an optimization problem defined by *Integer Linear Programming* to efficiently distribute cache resources and balance the network traffic and I/O cache relocations. Our experimental results on a real platform show that ELICA improves quality of service in terms of average and worst-case latency in HCIs by 3.1× and 23%, respectively, compared to the state-of-the-art.

Index Terms—Hyperconverged Infrastructures, Virtualization, I/O Cache, Load Balancing, Quality of Service, I/O Traffic.

1 Introduction

Hyperconverged Infrastructures (HCIs) replace legacy data center infrastructures by combining storage arrays, storage networks, and computing servers into an array of single physical nodes, managed under software-defined platforms [30], [74], [92]. As a cost-efficient architecture, HCI finds its applications in many different areas, including *Big Data* [89], *Virtual Desktop Infrastructure* (VDI) [9], *Video Surveillance* [2], and *edge computing* [32]. It is also a promising alternative for applications whose computational resource requirement scales linearly with the storage demand. As the *International Data Corporation* reports, the market has a large tendency to HCI, due to the low provisioning cost and great scalability of HCI platforms [35], [36], [37], [38].

An HCI architecture with identical nodes is commonly used in enterprise products such as vSAN from VMWare [64], [85], VxRAIL, VxFlex, and XC from Dell EMC [22], [23], [24], [61], SimpliVity from HPE [34], and Acropolis from Nutanix [66]. In HCI, multiple nodes (e.g., four x86 nodes in Fig. 1) are connected to each other, whereas the storage subsystem is distributed over the nodes, providing scalable capacity and performance. Each physical node contains both storage and processing elements managed by a hypervisor hosting a *Storage Virtual Machine* (SVM) to enable storage sharing across all nodes and multiple *Virtual Machines* (VMs). Both storage and performance capacities monotonically scale by adding extra nodes at *no* service downtime.

Despite various advantages, HCI platforms need to couple with a variety of performance improvement paradigms such as I/O caching in virtualized platforms. I/O caching based on *Solid-State Drives* (SSDs) is widely used in enterprise systems [21], [33], [63]. Existing I/O caching schemes, however, are designed to work with legacy storage architectures and new techniques are necessary to adapt caching schemes into emerging HCI platform. VMs in contemporary HCIs are restricted to *only* use the local cache space of their host node, resulting in unfair resource allocation when one node faces a burst of I/O requests. The main challenges of I/O caching in HCI we aim to solve are: **a) imbalanced cache space requirement**: while some nodes host VMs with high cache space demands, there exist other nodes hosting less demanding VMs and **b) imbalanced I/O load**: some nodes encounter a burst of I/O requests leading to a large cache performance drop, while the rest of the nodes may have idle cache bandwidth. This imbalanced cache management negatively affects *Quality of Service* (QoS) and average I/O latency.

Previous efforts on optimizing the storage subsystem and I/O caching of HCI and multi-node architectures are very limited and are *not* targeted towards optimizing I/O caching across multiple nodes [29], [88], [27], [90], [93], [55], [56], [89], [57]. To the best of our knowledge, **none** of the previous studies have addressed the imbalanced load of I/O caches in HCI. Moreover, the capacity of neighboring HCI node resources in balancing the I/O cache load and decreasing the average and worst-case storage latency has **never** been explored before.

In this paper, we present an *Efficient and Load Balanced I/O Cache Architecture* (ELICA) that provides improved I/O latency and balanced network traffic and I/O cache load in HCI platforms. ELICA distributes the SSD resources within the cluster and shares the total SSD cache space throughout the nodes. To

Sina Ahmadi, Saba Ahmadian, and Hossein Asadi are with the Department of Computer Engineering, Sharif University of Technology, Emails: sina.ahmadi@sharif.edu, saba.ahmadian@sharif.edu, asadi@sharif.edu. Reza salkhordeh and André Brinkmann are with the Department of Computer Science, Johannes Gutenberg-University Mainz, Emails: rsalkhor@uni-mainz.de, brinkman@uni-mainz.de. Mostafa Kishani and Zdenek Becvar are with the Department of Telecommunication Engineering, Czech Technical University in Prague (CTU), Emails: kishamos@fel.cvut.cz, zdenek.becvar@fel.cvut.cz. Onur Mutlu is with the Information Technology and Electrical Engineering department at ETH Zurich, Email: omutlu@gmail.com.

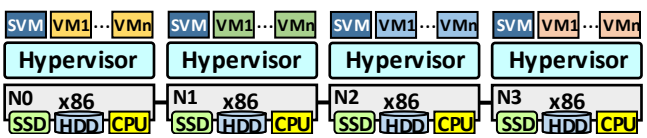


Fig. 1: Example HCI system (VM: Virtual Machine, SVM: Storage Virtual Machine).

achieve the minimum average latency, we define an optimization problem presented by *Integer Linear Programming* (ILP) that determines the efficient local and remote cache size for each VM. By assigning optimum cache size to active VMs and distributing the I/O cache resources within the cluster, ELICA significantly improves both worst-case and average I/O latency. To reduce the amount of network traffic and to balance the I/O load on SSD cache resources, we propose an online workload characterization scheme in which we manage the distribution of cache resources. In contrast to state-of-the-art load balancing schemes [4], [42] that try to cope with large I/O loads by bypassing the cache and directing requests to the disk subsystem leading to performance loss, ELICA responds to all requests from the cache thanks to its efficient I/O cache distribution across the nodes.

In summary, we offer the following **novel contributions**:

- This paper is the first to propose an efficient I/O caching architecture for HCI, distributing the cache resources throughout the nodes and balancing the load across the SSDs. By online I/O characterization of active VMs, we further propose a novel cache resource distribution scheme, which assigns an efficient cache size for VMs and distributes I/O cache resources within the cluster.
- To achieve the minimum average latency, we define an optimization problem using ILP, which assigns an efficient amount of cache space from local and remote nodes to VMs.
- We propose two cache migration policies, namely, *Instant Migration* and *Gradual Migration* to achieve either low network traffic or low cache space penalty in the cache migration process.
- We implement ELICA and our software-defined HCI module on a real cluster with three physical hardware nodes running Proxmox VE as the hypervisor [69] and Ceph [12] as its backend storage system, and evaluate their efficiency using realistic application workloads.
- We offer a dynamic resource allocation mechanism, which allocates and utilizes the available resources significantly more effectively than state-of-the-art schemes, providing an average of 82% cache utilization on all the workloads tested compared to 34% in the state-of-the-art. Experimental results show that ELICA improves average and worst-case latency by 3.1× and 23%, respectively.

The rest of this paper is organized as follows. Section 2 discusses the motivation and the experimental setup. We present our proposed architecture in Section 3 and then detail our I/O cache resource distribution scheme. In Section 4, we discuss the real-system implementation details used in the experiments and then present our experimental results. In Section 5, we briefly review the related work and compare them with our proposed architecture. Finally, we conclude the paper in Section 6.

2 Motivation

In this section, we first briefly review I/O caching in HCI. We then setup and conduct a set of experiments on a cluster of three physical servers to demonstrate the I/O imbalance. Finally, we present our motivational results.

2.1 I/O Caching in HCIs

In existing HCIs, storage elements, namely *disk groups*, are fully controlled by a *Storage Virtual Machine* (SVM). The disk group typically takes advantage of both SSDs and *Hard Disk Drives* (HDDs), where SSDs are used as a caching layer for HDDs [84], [64], [34]. SSDs work as a *global* I/O cache within each node, where the total cache space is shared between the active VMs. However, the I/O cache is *not* distributed within a cluster of HCI nodes; i.e., VMs are restricted to *only* use the local cache space of their hosting node. This architecture can result in imbalanced cache load and cache space usage between HCI nodes hosting VMs with different I/O demands.

Unlike traditional architectures, where compute and storage resources are physically and logically separated, HCI tightly integrates compute and storage into a unified platform. In conventional stacks, compute nodes interact with external storage arrays via network-based protocols (e.g., iSCSI or NFS), which allows independent scaling and optimization. In HCI, however, *Virtual Machines* (VMs) and their associated storage layers (e.g., SSD caches) coexist on the same physical node. This tight coupling introduces several unique performance and management challenges:

- **Resource contention**: Compute, I/O caching, and network traffic contend for shared CPU and memory bandwidth within a node, leading to interference and performance unpredictability.
- **Restricted cache boundaries**: Current HCI caching systems statically allocate SSD cache per node, preventing underutilized nodes from assisting overloaded ones.
- **Costly migrations**: Because compute and storage are co-located, migrating workloads or cache across nodes incurs high overhead, unlike traditional virtualization platforms where only compute or data can be moved independently.

Recent studies illustrate the limitations of existing HCI caching strategies:

- Dasarraju et al. [20] observe that current HCI caching setups such as VMware vSAN and Azure Stack HCI statically provision SSD caches per node, with no support for dynamic redistribution—limiting scalability in practice to 16–64 nodes, and potentially leaving remote cache capacity underutilized during imbalanced workloads.
- Shvidkiy et al. [73] evaluate an OpenStack-based HCI platform and demonstrate up to **40% throughput loss** for write-heavy workloads due to contention between storage and compute pipelines on the same node.
- Taheri et al. [80] benchmark VMware vSAN under high VM density and show a **2–3× latency increase** due to inefficient caching and lack of dynamic allocation support.

These findings emphasize the need for intelligent caching mechanisms that adapt to workload imbalance and enable cross-node coordination. ELICA directly targets these shortcomings by supporting cache migration, load-aware allocation, and global cache visibility across nodes. We incorporate these insights in our motivational setup to further illustrate the cache imbalance challenges in real-world HCI deployments.

2.2 Experimental Setup

Servers Configuration: In our experimental setup, we use a cluster of 2U rack-mount servers connected with 10GbE network. Each node is equipped with our software-defined HCI platform coupled with the free open-source Proxmox [69] hypervisor (version 6.2). Proxmox is a server virtualization and management software, which integrates the KVM hypervisor and *Linux containers* (LXC), software-defined storage and networking functionality in a single platform [69]. Each node employs 32 Intel(R) Xeon cores, 16GB Samsung DDR4 DRAM, one Samsung EVO 860 256GB SSD, and one TB of storage devices (two 500GB SATA 7.2K HDDs).

Storage Subsystem: The storage subsystem is built using the open-source software-defined storage platform *Ceph* (version 14.1.0) [12], providing various access types. We setup Intel OpenCAS in our software stack, which is a block-layer I/O cache and is well maintained and supported by the community [8]. We use SSDs as the I/O cache layer for HDDs. The high-level overview of our setup including the hardware and software stack is shown in Fig. 2.

Workloads: Each VM on the cluster (running CentOS 7.2) is equipped with four virtual CPU cores, 2GB of RAM, and 50GB of disk space, while the SSD cache space is allocated by ELICA. We first run a subset of Microsoft Azure workloads (detailed in [71] and publicly available on GitHub [59]) using 20 VMs on the HCI platform. Details on the number of VMs alongside the percentage of writes are briefly given in Table 1. We also run experiments using a big spectrum of MSR workloads from SNIA [75], [62] (Table 2). We assume write-back and *Least Recently Used* (LRU) replacement policies for the I/O cache.

To examine the effectiveness of our architecture in realistic deployment scenarios, we intentionally construct imbalanced workload configurations across the HCI nodes. While we use authentic Azure and MSR traces as our workload sources, we manually control the placement of VMs—assigning I/O-intensive VMs disproportionately to certain nodes (e.g., Node 2), while placing lighter workloads on others (e.g., Node 0). This design reflects common real-world conditions reported in prior work [7], [28], [81], where uncoordinated VM migrations, heterogeneous demands, and dynamic workload shifts lead to load imbalance across nodes in HCI environments.

2.3 Motivational Results

Quantitative Motivation. To further highlight the limitations of existing cache management strategies in HCI platforms, we conducted additional experiments comparing three approaches: (1) static per-node caching without migration (representing current industry practice, e.g., OpenCAS), (2) a naive round-robin migration scheme across nodes, and (3) our proposed dynamic, demand-driven allocation with cross-node coordination.

As shown in Fig. 4, static allocation results in up to 2.7× higher average I/O latency compared to ELICA, particularly under skewed workloads where certain nodes face cache saturation. The naive round-robin migration introduces cache thrashing effects, degrading latency by 1.6× relative to ELICA. The mentioned results confirm that existing HCI caching strategies fail to handle workload imbalance and skew effectively, stressing the need for the ELICA adaptive, traffic-aware cache orchestration.

