applied sciences

MDPI

Article

FirepanIF: High Performance Host-Side Flash Cache Warm-Up Method in Cloud Computing

Hyunchan Park [1], Munkyu Lee [2] and Cheol-Ho Hong [2,*]

[1] Division of Computer Science and Engineering, Jeonbuk National University, Jeonju 54896, Korea; hyunchan.park@jbnu.ac.kr

[2] School of Electrical and Electronics Engineering, Chung-Ang University, Seoul 06974, Korea; dse112@cau.ac.kr

* Correspondence: cheolhohong@cau.ac.kr

Received: 2 January 2020; Accepted: 30 January 2020; Published: 4 February 2020

Abstract: In cloud computing, a shared storage server, which provides a network-attached storage device, is usually used for centralized data management. However, when multiple virtual machines (VMs) concurrently access the storage server through the network, the performance of each VM may decrease due to limited bandwidth. To address this issue, a flash-based storage device such as a solid state drive (SSD) is often employed as a cache in the host server. This host-side flash cache saves remote data, which are frequently accessed by the VM, locally in the cache. However, frequent VM migration in the data center can weaken the effectiveness of a host-side flash cache as the migrated VM needs to warm up its flash cache again on the destination machine. This study proposes Cachemior, Firepan, and FirepanIF for rapid flash-cache migration in cloud computing. Cachemior warms up the flash cache with a data preloading approach using the shared storage server after VM migration. However, it does not achieve a satisfactory level of performance. Firepan and FirepanIF use the source node's flash cache as the data source for flash cache warm-up. They can migrate the flash-cache more quickly than conventional methods as they can avoid storage and network congestion on the shared storage server. Firepan incurs downtime of the VM during flash cache migration for data consistency. FirepanIF minimizes the VM downtime with the invalidation filter, which traces the I/O activity of the migrated VM during flash cache migration in order to invalidate inconsistent cache blocks. We implement and evaluate the three flash cache migration techniques in a realistic virtualized environment. FirepanIF demonstrates that it can improve the performance of the I/O workload by up to 21.87% compared to conventional methods.

Keywords: cloud computing; flash cache migration; flash cache warm-up

1. Introduction

Over the past decade, cloud computing evolved due to stabilized infrastructure based on reliable virtualization technologies. With this advanced infrastructure, cloud vendors provide their customers with the following three types of services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Service as a Service (SaaS). Cloud computing users can now use a centralized cloud platform with powerful and limitless computation and storage resources in a reliable and flexible manner. In addition, owing to open source cloud platforms such as OpenNebula [1] and OpenStack [2], any organization can build a private cloud for its own purposes [3]. As a next step, cloud computing currently seeks to enrich the variety of services and also provide a high quality of services (QoS) to its customers [4–6]. Optimizing cloud infrastructure with multi-objectives to improve customer satisfaction and resource provisioning also has become an important research topic [7,8].

The introduction of cloud computing contributes to flexibility in the use of computation and storage resources, but an overcrowded data center host would fail to provide a high QoS to its tenant. A viable strategy to mitigate this issue is the adoption of virtual machine (VM) migration for load balancing [9], which can move a VM from a congested host to non-crowded one. To reduce the downtime of the VM during migration, a shared storage server, which provides a network-attached storage device, is usually used [10]. Because the disk image file of the VM is located on the shared storage server and accessible via the network, the image does not need to be copied during migration, and only the content of the VM memory will be transferred.

To efficiently exploit the shared storage server, a flash-based storage device such as a solid state drive (SSD) is often employed as a cache per host machine [11]. When multiple VMs concurrently access the storage server through the network, the performance will decrease due to limited network and shared storage bandwidth. A host-side flash cache addresses this situation by saving remote data, which are frequently accessed by the VM, locally in the cache. The local cache architecture does not only reduce data access time with a shortened data path, but also contributes to resolving network and shared storage congestion.