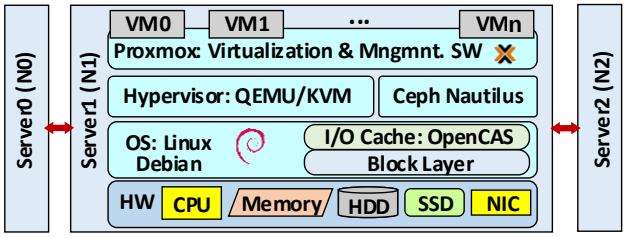


Fig. 2: Implementation stack (hardware and software)

TABLE 1: Microsoft Azure applications and their assignment into the VMs and hosting nodes; (X/Y) denotes *total I/O requests* over *write percentage*.

Node-0 (N0)	Node-1 (N1)	Node-2 (N2)
VM0 (100/64) VM1 (500/7) VM2 (800/9) VM3 (1K/11)	VM4 (3.5K/8) VM5 (10.6K/99) VM6 (20K/99) VM7 (43K/7) VM8 (55K/6) VM9 (88K/7)	VM10 (230K/20) VM11 (472K/6) VM12 (513K/13) VM13 (1200K/10) VM14 (1300K/9) VM15 (1600K/11) VM16 (1800K/6) VM17 (1900K/10) VM18 (2200K/9) VM19 (2700K/12)

To conduct our motivational experiments, we use the Microsoft Azure dataset. In this dataset, there are 855 distinct applications requesting I/O accesses over a two-week period. Among these applications, there is a variation in how frequently they are invoked. While some of them are invoked once per hour, others are called more or less frequently. The nature of these workloads is mostly read-intensive. We group the applications into three categories and select several applications from each group: a) less than 1K accesses, b) between 1K and 100K accesses, and c) more than 100K accesses.

To emulate realistic imbalance in VM distribution, we assign the traces to nodes in a non-uniform manner—placing more I/O-intensive VMs on a subset of nodes to simulate skewed cache demand and stress-test the behavior of our system. This controlled imbalance is crucial to evaluate how ELICA mitigates cache pressure through dynamic resource allocation. For the main experimental results (Sec. 4.2), we conduct the experiments using both workloads (Table 1 and Table 2).

Fig. 3 reports the I/O cache (SSD) utilization and disk subsystem (HDD) utilization for the three server nodes in our experimental setup. The utilization is captured in the SSD/HDD device level using Linux *iostat* [40], reporting the device bandwidth utilization, calculated as the percentage of elapsed time during which I/O requests are issued to the device. The I/O cache utilization on Server 0 in Fig. 3 never reaches 10%. The utilization is capped at 60% for Server 1 with some fluctuations. This could be due to the nature of these workloads being more memory-intensive, reading large chunks of data from the storage in different periods and bringing it into the memory. This way, reading from the cache is sometimes beneficial and sometimes not, justifying the fluctuating disk subsystem utilization. Server 2, on the other hand, has a high density of I/O-intensive applications, showing a rise in I/O cache

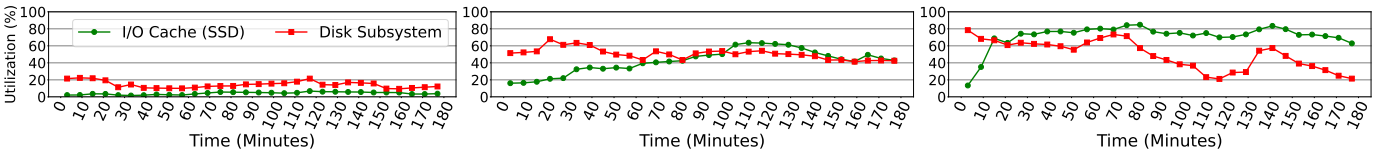


Fig. 3: I/O cache and disk subsystem utilization for a 3-hour run of workloads specified in Table 1.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2025.3592275

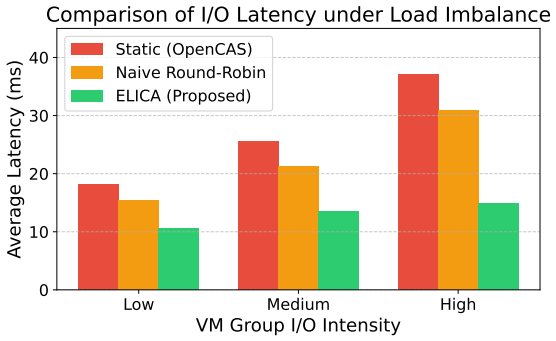


Fig. 4: Latency comparison under different caching strategies and various load conditions.

TABLE 2: MSR Workloads and their assignment into the VMs and hosting nodes; (X/Y) denotes *total I/O requests* over *write percentage*.

Node-0 (N0)	Node-1 (N1)	Node-2 (N2)
VM0:mds_0 (1.2M/12) VM1:rsrch_1 (14K/7) VM2:hm_1 (604K/5) VM3:proj_0 (4.1M/13)	VM4:mds_1 (1.6M/7) VM5:rsrch_2 (206K/34) VM6:src1_0 (37M/44) VM7:proj_1 (23M/11) VM8:stg_0 (2M/85) VM9:prn_0 (5.5M/89)	VM10:rsrch_0 (1.4M/91) VM11:hm_0 (3.9M/65) VM12:src2_0 (1.5M/89) VM13:proj_2 (29M/12) VM14:stg_1 (2.1M/36) VM15:prn_1 (11.1M/25) VM16:usr_0 (2.2M/60) VM17:usr_1 (44M/9) VM18:usr_2 (10M/19) VM19:web_0 (2M/70)

utilization and a decline in HDD utilization in general.

The motivational results serve to show two main challenges of I/O caching in HCI: 1) the imbalanced cache space requirement, and 2) the imbalanced I/O load. As for the first one, for example, in Fig. 3c it is seen that the I/O cache utilization reaches to almost 80% during the run, while in Fig. 3a, the I/O cache is underutilized (less than 10%). This paves the way for an opportunity for using the available cache space on that server, if necessary, to accommodate for workloads with more cache requirements on servers other than the one hosting those specific VMs. As for the I/O load imbalance, one can use the available bandwidth of the I/O cache to serve I/O requests from other nodes. This creates a balanced utilization across all nodes, which helps to improve overall performance. This is what we target to achieve in ELICA, namely, to balance the load of SSDs by allocating remote cache resources from idle nodes to the ones hosting cache-hungry VMs.

3 Proposed Architecture

In this section, we present ELICA and elaborate on how it dynamically partitions the total SSD cache space within the cluster and how it distributes I/O cache resources. We then present how the proposed architecture estimates the cache size of each VM.

3.1 Overview of ELICA

Fig. 5a shows an overview of ELICA. We use HDDs as the main storage media alongside SSDs in the caching layer to address the desired performance at a reasonable cost. To control both hypervisor and shared storage pool, we propose a *Unified Management System* (UMS), which enables configuring the HCI through a central interface. UMS is connected to SVMs and collects information on a) total number of storage elements, and b) allocation of the storage to each individual VM within the cluster. I/O stack management and storage elements handling is also performed by SVM.

The *Storage Manager* of UMS enables ELICA to dynamically partition the total I/O cache space between running VMs and distributes the cache resources through the array of nodes. Fig. 5b shows the steps ELICA takes to manage the I/O cache resources across VMs. It has three main phases:

(a) Inter-VM Cache Management to Determine Size of Cache Resources: We use ILP to determine the size of cache resource allocated to each VM of the local HCI node as well as of remote HCI nodes. This step provides a mapping table of cache resources to VMs. The objective function of ILP is minimum average latency of storage accesses. According to Fig. 5b, ① efficient cache size is estimated, ② it is checked whether the current and new cache sizes are the same, ③ in case the allocated cache size is not changed, we check whether remote cache is allocated, ④ in case no remote cache is allocated, no action is required.

(b) Intra-VM Cache Management to Determine Cache Chunks of Remotely Allocated VMs: We determine which storage requests of VM are handled using local cache resources and which requests use remote resources (⑤ in Fig. 5b). This phase is based on two alternative heuristics, ⓐ trying to reduce the network traffic, called *ELICA-Traffic*, and ⓑ managing to balance the SSD cache load, called *ELICA-Load*. We independently evaluate the performance of both methods for various workloads (Sec. 4.2). Despite Inter-VM cache management that takes place in larger decision periods, Intra-VM cache manager continuously operates during system runtime.

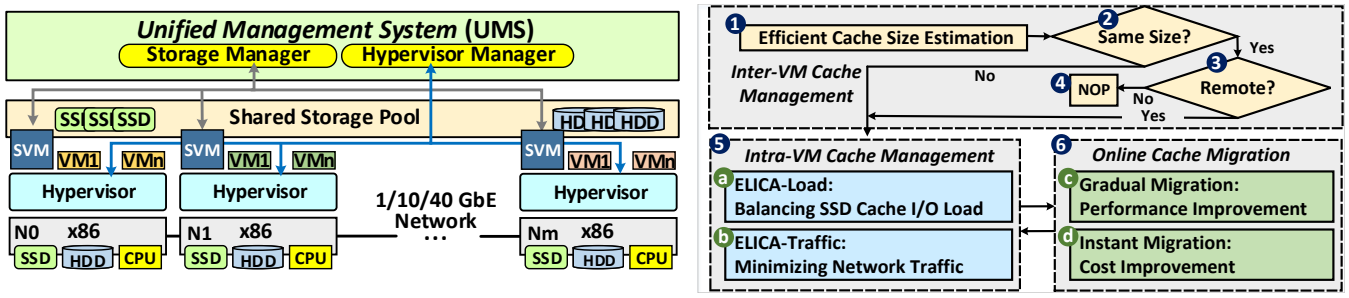
To distinguish local and remote cache hits, ELICA maintains a global stripe-based address mapping table. Each cache block is tagged with ownership metadata that indicates the node responsible for that stripe. When an I/O request arrives, the UMS checks this mapping table to identify whether the data is stored locally or needs to be fetched from a remote cache. This lookup enables accurate redirection of remote cache hits without requiring global broadcast or excessive coordination overhead. The mapping table is kept in-memory and accessed using constant-time (O(1)) hash lookups, contributing negligible overhead to the I/O path.

(c) Cache Migration between HCI Nodes: If either of Inter-VM or Intra-VM cache management phases decide on reallocating a local cache chunk into a remote node (or vice versa), the cache migration is needed (⑥ in Fig. 5b). We propose two alternative heuristics for cache migration, independently evaluated for various workloads (Sec. 4.2), ⓒ *Gradual Migration* and ⓓ *Instant Migration* to achieve either low network traffic or low cost (low cache space penalty).

3.2 Online Cache Size Estimation

To dynamically partition the SSD cache across active VMs in HCI, we propose an online I/O characterization scheme to determine the performance demands of VMs. We track the I/O accesses of VMs inside the SVM and collect the workload parameters such as request type, destination address, request size, and owner VM. To assign an efficient cache size to VMs assuming LRU policy, we evaluate the LRU *Stack Distance* (SD) [3]. ELICA predicts a cache size for each VM based on stack distance. If the predicted cache space is greater than the physical cache space available on the node, we find the optimal cache size for each VM to achieve the minimum average latency. To do so, we define an optimization problem and present it using ILP.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2025.3592275



(a) Overview of ELICA (b) Cache Management Flow of ELICA

Fig. 5: Overview of ELICA architecture and cache management flow.

(a) Ideal Cache Size Estimation: SVMs periodically calculate the reuse distance (*LRU Stack Distance*) of the running VMs at the hypervisor they belong to. The output of this step is a set of $SD_{i,j}$ where i and j denote the node (hypervisor) ID and VM ID, respectively. $SD_{i,j}$ is the maximum stack distance in $VM_{i,j}$ [3]. Stack distance of the current storage block access is equal to the number of distinct blocks accessed since the previous access to the current block [3]. Hence, for a cache size greater than stack distance, the current storage block access is a hit. Assuming LRU and a minor change in the workload behavior in a period of the run, by allocating $B > SD_{i,j}$ blocks of cache to $VM_{i,j}$, those blocks accessed in the future are *not* evicted. Hence, $c_{idl_{i,j}}$, the ideal I/O cache size for $VM_{i,j}$ (VM j of node i) is calculated by Equation (1), where *cacheBlockSize* is set to 8KB.

$$c_{idl_{i,j}} = (SD_{i,j} + 1) \times cacheBlockSize \tag{1}$$

(b) Efficient Cache Size Estimation: UMS collects total I/O cache demands of active VMs from SVMs and allocates the cache space to the VMs throughout the nodes. UMS checks whether the aggregation of estimated ideal cache sizes fits in the cache resources on each node. If the aggregation of ideal cache sizes estimated does not fit in the available cache resources of that node, UMS finds the optimal cache size for each VM by allocating cache resources from both local and remote nodes.

(c) System Model: We consider n_{nodes} disaggregated nodes $N = \{N_1, N_2, ..., N_{n_{nodes}}\}$. N_i $(1 \leq i \leq n_{nodes})$ has the local cache capacity of c_{cache_i} and n_{VM_i} virtual machines, where $VM_i = \{VM_{i,1}, VM_{i,2}, ..., VM_{i,n_{VM_i}}\}$. The cache space allocated to $VM_{i,j}$ is denoted as $c_{i,j}$. The real workloads have usually very large stack distance for some storage block accesses. Consequently, in practice, the aggregation of ideal cache sizes estimated for each node does not fit in the local cache, hence, Equation 2 is not satisfied. Otherwise, the ideal cache size is allocated to each VM using the local cache of each node. In Equation 2, $c_{idl_{i,j}}$ refers to the ideal cache size of j^{th} VM on the i^{th} node.

$$\forall i \in N : \sum_{j=1}^{n_{VM_i}} c_{idl_{i,j}} \leq c_{cache_i} \tag{2}$$

We ensure that the aggregated cache space allocated to VMs does not surpass the aggregated cache capacity of all nodes. We also assume that the cache space allocated to each VM does not surpass the ideal cache space estimated for that VM. The main objective is to minimize the aggregate *AVerage Latency* (AVL) of VMs, defined as follows (parameters listed in Table 3):

$$AVL(VM_{i,j}) = HR(c_{i,j}) \times \frac{1}{c_{local_{i,j}} + c_{remote_{i,j}}} \times [c_{local_{i,j}} \times L_{SSD_{local}} + c_{remote_{i,j}} \times (L_{SSD_{remote}} + L_{Network})] + (1 - HR(c_{i,j})) \times L_{HDD} \tag{3}$$

(d) Problem Definition: We use ILP as an efficient mathematical method to formulate optimization models. The ILP formulation for minimum aggregate AVL is as follows:

$$\min \sum_{i=1}^{n_{nodes}} \sum_{j=1}^{n_{VM_i}} AVL(VM_{i,j}) \tag{4}$$

$$s.t. \quad \sum_{i=1}^{n_{nodes}} \sum_{j=1}^{n_{VM_i}} c_{i,j} \leq \sum_{k=1}^{n_{nodes}} c_{cache_k} \quad (a)$$

$$c_{i,j} = c_{local_{i,j}} + c_{remote_{i,j}}, \quad \forall i \in N, \forall j \in VM_i \quad (b)$$

$$c_{remote_{i,j}} = \sum_{k=1, k \neq i}^{n_{nodes}} c_{remote_{i,j_k}}, \quad \forall i \in N, \forall j \in VM_i \quad (c)$$

$$\sum_{j=1}^{n_{VM_i}} c_{local_{i,j}} + \sum_{j=1, j \neq i}^{n_{nodes}} \sum_{k=1}^{n_{VM_i}} c_{remote_{j,k_i}} \leq c_{cache_i} \quad \forall i \in N, \quad (d)$$

$$c_{i,j} \leq c_{idl_{i,j}}, \quad \forall i \in N, \forall j \in VM_i, \quad (e)$$

The ILP constraints are explained in Table 4. At the first run, cache sizes are evenly distributed across VMs. During each interval t, I/O traces are collected by UMS and the information about cache hits are also monitored. After this interval, the stack distance for each VM is calculated using an efficient and optimal algorithm [65] and then ideal cache sizes are determined. If the aggregation of ideal cache sizes does not fit in available local resources, efficient cache sizes are obtained by solving the ILP problem.

TABLE 3: Definition of parameters used in Equation 3

Parameter	Description
$AVL(VM_{i,j})$	Average latency of $VM_{i,j}$
$c_{i,j}$	Cache size allocated to $VM_{i,j}$
$c_{locali,j}$	Local cache space allocated to $VM_{i,j}$
$c_{remotei,j}$	Remote cache space allocated to $VM_{i,j}$
L_{HDD}	Average read/write latency (i.e., service time) of HDDs
$L_{SSD_{local}}$	Average read/write latency of local SSD cache
$L_{SSD_{remote}}$	Average read/write latency of remote SSD cache
$L_{Network}$	Network latency when remote cache is accessed
$HR(c_{i,j})$	Cache hit ratio of $VM_{i,j}$ from *miss ratio curves* [3], [87]

TABLE 4: Constraints (Cs) of ILP problem defined in Eq. 4

Cs	Description
(a)	Sum of all cache sizes assigned to all VMs cannot exceed aggregate cache size available on all nodes.
(b)	VM cache consists of local and remote cache resources.
(c)	VM remote cache is sum of cache resources allocated from all nodes other than hosting node ($c_{remote_{i,j_k}}$ stands for remote cache space allocated from node k (N_k) to $VM_{i,j}$).
(d)	Node cache size is greater or equal to cache space allocated to local hosed VMs and remote VMs hosted by other nodes.
(e)	Cache space of no VM can exceed ideal cache size.

3.3 I/O Cache Resource Distribution

This section presents how ELICA allocates the I/O cache resources, reducing the network traffic and balancing the I/O load of SSD cache devices. We then reconfigure the cache and based on cache demands of VMs, we reallocate the cache space within the nodes or migrate the cache. Distributing I/O cache resources comes with the following: **First**, the allocation of remote cache resources requires communication through the network. This overhead can be critical, especially when we have limited network bandwidth. **Second**, the balanced load between SSD cache devices can help both system performance and SSD lifetime. **Third**, migrating cache resources should be decided based on the network traffic and cache performance. **Fourth**, cache policy and cache coherency should also be decided. These are addressed next in detail. The proposed scheme for the migration of cache resources is distinguished from VM migration, which is occasionally used for maintenance and failure handling [18], [94], [86].

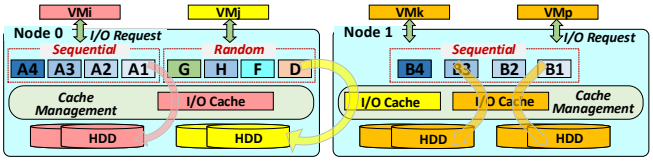
3.4 ELICA-Traffic: Minimizing Network Traffic

ELICA-Traffic monitors the history of accesses and allocates the local cache to the requests with sequential access pattern.

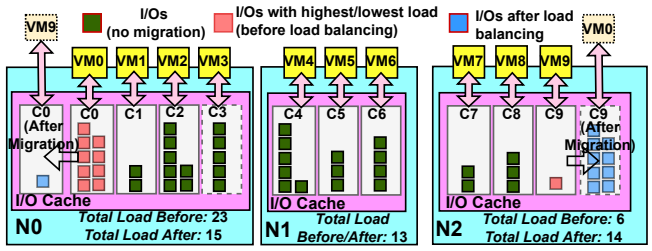
We prioritize placing sequential workloads in local cache over random ones. This design choice is motivated by two key factors: (1) Sequential workloads tend to generate high-bandwidth, predictable access patterns that are more sensitive to network-induced latency and congestion. Keeping them local ensures lower access latency and avoids additional network hops. (2) Caching sequential data locally reduces cache fragmentation, as these accesses exhibit higher spatial locality and can utilize cache space more efficiently.

Our experimental results (Figure 12) confirm that prioritizing sequential requests in local cache improves average latency by **23%** and reduces worst-case tail latency under both MSR and Azure workloads compared to demand-driven allocation policies.

Compared to random workloads, sequential accesses impose higher bandwidth overhead to the network (which is similar to large-size I/O). To reduce the network traffic, ELICA-Traffic allocates the cache space from the local SSDs. If the local resources are not sufficient, we allocate local cache to the sequential requests while the cache for random accesses is allocated from remote nodes (Fig. 6a). ELICA-Traffic recognizes a sequence of requests to consecutive addresses as sequential, when the aggregate size of requests surpasses a threshold (64KB in our experiments). The sequential requests use the local cache resources until a change in the pattern is recognized.



(a) ELICA-Traffic



(b) ELICA-Load

Fig. 6: Overview of ELICA-Traffic and ELICA-Load.

3.5 ELICA-Load: Balancing SSD Cache Load

The purpose of ELICA-Load is balancing SSD cache load within the HCI in a way that the capacity of cache allocated to each VM does not change. The main idea is to exchange cache chunks between busy and idle nodes to balance the SSD cache load. To this end, we first estimate the load on the SSDs. We consider *Queue Depth* (QD), the number of waiting requests in each SSD queue, as a representative for SSD load. A large QD leads to higher latency. ELICA selects n cache chunks from Node A with the maximum load (QD) and exchanges them with n cache chunks from Node B, which has the least load. This way, the load on the busy node is decreased and, consequently, waiting time becomes shorter and the overall performance will improve. Note that n cache chunks of Node A (which has the maximum load) are selected from the VM with the maximum load, while n cache chunks of Node B are selected from the VM with the minimum load.

Fig. 6b shows an exemplary HCI with three nodes and the load on each VM cache. Using the load balancing scheme, ELICA decides to exchange the cache with the highest load (C0) from the busy node (Node-0) with the cache of the lowest load (C9) from the least-busy node (Node-2). Doing so, we reduce the load on Node-0 by 35% (from 23 down to 15) while the load on Node-2 increases from 6 to 14.

3.6 Online I/O Cache Migration

ELICA migrates the cache of a VM from the hosting node to a remote node for two reasons: a) cache space of the local node is not sufficient and b) to balance SSD load. We propose an online migration scheme by considering three key parameters affecting the migration cost: 1) *migration time*, 2) *reserved capacity*, and 3) *bandwidth* imposed on SSDs and network during migration. *Migration time* denotes the duration of moving the VM cache from $node_i$ to $node_j$. *Reserved capacity* stands for the cache space allocated to the VM in the previous location. Once the migration is accomplished, the reserved capacity is released. *Bandwidth* denotes the I/O traffic imposed on the source SSD (due to reading data blocks), destination SSD (due to writing migrated data blocks), and network.

To mitigate the migration cost, ELICA employs two migration schemes: 1) *Gradual Migration* (GM) and 2) *Instant Migration* (IM). GM aims to reduce the *bandwidth* requirements while increasing *migration time* and imposing high *reserved capacity* due to long migration time. In contrast, IM targets minimum *migration time* and aims to reduce *reserved capacity* overhead

while consuming high *bandwidth* and imposing greater I/O load compared to GM. Fig. 7 depicts the migration schemes using a state diagram.

3.6.1 Gradual Migration (GM):

GM migrates cache blocks once they are referenced by a read or write access. To gradually migrate an I/O cache resource from one node to another, we first allocate the required cache space (reserved capacity) in the destination SSD and then perform the following. **First**, all accesses to the storage blocks are directed to the remote cache. **Second**, we check the request type and in the case of write access, we check whether the request hits in the cache or not. In the case of a cache miss (i.e., in both local and remote caches), the access (for writes) is served by the remote cache. Otherwise, in the case of a cache hit in the local cache, the request is served by the remote cache and the corresponding block in the local cache is invalidated. **Third**, in case of read access and upon a cache hit, the request is handled through the remote cache. Otherwise, in case of cache miss, we check whether the request can be served by the local cache or not. In the case of a hit in the local cache, we serve the request by the local cache and also migrate the corresponding block to the remote cache. Finally, in case of a miss in the local cache, the request is served by the disk subsystem and the data is promoted to the remote cache.

GM provides two key benefits: 1) it reduces the number of accesses to the source SSD and 2) it prevents serving the accesses including write, RAW, eviction, and promotion from the local cache while only read hits are responded through the local cache. GM is not efficient regarding *reserved capacity*. It may take a long time while we need to dedicate the cache *reserved capacity* in both local and remote caches. We propose an alternative method, called *Instant Migration* (IM) offering an efficient *reserved capacity* at a bandwidth penalty.

3.6.2 Instant Migration (IM):

IM aims to aggressively migrate the cache resources to minimize the *migration time* and enhance *reserved capacity* while imposing a high *bandwidth* requirement and I/O load. To instantly migrate a cache resource from $node_i$ to $node_j$, we first allocate the required space in $node_j$. We then take the following steps. **First**, we start reading the cache blocks sequentially from the local cache and writing into the remote one. We then invalidate the migrated cache blocks in the local cache. **Second**, the I/O access is first routed to the remote cache and in case of a miss, we check the content of the local cache. If the data block hits in the local cache, we respond it, move the block to the remote cache, and invalidate the local cache. **Third**, in case of a miss in both the remote and local caches, we respond from the disk subsystem and promote the data block to the remote cache. Using IM, we can migrate the I/O cache quickly and release the *reserved capacity* in the local node as soon as possible.

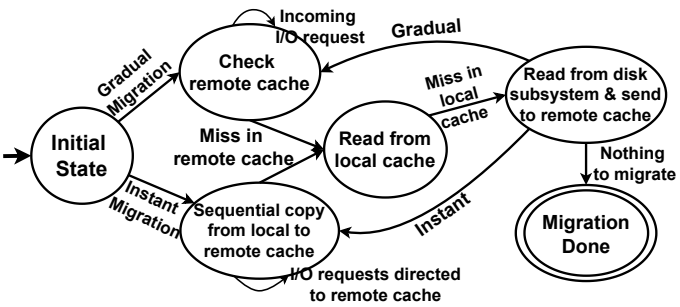


Fig. 7: State diagram of ELICA migration schemes

3.6.3 Network Traffic Considerations

While ELICA allows redirecting I/O requests to remote caches, this redirection introduces additional network traffic, particularly under write-intensive or bursty workloads. To mitigate network traffic overheads, ELICA integrates two complementary mechanisms:

- **Rate-limited Migration**: ELICA limits the number of cache migrations initiated per unit time, ensuring that network bandwidth used for migration does not overwhelm the data plane. This prevents sudden spikes in traffic that could interfere with VM I/O requests.
- **Latency-aware Migration Selection**: Migration decisions incorporate awareness of the current network congestion state. During periods of high congestion, ELICA employs Gradual Migration to spread data movement over time, while Instant Migration is used during low-congestion periods to expedite data redistribution.

These mechanisms ensure that remote cache access and migrations introduce minimal disruption to network performance, while still enabling the benefits of dynamic, load-balanced cache allocation.

3.7 Cache Policy and Cache Coherency

There are generally two types of caching policies, namely, *write-through* and *write-back*. In write-through, data is persisted synchronously in both the SSD cache and disk subsystem. Write-back works asynchronously and keeps the modified data temporarily in the SSD cache while destaging dirty data blocks to the disk subsystem later. One consideration when choosing a cache policy is the workload type. To reduce write traffic and in order to gain maximum improvement in terms of throughput and latency, we choose write-back as our caching policy [48].

Another important parameter is cache coherency, which refers to handling data stored on multiple caches and ensuring the consistency of data across the system. In our proposed I/O caching, coherency could potentially be threatened in the following scenario. When allocating remote cache resources to VMs and dynamic reallocation of cache at runtime which needs data migration, it is possible that data blocks in the migration phase are changed in the application layer. This issue can be studied in three situations.

(a) When data has not been migrated yet. The VM request for changing a block or blocks of data is first applied and then the migration starts.

(b) When data is being migrated to the destination node. After receiving a request, the source node running the application is responsible to notify the destination node on the change and provide it with the new data. The destination node then invalidates incoming data and replaces it with the new one.

(c) When data has been migrated to the destination node. The request will be directed to the remote cache (as discussed in Sec. 3.6). Since the data has been fully placed inside the remote cache and invalidated in the local one, there will be only one copy of the data and coherency is thus preserved.

Instant Migration and Update Safety. In ELICA, both Instant Migration (IM) and Gradual Migration (GM) adhere to the same three-phase consistency mechanism to ensure that data is never moved while being updated. Specifically, during **Instant Migration**, ELICA ensures that:

- All in-flight or pending writes to the cache block are flushed and acknowledged before migration starts.
- Any write arriving during the transfer is intercepted at the source, tracked, and resent to the destination after migration is completed.
- Only after all blocks are safely received and confirmed at the destination node, ELICA updates the global mapping and invalidates the local copy.

This design ensures that even in Instant Migration—which performs migration aggressively without throttling—**data is never moved concurrently with active writes**. Migration only starts after confirming that no write conflicts are present, guaranteeing data correctness.

4 Experimental Results

4.1 Implementation Details

Here, we detail our software stack implemented on physical servers. We first discuss the employed open-source tools and the changes made to accommodate our needs. Then, we detail the UMS implementation. Fig. 8 shows a high-level view of our implementation for a three-node cluster.

4.1.1 Hypervisor

There are several choices when it comes to selecting a hypervisor. One can use a full-fledged hypervisor such as ESXi. While providing many features, such hypervisors are not free. Since we aim to run UMS (details in Sec. 4.1.4) inside the hypervisor, while also keeping the cost at a minimum, we opted for a free and open-source hypervisor. We chose Proxmox [69], which is a virtualization platform based on *Debian* and has built-in features to integrate with *Ceph*.

4.1.2 Storage Subsystem

Choosing a platform for the backend storage is a critical decision as it should provide: 1) *scalability*: to scale out the nodes easily without any downtime or performance degradation, 2) *fault tolerance*: to ensure data accessibility using various techniques such as replication, 3) *compatibility*: to provide access to data in different formats. Ceph is an open-source distributed storage system, which is purely software-defined and has all three features. It uses *object storage* as its underlying layer but has the ability to provide file, block, and object level access through its interfaces. We use Ceph *Rados Block Device* (RBD), which provides block level access to data.

4.1.3 I/O Caching Mechanism

In order to build the I/O caching layer, we chose OpenCAS [8], as it is well maintained and updated. OpenCAS is a block caching software module that accelerates access to main storage. Using

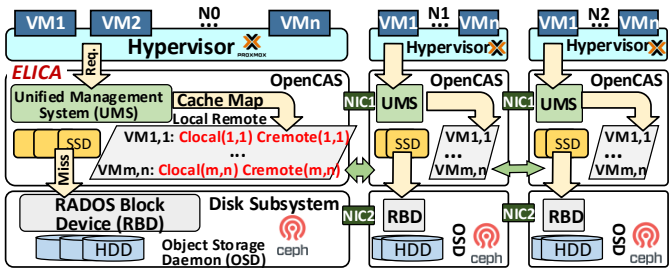


Fig. 8: High-level implementation schema

OpenCAS, we are also able to collect statistics such as cache hit ratio at a low memory and CPU overhead (less than 10% of a CPU core utilization per HCI node). Parameters such as caching mode and line size can easily be configured. In our experiments, we chose the write-back policy with 8KB cache line size. To integrate OpenCAS in ELICA, we made small changes to its source code. As detailed in Sec. 3.3, it is required to access certain blocks of data belonging to an arbitrary VM in the I/O cache. For this purpose, about 100 lines of code were added to OpenCAS so that the required data could be retrieved from the I/O cache.

4.1.4 Unified Management System (UMS)

UMS is made up of four modules written using C++ and Python in about 8K lines of code and runs inside the hypervisor. These modules communicate with each other using *User Datagram Protocol* (UDP) packets, suited for low-latency applications. They maintain a map of current cache sizes (local and remote) for VMs.

(a) I/O Request Capture: It is responsible for taking care of incoming I/O requests. Each VM running inside the hypervisor has a unique *Process ID* (PID) assigned. By using Linux tools such as *blktrace* [10], it can identify the source of each incoming request and make a queue of the requests based on the time they are issued at the VM level. Blktrace has a very low overhead, less than 2% as reported in [1], [54], not seriously affecting the overall system performance.

(b) Dynamic Cache Allocation: Cache stats from previous run are used to find out whether current cache sizes need to be changed by calculating their new *stack distance* for each configurable interval. New remote cache sizes that need to be accommodated by another node are communicated with the corresponding node UMS. The destination node allocates the specified amount of cache space for a VM residing in another node, updates the map, and then notifies the other nodes.

(c) Local/Remote Handling: On distribution of I/O cache resources, cluster network and SSD load are two main concerns addressed in ELICA. Based on the method used, this module handles allocation of local or remote cache to requests. E.g., reads have higher priority for local cache.

(d) Stats Collector: Cache statistics such as hit and miss ratios are collected for each individual VM to be used in the next run by the Dynamic Cache Allocation module.

4.1.5 Workloads

We use two widely adopted real-world traces in our evaluation: the Microsoft Azure VM I/O traces and the MSR Cambridge enterprise workload suite. These workloads represent realistic, production-level I/O behavior observed in cloud and enterprise HCI deployments. Specifically:

- **MSR traces** include I/O patterns from various server classes (e.g., web, development, project servers), capturing diverse access patterns, temporal locality, and request sizes [62].
- **Azure traces** capture VM-level I/O activity in production cloud environments, reflecting multi-tenant, bursty workloads typical of virtualized infrastructures [11].

These traces are commonly used in systems research (e.g., [68], [43], [25], [19], [91], [39]) to benchmark caching, scheduling, and storage policies under heterogeneous and imbalanced loads. Their diversity and realism make them especially suitable for evaluating dynamic cache allocation in HCI systems like ELICA.

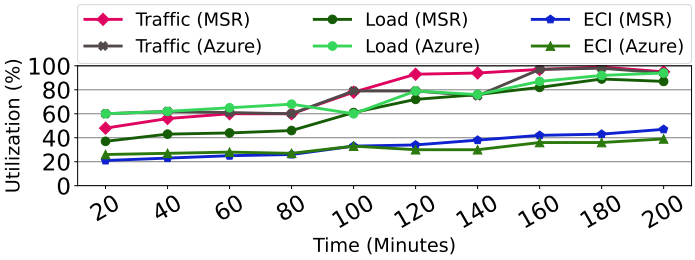


Fig. 9: Cache Utilization

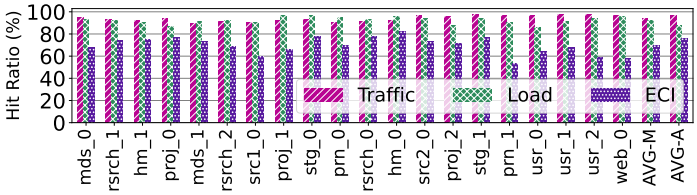


Fig. 10: Hit ratio (AVG-M and AVG-A are average over MSR and Azure workloads, respectively.)

4.2 Experimental Results

In this section, we evaluate ELICA and show how it effectively manages the I/O cache resources in HCI and improves worst-case and average latency. We compare ELICA with the state-of-the-art hypervisor-based caching scheme, ECI-Cache [3], in which the cache resources of each node are locally managed (denoted as ECI). We implement and evaluate both ELICA and ECI-Cache on a physical HCI platform for both Microsoft Azure and MSR workloads (Table 1 and Table 2). We choose our monitoring interval to be 10 minutes, meaning that after this interval, the new cache stats are collected, the ILP is solved in the background, and the new cache sizes are determined for the next interval. A smaller monitoring interval translates to a more accurate cache resource allocation at the cost of a higher computation overhead. The efficient length of monitoring interval, however, is related to the frequency of changes in the workloads. We examine all workloads with different monitoring intervals and realize that the ILP accuracy does not have a considerable improvement for monitoring intervals smaller than 10 minutes.

4.2.1 Cache Resource Allocation

Here we compare cache allocation of ELICA and ECI, residing in the hypervisor. We use GM to move I/O cache resources. Fig. 9 shows the cache resource usage for each method for MSR and Azure workloads. We make the following major observations. ELICA increases the I/O cache allocation by 2.7× compared to ECI for MSR and 2.4× for Azure workloads. This is thanks to our global resource allocation. VMs in ECI are restricted to *only* use the cache of the local nodes. Hence, some nodes face the lack of cache space while some other nodes have idle cache space. Considering the average of workloads on all three nodes, ELICA demonstrates 76% and 70% higher cache resource utilization compared to ECI for MSR and Azure workloads, respectively.

To better compare performance results, Fig. 10 compares the hit ratios of ELICA-Traffic and ELICA-Load schemes as well as ECI. By efficient cache allocation, ELICA shows 33% and 35% higher hit ratio compared to ECI for the MSR and Azure workloads, respectively. This improvement was expected since in both ELICA-Traffic and ELICA-Load, more cache resources are allocated to VMs. An important observation is the greater impact of ELICA on heavy workloads. For example, for prn_0 (on VM9), which is an I/O intensive workload, the improvement of ELICA in both Traffic and Load schemes is approximately 76%. This shows when local cache is not enough, remote cache allocation is very beneficial.

4.2.2 I/O Cache Resource Distribution

We examine ELICA to show how the proposed resource manager effectively distributes the VM cache resources across the HCI array. ELICA aims to use both local and remote SSDs for cache allocation, achieving an efficient resource distribution when the demands of nodes have a large variance. Fig. 11 shows the percentage of remote resources allocated by ELICA-Load and ELICA-Traffic. We can make the following major observations. ELICA-Load allocates 37% remote resources, while ELICA-Traffic allocates 31% remote resources on both MSR and Azure workloads. To reduce the I/O load on the local SSDs, ELICA-Load moves the VMs cache resources to remote nodes. Meanwhile, ELICA-Traffic that targets reducing the network traffic aims to allocate fewer number of remote resources to VMs.

VMs running on *Node 0* have the least amount of remote cache allocated to them (Fig. 11). Since the server hosts fewer VMs running on this node, and except for one workload (proj_0 on VM3), the other three are less I/O intensive, it rarely faces lack of cache space and it can accommodate all the requests using its local cache. Another observation is the higher amount of remote cache allocated to write-heavy workloads. For example, VM12 is running src2_0, which is write-intensive and it has a high amount of remote cache.

4.2.3 Performance Improvement

We compare the performance of ELICA-Load and ELICA-Traffic with ECI in terms of worst-case and average latencies of active VMs using *iostat* [40]. Compared to ECI, ELICA achieves higher hit ratio and lower latency since it distributes the I/O caches among the nodes. Hence, VMs can use the free space of SSDs in the remote nodes. Besides, ELICA balances I/O load in the HCI array. In case of high I/O load on the nodes hosting VMs with high access frequency, the cache resources are moved to the nodes with idle SSD bandwidth.

4.2.4 Average Latency Improvement:

Fig. 12a compares the average latency of ELICA-Load and ELICA-Traffic with ECI. Using GM and IM on MSR workloads, and compared to the ECI based HCI, ELICA-Load improves the average latency by 2.76× (176%), while ELICA-Traffic shows a 3.37× (237%) latency improvement. These numbers are 2.83× for ELICA-Load and 3.44× for ELICA-Traffic on Azure workloads. This can be explained by the fact that ECI only allocates local cache and fails to provide efficient amount of cache to the VMs, leading to more frequent HDD accesses. For a workload such as N0-VM3 (running proj_0), ELICA mostly allocates local cache with very minimal remote cache and is able to achieve almost

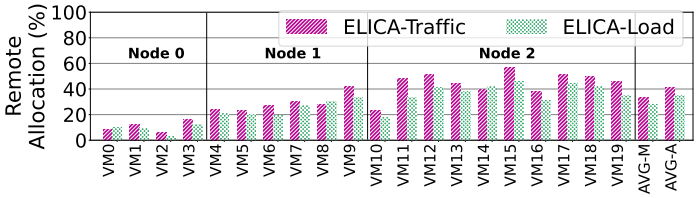


Fig. 11: Percentage of remote cache allocated

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2025.3592275

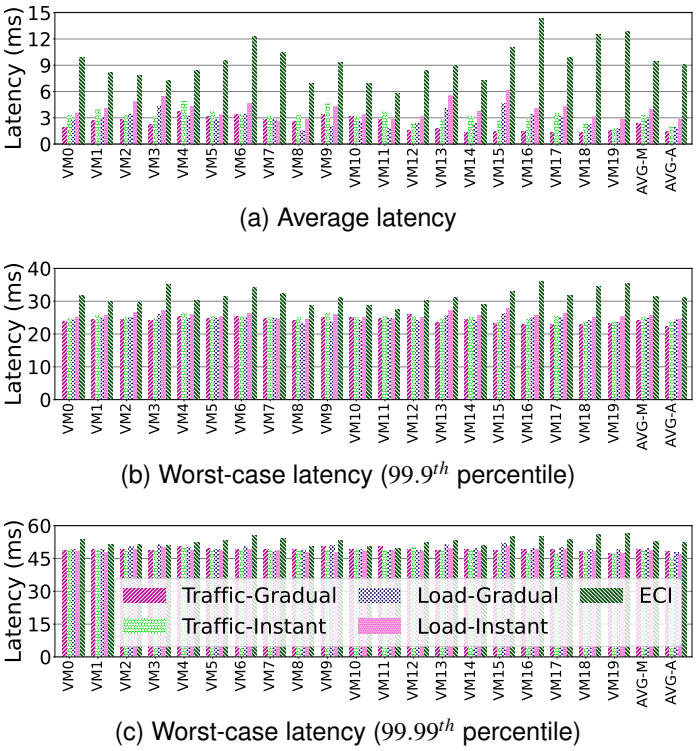


Fig. 12: GM vs. IM: average and worst-case latency (AVG-M: average over MSR, AVG-A: average over Azure)

1.9× latency improvement over ECI. The significant improvement is due to more cache space released by moving highly loaded I/O caches to other nodes. Workloads that are assigned more remote cache tend to have higher average latency compared to workloads with more local cache. This latency, however, is still much lower than the case where all workloads on a server use only the local cache available on that server (e.g., ECI), exhausting its cache resources.

4.2.5 Worst-Case Latency Improvement

Fig. 12b and Fig. 12c compare the worst-case latency achieved by ELICA and ECI based HCI for the 99.9^{th} and 99.99^{th} percentiles, respectively. The average results over both MSR and Azure workloads are also shown in this figure. ELICA-Load and ELICA-Traffic achieve an average worst-case latency improvement of 23% and 13% for three and four 9's, respectively. Moving from three-nine to four-nine percentile, the tail latency for both ELICA methods seems to get closer to ECI. This is because worst-case latency is highly correlated to HDD queue depth. As these parameters increase, the waiting time for I/O requests also increases, resulting in longer latency. It is also the case that remote cache may not be available to be assigned to a VM cache at some point (due to reasons such as cache space used up by other VMs). This will cause ELICA to act very similar to ECI, allocating and using only the local cache.

4.2.6 Per-Node Throughput Under Varying Load

To evaluate the ELICA scalability and performance under varying stress levels and application profiles, we measured I/O throughput across nodes under three load conditions, using workloads representative of real-world HCI applications.

Workload Profiles:

- **OLTP-like workload (similar to TPC-C pattern):** Simulates small, random, write-heavy transactional workloads, representative of database backends.
- **OLAP-like workload (similar to TPC-H pattern):** Simulates large, sequential, read-intensive analytical queries.
- **Mixed workload (enterprise-like file server pattern):** Emulates a mix of sequential and random I/O with varying read/write ratios.

Load Profiles and Mapping to Applications:

- **Low Load:** Nodes primarily run OLAP-like VMs with low I/O frequency, emulating off-peak analytical workloads.
- **Moderate Load:** Nodes host a balanced mix of OLTP, OLAP, and mixed workload VMs, reflecting typical enterprise activity.
- **High Load:** Nodes run multiple OLTP-like VMs with aggressive random I/O, simulating peak transactional demand.

Results: Fig. 14 shows the per-node aggregate throughput (in MB/s) under each scenario:

- Under **low load**, Nodes 0–2 achieve throughput between 280–300 MB/s, reflecting optimal cache usage and minimal queuing with read-dominant traffic.
- Under **moderate load**, throughput remains stable at 260–270 MB/s, as the ELICA adaptive cache allocation ensures balanced performance across diverse I/O types.
- Under **high load**, throughput drops to 215–230 MB/s due to SSD queuing pressure from write-heavy, random OLTP-style workloads. However, the degradation is controlled, as the ELICA traffic-aware balancing and migration prevent bottlenecks.

These results, illustrated in Fig. 14, confirm that ELICA maintains robust throughput and scalability across varying application-level workloads and system stress conditions, validating its ability to support realistic, end-to-end HCI applications.

4.2.7 ILP Computational Overhead

In ELICA, we solve ILP in each 10-minute interval. By limiting the search space and defining bounds using *Branch and Bound* method, we can solve ILP with a reasonable time and memory complexity. To show the ILP overhead, we perform computations using different number of VMs. Fig. 13 shows what percentage

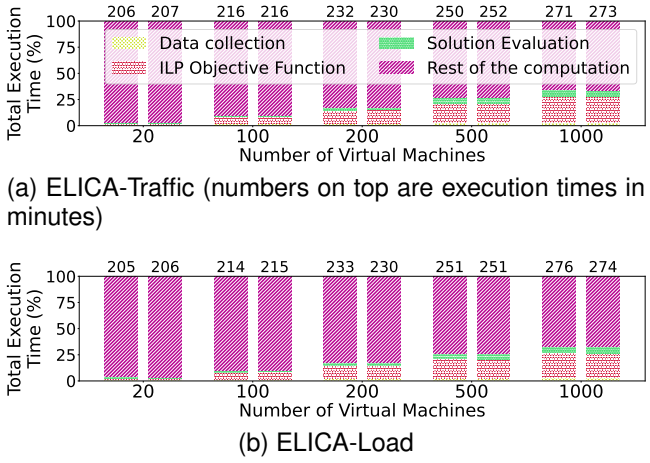


Fig. 13: ILP overhead (Left bar: GM and right bar: IM)

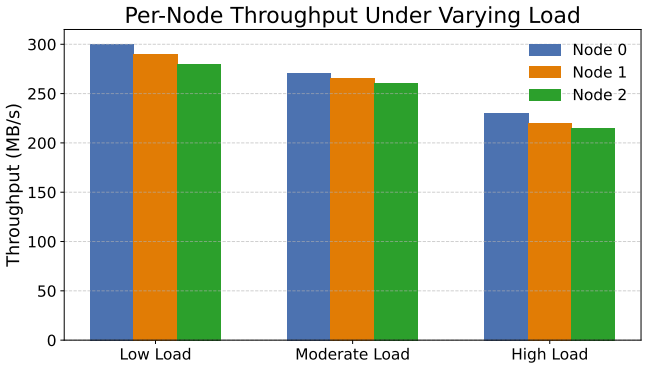


Fig. 14: Per-node throughput under three different load scenarios.

of the total execution time solving ILP takes. The execution times (in minutes) for each run is demonstrated on top of each bar. To evaluate the ILP overhead, we stop the system and run the ILP offline and compare it with the concurrent system run. ILP calculation consists of three main phases. First, all information regarding cache sizes are collected from different nodes. Second, the ILP objective function is solved and new cache sizes are obtained. Third, the most recent ILP solution is evaluated. The results show the overhead is by 18% of the total execution time when 500 VMs run in the cluster. ILP computations are executed using *only* a single thread on one of the servers. While ILP is not memory-intensive, it is compute-intensive, imposing overhead on a single CPU core. This overhead can be evenly distributed over multiple CPU cores by extending our implementation to a multi-threaded program. Fig. 18 shows the CPU overhead of ILP in a single-thread implementation. Increasing the ILP computation interval is another approach to reduce this overhead. Besides, as HCI is mostly targeted for small to medium-size deployments, we conclude that the ILP computation overhead is negligible and can easily be compensated by either increasing time interval or multi-threaded programming.

Our choice of ILP is motivated by its structured modeling of system constraints and the availability of efficient solvers that return near-optimal solutions using hybrid optimization techniques (e.g., branch-and-bound, heuristics). While ILP is NP-hard, our workload-aware formulation remains lightweight and solvable within seconds even at large VM scales. The solver operates asynchronously in the control plane and is decoupled from the I/O path, making it a practical choice for dynamic cache allocation.

We acknowledge that alternative approaches such as heuristic algorithms or *Machine Learning* (ML) models could also be explored. However, for this work, we prioritized explainability and minimal runtime dependencies. ML-based cache management, especially using reinforcement learning, remains a promising direction for future research.

4.2.8 ILP Accuracy and System Impact

ILP Prediction Accuracy. To evaluate how well ILP models real-world cache demands, we compared the cache sizes predicted by the solver to the actual usage measured at runtime. As shown in Fig. 15, the majority of predicted allocations closely match observed usage.

- The *Mean Absolute Error* (MAE) between predicted and actual cache usage is **24.3 MB**, which is low given that average per-VM allocations are around 512 MB.
- **85% of VMs** experienced cache size prediction errors within **10%** of actual usage.

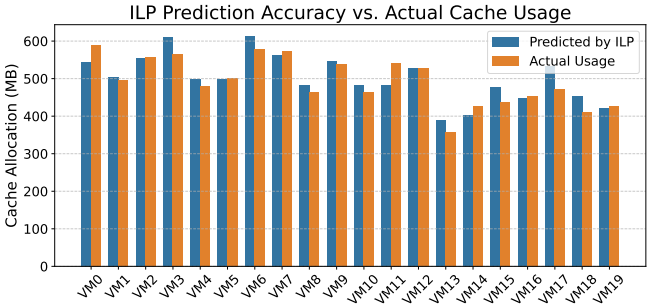


Fig. 15: ILP prediction accuracy: per-VM predicted cache size vs. actual cache usage.

4.2.9 SSD Endurance and Wear-Leveling

SSD endurance is primarily affected by write amplification and sustained write volume. ELICA mitigates these effects through a combination of traffic-aware migration and adaptive cache allocation:

- **Write-aware migration:** Gradual migration spreads writes over time, avoiding sudden bursts that can prematurely degrade flash cells. This leads to more uniform wear leveling across devices.
- **Workload-aware placement:** Write-heavy workloads are not preferentially allocated to a single node. Instead, cache space is distributed based on traffic patterns, avoiding overuse of any specific SSD.
- **Observed workload balance:** In our experiments, workloads exhibited a wide range of read/write ratios (6%–85% writes across VMs). ELICA adapts to this heterogeneity and ensures no node becomes a hotspot for write activity.

To quantify the impact of ELICA on SSD endurance, we track cumulative SSD writes per node under skewed load. As shown in Fig. 16, static cache partitioning results in a significantly higher write volume on Node 1. In contrast, ELICA redistributes the load, leading to a more balanced write profile across all nodes.

We also estimate SSD wear by comparing the percentage of P/E cycle budget consumed across nodes. As shown in Fig. 17, the static policy results in uneven wear, with Node 1 consuming 45% of its P/E budget. ELICA maintains uniform wear levels at approximately 30% per node.

4.2.10 Network Bandwidth Utilization across servers

The main overhead of ELICA on the network is transmitting data blocks. To monitor network usage, we use *IPTraf* [41]. Fig. 19 shows the total network bandwidth utilization for three servers in the experiments. We only report usage for ELICA related communication while leaving out the usage for non-ELICA processes. The network utilization in our study does not go beyond 40% in total (14%, on average).

4.2.11 GM vs. IM

We first redo our previous experiments using IM to make a comparison on performance in terms of average and worst-case latency. We then perform experiments to compare the performance of our proposed migration schemes in terms of *bandwidth* and *migration time* to show possible trade-offs.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2025.3592275

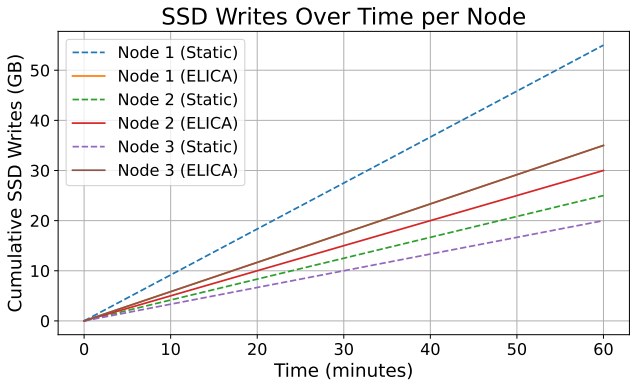


Fig. 16: Cumulative SSD writes over time under skewed load. Static policy overuses Node 1, while ELICA distributes writes more evenly.

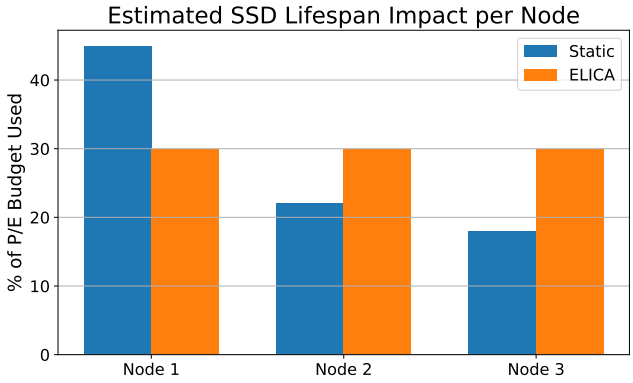


Fig. 17: Estimated SSD wear (% of P/E budget used per node). ELICA achieves better wear leveling across the cluster.

Fig. 12 compares the average and worst-case latency achieved using two migration schemes. For the average latency, IM is about 23% faster (Fig. 12a), while for the worst-case latency, we observe a 10% improvement. We expected to see better latency results for IM since it aims to quickly migrate cache resources to the destination node, while GM only migrates when a corresponding request is issued. This causes an overhead in searching as ELICA needs to search both local and remote caches to find the desired data blocks. Although this can also happen in IM, it would not last for a long time compared to GM, hence resulting in lower latency.

Fig. 20a shows the total amount of time spent and bandwidth used for two migration schemes of three sample workloads. These workloads have the most remote cache resources allocated (Fig. 11). Using GM, it takes a maximum of 14 minutes to migrate local cache to the other nodes for *prn_1* (VM15). The migration happens at a bandwidth of up to 7 MB/s, while using IM finishes the migration in less than two minutes at up to 223 MB/s bandwidth (7× faster but with 31× higher bandwidth overhead than GM). The same scenario is valid for other two workloads. To summarize, IM provides faster migration at a higher bandwidth while GM takes bandwidth into account and does the migration at a slower pace.

To ensure data correctness during migration, we run 10,000 synthetic migration events under write-heavy workloads. Each migrated block is verified using SHA-256 checksums. Table 5 summarizes the results for both gradual (GM) and instant (IM) migration. In all runs, we observed zero block corruption, confirming that ELICA preserves cache consistency.

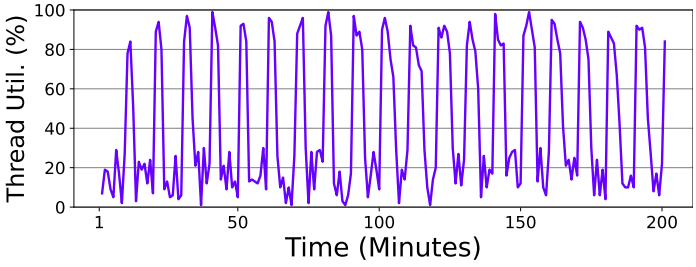


Fig. 18: Utilization of single-thread implementation calculating ILP over time.

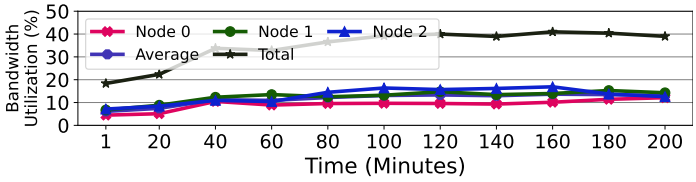


Fig. 19: Servers Bandwidth Utilization

TABLE 5: Cache consistency validation under synthetic migration tests

Metric	Instant	Gradual
Migrations tested	10,000	10,000
Corrupted blocks detected	0	0
Integrity verification success	100%	100%

5 Related Works

In this section, we show why previous schemes cannot be effectively applied to the existing HCIs. Several studies [77], [78], [72], [45], [47], [46], [44] developed various schemes to enhance the reliability of storage arrays and HCIs while *none* of them takes the performance improvement of HCIs into account. Table 6 summarizes contribution domains of previous works.

5.1 I/O Caching in Virtualized Platforms

Several SSD-based I/O caching schemes are proposed for virtualized platforms, improving performance, while considering endurance and reliability [53], [58], [49], [3], [6], [42], [4] (Table 6). Contemporary I/O caching schemes for virtualized platforms mainly focus on partitioning the total SSD cache space between running VMs to maximize the performance. Such schemes, however, are optimized to single host structures. They are also not compatible with HCIs as they lead to imbalanced I/O load and cache space allocation (as discussed in Sec. 4.2). A group of studies also manage to estimate an efficient cache size for virtualized platforms [6], [53], [58], [49], [3] using parameters such as *working set size*, locality, and reuse intensity of the running workloads. Few studies (e.g., CloudCache [6]) propose to migrate the entire VM (not only the I/O cache) to other hosts in case of insufficient cache space. Recent work [51] has explored machine learning-based approaches for fair and efficient cache management in NVMe SSDs, demonstrating the potential for intelligent caching strategies in virtualized environments.

5.2 Load Balancing in Distributed / Storage Systems

Several schemes migrate high-load VMs to nodes having less load [7], [16], [67]. However, costs associated with live migration of VMs could be prohibitive in practical deployments of HCI such as edge computing [17], [76], [95]. SIB [42], LBICA [4], and DistCache [52] are three recent I/O cache load balancing schemes. In case of burst and heavy I/O accesses, both LBICA and SIB balance the load of the SSD cache by bypassing the requests into the disk subsystem. Hence, the requests that are considered to be responded by the cache (hit) experience the high latency of disk subsystem, which leads to considerable I/O performance drop. These schemes also do not take the distributed architecture of HCIs into account.

In [15], the intersection of data deduplication and migration strategies have been explored as a way to achieve load balancing in distributed storage systems, highlighting the complex interplay between data management and system performance.

Recent studies have proposed another category of cache management techniques to improve fairness among competing applications accessing shared SSDs [50], although they use DRAM as cache.

5.3 I/O Cache Migration

Online migration of data across data centers in a geo-distributed manner is addressed in [82], where *distributed storage overlays* are introduced to cache data objects and facilitate data migration. Akkio [5] is a locality management service, deciding on when and how to perform data migration regarding the data access footprint, reducing the response time and resource usage. Clark et al. [18] investigate the live migration of entire VMs, which is completely different from I/O cache migration proposed in ELICA. VM migration comes with downtime in the operation or serious performance degradation and has a large migration time and network traffic, only used for occasions such as failures, maintenance, and hardware/software updates [18], [94], [86].

5.4 I/O Caching in Commercial Hypervisors

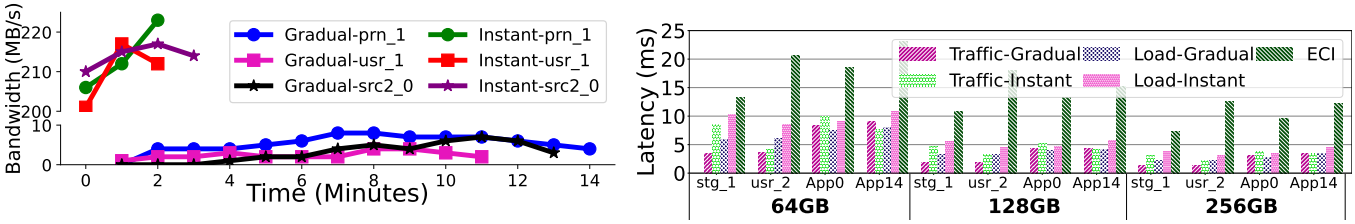
While commercial hypervisors [60], [84], [83] support I/O caching in both guest and host file-systems, they do *not* support distributed I/O caching. The deprecated VMware flash read cache (known as VFlash), as well as a third-party cache software such as *Virtunet* [79] developed to integrate with VMware, only use the local host resources. VMware vSAN [84], [83] also uses local cache resources in both all-flash and hybrid HCI solutions.

5.5 I/O caching in alternative platforms

Remero et al. [71] proposed a distributed cache for *Function-as-a-Service* (FaaS) applications. The service paradigm in FaaS applications is based on microservice containers that have a different software stack from VMs in HCIs. The work presented in [71] is useful for DRAM cache management, while ELICA manages SSD I/O cache. Hence, the contribution of [71] is not necessarily applicable to HCI platforms. In a similar effort, Wang et al. [88] propose a cache memory for FaaS application, relying on DRAM cache and a distributed architecture. To address cold start problem in FaaS applications, Fuerst and Sharma [29] propose FaasCache that keeps frequently used FaaS functions alive after they have finished execution. None of these studies use cache migration as a facility to balance the cache load. Farshin et al. [27] explore the use of *Direct Cache Access* (DCA) facility in Intel processors to improve packet processing latency in *Software-Defined Networks* (SDNs). Wang et al. [90] also explore DCA for SDNs in a recent study. Ge et al. [31] use I/O access hints in favor of I/O latency. Wu et al. [93] propose a caching mechanism for data storage systems, adapting to the characteristics of emerging storage media. As both works focus on a single data storage node, they do not deal with challenges such as imbalanced load between the nodes.

5.6 Discussions and Future Work

(a) Mirrored I/O Cache: One future direction of this research is distributing the mirrored cache into two different HCI nodes. When caching the write-pending data, it can be written into two cache modules for the sake of performance and reliability. ELICA makes room for distributing the mirrors into two different HCI nodes rather than a single node, doubling the read bandwidth of the local I/O cache. While this scheme can improve the average read latency, the overhead of addressing cache coherency upon the write requests should be explored.
(b) Endurance of I/O Cache: Data migration may have an impact on the endurance of SSD devices, depending on the selected migration mechanism. In GM, no extra cache write is imposed, as the cache is gradually migrated by redirecting new write operations. IM, however, comes with write overheads, as it aggressively migrates all cached data to the new destination.
(c) I/O Cache vs. Page Cache: In ELICA, we mainly focused on the I/O cache of HCI block storage that resides in the I/O block layer, just below the page cache. The page cache and I/O cache work independently in different layers of Linux storage stack, while optimizations in each layer can independently help overall storage performance [26], [14], [13].
(d) Reliability: Cache coherency concerns are fully addressed in ELICA, as discussed in Sec. 3.7. Both ELICA and ECI use SSD in the caching layer, resulting in similar reliability obligations. By employing mirrored I/O cache, ELICA outperforms ECI in the case of motherboard and/or system failures, which is single-point-of-failure in the ECI architecture. Nevertheless, a higher reliability level can be achieved only when data blocks are also replicated in multiple nodes.
(e) I/O Cache Size: To show the efficiency of ELICA with smaller cache sizes (64GB and 128GB), we perform experiments on four sample workloads (VM0 and VM14 chosen from Table 1 and stg_1 and usr_2 chosen from Table 2). Our experiments show ELICA improves average latency by 2.5× (151%), 3.4× (241%), and 3.5× (253%) over 64GB, 128GB, and 256GB cache sizes, respectively, compared to ECI. ELICA also improves worst-case latency by 26%, 30%, and 31% over 64GB, 128GB, and 256GB cache size, respectively (detailed results are removed for the sake of brevity). Further analysis and use of heterogeneous cache size on different nodes is another future direction for this research.
(f) All Methods in One: While in this study we examined ELICA-Traffic and ELICA-Load separately, combining both ELICA-Traffic and ELICA-Load methods is left for future work, as they seem to have a co-dependency. As an example, in trying to balance the load, we need to accept some traffic overheads and vice versa.
(g) Comparison with Commercial Systems: We compare ELICA against ECI-Cache [3], a prior hypervisor-based caching system that supports per-VM cache partitioning and workload-aware allocation. Its architecture closely aligns with our goals, enabling a meaningful technical comparison. In contrast, commercial platforms such as VMware vSAN, OpenShift, and Ignite, differ significantly in design: vSAN offers static, node-local caching without per-VM migration; OpenShift targets containerized storage without block-level cache control; and Ignite requires application-level integration. These differences make direct, fair comparison infeasible. Future work will explore integrating ELICA as a pluggable cache orchestrator into such environments.



(a) GM vs. IM for three sample workloads

(b) Avg. Latency (cache size= 64GB/128GB/256GB)

Fig. 20: Comparison of a) Gradual vs. Instant Migration schemes and b) average latency.

TABLE 6: Domain of contributions in previous works

System Type / Contributions	Cache/VM Migration	Load Balancing	Unified Cache	Cache Size Alloc.	Reliability/Endurance/Coherency
HCI	ELICA	ELICA	ELICA	ELICA	[77], [78], [72]
Distributed Storage/Virtualized	[82], [5], [18]	[70], [7], [16], [67], [52]		[6]	[6], [42], [4]
Serverless		[71], [88]	[71], [88], [29]	[71], [88], [29]	[88]
Single Storage/Virtualized		[42], [4], [31], [93]		[49], [3], [53], [58]	[45], [47], [46], [44], [49], [3], [53], [58]

(h) Relationship to In-Memory Caching Systems: Distributed in-memory caches such as Apache Ignite, Memcached, and Redis are designed for application-level data acceleration, typically requiring explicit integration by developers. Distributed in-memory caches, cache the structured data (e.g., key-value pairs) in RAM to reduce access latency. In contrast, ELICA operates at the storage layer of HCI platforms, transparently managing SSD-based block-level cache for virtual machines. Unlike in-memory caches, ELICA requires no application modifications and addresses system-level concerns such as cache migration, load imbalance, and endurance. The two approaches are complementary: in-memory caching improves data access at the application tier, while ELICA improves storage I/O performance at the infrastructure tier.

6 Conclusion

In this paper, we presented an I/O caching architecture for HCIs, called ELICA, which allocates efficient cache spaces to VMs and effectively distributes the I/O cache resources throughout the HCI array, improving the QoS in terms of average and worst-case latency. To effectively partition the SSD cache space across active VMs, ELICA estimates an efficient cache size for each VM based on workload characteristics and distributes the I/O cache resources across the HCI array. To this end, an ILP optimization problem is presented. To reduce the performance overhead imposed by migrating the I/O cache, we proposed two online cache migration schemes, namely, Gradual and Instant Migration. The first migration scheme benefits network traffic, while the latter one manages to achieve the best cache space efficiency by migrating the I/O cache as soon as possible. Our experiments on a real platform revealed that ELICA improves the QoS in term of average and worst-case latency by 3.1× and 23%, respectively.

References

[1] Block i/o layer tracing using blktrace. https://smackerelofopinion.blogspot.com/2009/10/block-io-layer-tracing-using-blktrace.html. Accessed: 2022-10-08.

[2] Pivot3 hci solutions. https://pivot3.com/solutions/. Accessed: 2021-04-12.

[3] Saba Ahmadian, Onur Mutlu, and Hossein Asadi. ECI-Cache: A High-Endurance and Cost-Efficient I/O Caching Scheme for Virtualized Platforms. *Proc. ACM Meas. Anal. Comput. Syst. (POMACS)*, 2(1):9:1–9:34, April 2018.

[4] Saba Ahmadian, Reza Salkhordeh, and Hossein Asadi. LBICA: A Load Balancer for I/O Cache Architectures. In *to appear in Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019.

[5] Muthukaruppan Annamalai, Kaushik Ravichandran, Harish Srinivas, Igor Zinkovsky, Luning Pan, Tony Savor, David Nagle, and Michael Stumm. Sharding the shards: managing datastore locality at scale with akkio. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 445–460, 2018.

[6] Dulcardo Arteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, Dulcardo Zhao, MingArteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, and Ming Zhao. CloudCache: On-Demand Flash Cache Management for Cloud Computing. In *File and Storage Technologies (FAST)*, 2016.

[7] Emmanuel Arzuaga and David R Kaeli. Quantifying load imbalance on virtualized enterprise servers. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 235–242, 2010.

[8] Robert Baldyga. Opencas. https://github.com/open-cas/open-cas-linux/, 2018.

[9] Ann Bednarz. What is hyperconvergence? https://www.networkworld.com/article/3207567/what-is-hyperconvergence.html. Accessed: 2021-04-12.

[10] Blktrace. Blktrace: Block Layer IO Tracing Tool. https://linux.die.net/man/8/blktrace, 2006.

[11] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157, 2011.

[12] Ceph. An Open-source, Distributed Storage System. Last accessed: January 1, 2022.

[13] Hsung-Pin Chang, Yu-Cain He, and Da-Wei Chang. An integrated memory and ssd caching i/o subsystem. In *IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 823–824. IEEE, 2018.

[14] Hsung-Pin Chang, Chien-Neng Liao, and Da-Wei Chang. A page cache management scheme in cloud computing environments. In *IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 974–979. IEEE, 2019.

[15] Geyao Cheng, Lailong Luo, Junxu Xia, Deke Guo, and Yuchen Sun. When deduplication meets migration: An efficient and adaptive strategy in distributed storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[16] Hyung Won Choi, Hukeun Kwak, Andrew Sohn, and Kyusik Chung. Autonomous learning for efficient resource utilization of dynamic vm migration. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 185–194, 2008.

[17] Anita Choudhary, Mahesh Chandra Govil, Girdhari Singh, Lalit K Awasthi, Emmanuel S Pilli, and Divya Kapil. A critical survey of live virtual machine migration techniques. *Journal of Cloud Computing*, 6(1):1–41, 2017.

[18] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286, 2005.

[19] Marcin Copik, Roman Böhringer, Alexandru Calotoiu, and Torsten Hoefler. Fmi: Fast and cheap message passing for serverless functions. In *Proceedings of the 37th International Conference on Supercomputing*, pages 373–385, 2023.

[20] VK Dasarraju. Implementing and managing hyper-converged infrastructure with vmware vsan and azure stack hci. *J Artif Intell Mach Learn & Data Sci 2024*, 2(2):666–670.

[21] Dell EMC. Dell EMC Unity: FAST Technology Overview, 2018.

[22] Dell EMC. DELL EMC VXRACK FLEX Product Overview, 2019. Accessed: Mar. 2019.

[23] Dell EMC. Dell EMC VXRail Appliance TechBook, 2019. Accessed: Mar. 2019.

[24] Dell EMC and Intel. Dell EMC XC Series Advancements and Vision for Customers in the Hyperconverged Era, 2019. Accessed: Mar. 2019.

[25] Tamer Eldeeb, Xincheng Xie, Philip A Bernstein, Asaf Cidon, and Junfeng Yang. Chardonnay: Fast and general datacenter transactions for {On-Disk} databases. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 343–360, 2023.

[26] Ziqi Fan and Dongchul Park. Extending ssd lifespan with comprehensive non-volatile memory-based write buffers. *Journal of Computer Science and Technology*, 34(1):113–132, 2019.

[27] Alireza Farshin, Amir Roozbeh, Gerald Q. Maguire Jr., and Dejan Kostic. Reexamining direct cache access to optimize I/O intensive applications for multi-hundred-gigabit networks. In Ada Gavrilovska and Erez Zadok, editors, *USENIX Annual Technical Conference (ATC), July 15-17*, pages 673–689, 2020.

[28] João B Fernandes, Ítalo AS de Assis, Idalmis Martins, Tiago Barros, and Samuel Xavier-de Souza. Adaptive asynchronous work-stealing for distributed load-balancing in heterogeneous systems. *arXiv preprint arXiv:2401.04494*, 2024.

[29] Alexander Fuerst and Prateek Sharma. Faascache: keeping serverless computing alive with greedy-dual caching. In Tim Sherwood, Emery D. Berger, and Christos Kozyrakis, editors, *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*, pages 386–400. ACM, 2021.

[30] Suket Gakhar and Vishesh Kumar Nirwal. Device Replacement for Hyper-Converged Infrastructure Computing Environments, 2018. US Patent App. 15/684,965.

[31] Xiongzi Ge, Zhichao Cao, David HC Du, Pradeep Ganesan, and Dennis Hahn. Hintstor: A framework to study i/o hints in heterogeneous storage. *ACM Transactions on Storage (TOS)*, 18(2):1–24, 2022.

[32] Najmul Hassan, Kok-Lim Alvin Yau, and Celimuge Wu. Edge computing in 5g: A review. *IEEE Access*, 7:127276–127289, 2019.

[33] HP. HPE Smart Array SR SmartCache, 2018.

[34] HPE. HPE SimpliVity Hyperconverged Infrastructure for VMware vSphere, 2019. Accessed: Mar. 2019.

[35] IDC. Worldwide Converged Systems Revenue, 2018.

[36] IDC. Worldwide Converged Systems Support and Deployment Services Forecast, 2019–2023, 2019.

[37] IDC. Worldwide Enterprise Infrastructure Workloads Forecast, 2021–2025, 2021.

[38] IDC. What Role does Hyperconverged Infrastructure Play in European Companies' Infrastructure Plans?, 2022.

[39] Ilias Iliadis. Reliability evaluation of erasure-coded storage systems with latent errors. *ACM Transactions on Storage*, 19(1):1–47, 2023.

[40] IOSTAT. IOSTAT. https://linux.die.net/man/1/iostat, 2018.

[41] IPTraf. IPTraf. https://github.com/iptraf-ng/iptraf-ng. Accessed: Apr 2022.

[42] J. Kim, H. Roh, and S. Park. Selective I/O Bypass and Load Balancing Method for Write-Through SSD Caching in Big Data Analytics. *IEEE Transactions on Computers (TC)*, 67(4):589–595, April 2018.

[43] Timothy Kim, Sanjith Athlur, Saurabh Kadekodi, Francisco Maturana, Dax Delvira, Arif Merchant, Gregory R Ganger, and KV Rashmi. Morph: Efficient file-lifetime redundancy management for cluster file systems. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 330–346, 2024.

[44] Mostafa Kishani, Saba Ahmadian, and Hossein Asadi. A modeling framework for reliability of erasure codes in ssd arrays. *IEEE Transactions on Computers*, 69(5):649–665, 2019.

[45] Mostafa Kishani and Hossein Asadi. Modeling impact of human errors on the data unavailability and data loss of storage systems. *IEEE Transactions on Reliability*, 67(3):1111–1127, 2018.

[46] Mostafa Kishani, Reza Eftekhari, and Hossein Asadi. Evaluating impact of human errors on the availability of data storage systems. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 314–317. European Design and Automation Association, 2017.

[47] Mostafa Kishani, Mehdi Tahoori, and Hossein Asadi. Dependability analysis of data storage systems in presence of soft errors. *IEEE Transactions on Reliability*, 68(1):201–215, 2019.

[48] Ricardo Koller, Leonardo Marmol, Raju Rangaswami, Swaminathan Sundararaman, Nisha Talagala, and Ming Zhao. Write policies for host-side flash caches. In *11th USENIX Conference on File and Storage Technologies (FAST)*, pages 45–58, 2013.

[49] Ricardo Koller, Ali José Mashtizadeh, and Raju Rangaswami. Centaur: Host-Side SSD Caching for Storage Performance Control. In *IEEE International Conference on Autonomic Computing (ICAC)*, 2015.

[50] Renping Liu, Zhenhua Tan, Linbo Long, Yu Wu, Yujuan Tan, and Duo Liu. Improving fairness for ssd devices through dram over-provisioning cache management. *IEEE Transactions on Parallel and Distributed Systems*, 33(10):2444–2454, 2022.

[51] Weiguang Liu, Jinhua Cui, Tiantian Li, Junwei Liu, and Laurence T Yang. A space-efficient fair cache scheme based on machine learning for nvme ssds. *IEEE Transactions on Parallel and Distributed Systems*, 34(1):383–399, 2022.

[52] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. Distcache: Provable load balancing for large-scale storage systems with distributed caching. In *17th USENIX Conference on File and Storage Technologies (FAST)*, pages 143–157, 2019.

[53] Tian Luo, Siyuan Ma, Rubao Lee, Xiaodong Zhang, Deng Liu, and Li Zhou. S-cave: Effective ssd caching to improve virtual machine storage performance. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, pages 103–112, 2013.

[54] Shagufta Mehnaz and Elisa Bertino. Ghostbuster: A fine-grained approach for anomaly detection in file system accesses. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, page 3–14, New York, NY, USA, 2017. Association for Computing Machinery.

[55] Carlos Melo, Jamilson Dantas, Paulo Maciel, Danilo Mendonça Oliveira, Jean Araujo, Rubens Matos, and Iure Fé. Models for hyper-converged cloud computing infrastructures planning. *International Journal of Grid and Utility Computing*, 11(2):196–208, 2020.

[56] Carlos Melo, Jamilson Dantas, Andre Oliveira, Danilo Oliveira, Iure Fé, Jean Araujo, Rubens Matos, and Paulo Maciel. Availability models for hyper-converged cloud computing infrastructures. In *2018 Annual IEEE International Systems Conference (SysCon)*, pages 1–7. IEEE, 2018.

[57] Santiago Meneses, Edgar Maya, and Carlos Vasquez. Network design defined by software on a hyper-converged infrastructure. case study: Northern technical university fica data center. In *International Conference on Systems and Information Sciences*, pages 272–280. Springer, 2020.

[58] Fei Meng, Li Zhou, Xiaosong Ma, Sandeep Uttamchandani, and Deng Liu. vCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment. In *Proceedings of USENIX Annual Technical Conference (ATC)*, 2014.

[59] Microsoft. Azure Functions Blob Access Trace 2020. https://github.com/Azure/AzurePublicDataset/blob/master/AzureFunctionsBlobDataset2020.md. Accessed: June 2022.

[60] Microsoft. Hyper-V Storage Caching Layers.

[61] Mike Leone. Dell EMC XC Family: Hardware that Matters to the Digitally Transformed Business, 2019. Accessed: Mar. 2019.

[62] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write Off-Loading: Practical Power Management for Enterprise Storage. *ACM Transactions on Storage (TOS)*, 4(3):10, 2008.

[63] NetApp. SSD Cache Feature. https://library.netapp.com, 2017.

[64] John Nicholson and Pete Koehler. A Day in the Life of a VSAN I/O – Diving in to the I/O Flow of vSAN.

[65] Qingpeng Niu, James Dinan, Qingda Lu, and Ponnuswamy Sadayappan. Parda: A fast parallel reuse distance analysis algorithm. In *IEEE 26th International Parallel and Distributed Processing Symposium*, pages 1284–1294, 2012.

[66] Nutanix. *Hyperconverged Infrastructure: The Definitive Guide*. NUTANIX, 2018.

[67] Jong-Geun Park, Jin-Mee Kim, Hoon Choi, and Young-Choon Woo. Virtual machine migration in self-managing virtualized server environments. In *11th International Conference on Advanced Communication Technology*, volume 3, pages 2077–2083, 2009.

[68] J Gregory Pauloski, Valerie Hayot-Sasson, Logan Ward, Alexander Brace, André Bauer, Kyle Chard, and Ian Foster. Object proxy patterns for accelerating distributed applications. *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[69] Proxmox. Proxmox VE Administration Guide. Last accessed: December 26, 2021.

[70] Deepak Puthal, Mohammad S Obaidat, Priyadarsi Nanda, Mukesh Prasad, Saraju P Mohanty, and Albert Y Zomaya. Secure and sustainable load balancing of edge data centers in fog computing. *IEEE Communications Magazine*, 56(5):60–65, 2018.

[71] Francisco Romero, Gohar Irfan Chaudhry, Iñigo Goiri, Pragna Gopa, Paul Batum, Neeraja J. Yadwadkar, Rodrigo Fonseca, Christos Kozyrakis, and Ricardo Bianchini. Faa$t: A transparent auto-scaling cache for serverless

applications. In *ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4, 2021*, pages 122–137, 2021.

[72] Raj Shekar, Matti Vanninen, George Costea, Richard Carter, and Evan Chiu. Hyperconverged Infrastructure Supporting Storage and Compute Capabilities, 2018. US Patent App. 15/431,525.

[73] Artem A Shvidkiy, Anastasia V Spirkina, Anastasiia A Savelieva, and Aleksey V Tarlykov. Evaluation of the impact the hyper-converged infrastructure storage subsystem synchronization on the overall performance. In *2020 12th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 248–252. IEEE, 2020.

[74] Gireesha Udagani Siddappa, Samdeep Nayak, Ravi Kumar Reddy Kottapalli, Srinivas Sampatkumar Hemige, and Shubham Verma. Methods and Apparatus to Manage Compute Resources in a Hyperconverged Infrastructure Computing Environment, 2019. US Patent App. 16/133,742.

[75] SNIA. Microsoft Enterprise Traces, Storage Networking Industry Association IOTTA Repository, 2022.

[76] Gulshan Soni and Mala Kalra. Comparative study of live virtual machine migration techniques in cloud. *International Journal of Computer Applications*, 84(14), 2013.

[77] Sudhir Srinivasan, Devon Reed, and Daniel Cummins. Hybrid Hyper-Converged Infrastructure and Storage Appliance, 2017. US Patent 9,830,082.

[78] Sudhir Srinivasan, Devon Reed, and Daniel Cummins. Hyper-Converged Infrastructure Based on Server Pairs, 2017. US Patent 9,778,865.

[79] Virtunet Systems. Host-side caching software virtucache.

[80] H Reza Taheri, Gary Little, Bhavik Desai, Andrew Bond, Doug Johnson, and Greg Kopczynski. Characterizing the performance and resilience of hci clusters with the tpcx-hci benchmark. In *Performance Evaluation and Benchmarking for the Era of Artificial Intelligence: 10th TPC Technology Conference, TPCTC 2018, Rio de Janeiro, Brazil, August 27–31, 2018, Revised Selected Papers 10*, pages 58–70. Springer, 2019.

[81] Wen-Hong Tian, Min-Xian Xu, Guang-Yao Zhou, Kui Wu, Cheng-Zhong Xu, and Rajkumar Buyya. Prepartition: load balancing approach for virtual machine reservations in a cloud data center. *Journal of Computer Science and Technology*, 38(4):773–792, 2023.

[82] Nguyen Tran, Marcos K Aguilera, and Mahesh Balakrishnan. Online migration for geo-distributed storage systems. In *USENIX Annual Technical Conference*, 2011.

[83] VMware. Vsan disk groups.

[84] VMWare. An Overview of VMWare Virtual SAN Caching Algorithms, 2019.

[85] VMWare. VMWare VSAN 6.7 Technical Overview, 2019.

[86] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *IEEE International Conference on Cloud Computing*, pages 254–265. Springer, 2009.

[87] Carl A Waldspurger, Nohhyun Park, Alexander Garthwaite, and Irfan Ahmad. Efficient MRC Construction with SHARDS. In *File and Storage Technologies (FAST)*, 2015.

[88] Ao Wang, Jingyuan Zhang, Xiaolong Ma, Ali Anwar, Lukas Rupprecht, Dimitrios Skourtis, Vasily Tarasov, Feng Yan, and Yue Cheng. Infinicache: Exploiting ephemeral serverless functions to build a cost-effective memory cache. In *18th USENIX Conference on File and Storage Technologies (FAST), Santa Clara, CA, USA, February 24-27*, pages 267–281, 2020.

[89] Chianing Johnny Wang and BaekGyu Kim. Automotive big data pipeline: Disaggregated hyper-converged infrastructure vs hyper-converged infrastructure. In *IEEE International Conference on Big Data (Big Data)*, pages 1784–1787, 2020.

[90] Minhu Wang, Mingwei Xu, and Jianping Wu. Understanding I/O direct cache access performance for end host networking. *Proc. ACM Meas. Anal. Comput. Syst.*, 6(1):22:1–22:37, 2022.

[91] Zhongyu Wang, Yaheng Song, Erci Xu, Haonan Wu, Guangxun Tong, Shizhuo Sun, Haoran Li, Jincheng Liu, Lijun Ding, Rong Liu, et al. Ransom access memories: Achieving practical ransomware protection in cloud with {DeftPunk}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 687–702, 2024.

[92] Jiatai Wu, Rama Krishna Gurram, and Krishna Kattumadam. Systems and Methods for Policy Driven Storage in a Hyper-Convergence Data Center, 2018. US Patent App. 10/078,465.

[93] Kan Wu, Zhihan Guo, Guanzhou Hu, Kaiwei Tu, Ramnatthan Alagappan, Rathijit Sen, Kwanghyun Park, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. The storage hierarchy is not a hierarchy: Optimizing caching on modern storage devices with orthus. In Marcos K. Aguilera and Gala Yadgar, editors, *19th USENIX Conference on File and Storage Technologies (FAST), February 23-25*, pages 307–323, 2021.

[94] Fei Xu, Fangming Liu, Hai Jin, and Athanasios V Vasilakos. Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. *Proceedings of the IEEE*, 102(1):11–31, 2013.

[95] Fei Zhang, Guangming Liu, Xiaoming Fu, and Ramin Yahyapour. A survey on virtual machine migration: Challenges, techniques, and open issues. *IEEE Communications Surveys & Tutorials*, 20(2):1206–1243, 2018.