However, the effectiveness of a host-side flash cache can be debilitated by frequent VM migration. In addition to load balancing, VM migration is often performed in a cloud data center for other purposes such as system maintenance and network optimization [12,13]. However, when a VM migration is performed, the warm host cache on the source node will be invalidated, and the migrated VM requires warming up its host-side cache again on the destination machine. This sudden deteriorated cache locality definitely influences the I/O performance of the migrated VM. As the working set of the workload increases, performance degradation due to the cold cache can last for a longer period [14].

This study proposes *Cachemior*, *Firepan*, and *FirepanIF* for our exploration of an ideal host-side flash cache warm-up method for high performance VM migration. Cachemior (*Cache mi*grat*or*) was introduced in our previous work to enable a hot start of the host-side flash cache [15]. During VM migration, Cachemior sends a hot cache list, which contains the addresses of frequently accessed cache blocks, to the destination node. A separate thread in the destination node then warms up the host-side flash cache by accessing the shared storage server. However, Cachemior incurs storage and network congestion on the shared storage server, which impacts the performance of the I/O workloads in the migrated VM negatively. Both Firepan and FirepanIF improve the performance of Cachemior by directly copying all the contents of the flash cache from the source to the destination node. However, Firepan does not allow the migrated VM to resume before the completion of flash cache migration due to a synchronization reason. FirepanIF addresses this issue with an invalidation filter (IF), which traces the I/O activity of the migrated VM during flash cache migration in order to invalidate inconsistent cache blocks. We compare the performance of Cachemior, Firepan, and FirepanIF in Section 4.

The contributions of this paper are summarized as follows:

- We propose Firepan and FirepanIF that use the source node's flash cache as the data source for flash cache warm-up. They can migrate the flash-cache more quickly than conventional methods as they can avoid storage and network congestion on the shared storage server.
- We develop FirepanIF that simultaneously achieves rapid flash-cache migration and minimizes the VM downtime with the invalidation filter, which solves the synchronization problem between the I/O operations generated by the destination VM and flash-cache migration.
- We implement and evaluate the three different flash cache migration techniques in a realistic virtualized environment. FirepanIF demonstrates that it can improve the performance of the I/O workload by up to 21.87% when the size of the flash cache is 20 GB. In addition, the experimental results confirm that our new approach does not generate any negative effects on the neighbor VMs.

The remainder of this paper is structured as follows: In Section 2, we explain the background of the SSD caching software and related work. In Section 3, we elaborate on the detailed design of

Cachemior, Firepan, and FirepanIF. In Section 4, we show the performance evaluation results. Finally, we present our conclusions in Section 5.

2. Background and Related Work

2.1. SSD Caching Software

Our proposed models are based on EnhanceIO [16] for Linux (github.com/stec-inc/EnhanceIO), which is conventional SSD caching software. EnhanceIO is implemented as a loadable kernel module, which can easily extend the Linux kernel for creating, configuring, and analyzing SSD-based cached environments. Figure 1 shows a Linux system with EnhanceIO. EnhanceIO accelerates repeated I/O requests by caching disk blocks in the host-side flash cache. The I/O requests from user applications are sent to EnhanceIO in the kernel space, and EnhanceIO accesses the host-side SSD cache on a cache hit or the shared storage on a miss. EnhanceIO also supports various cache replacement policies including Least-Recently Used (LRU); First-In, First-Out (FIFO); and Random. We choose to use LRU in this study because it generally performs better than FIFO or Random. EnhanceIO provides three cache write policies: read-only, write-through, and write-back. The read-only method inserts new cache items only when read requests are performed. If a write request is generated, the corresponding block in the flash cache is invalidated, and the request is redirected to the shared storage server. The write-through approach will write a data item simultaneously into both the cache and the storage server upon a write request. The write-back method first writes a data item into the cache upon a write request, and later the cached item will be moved to the storage server when the I/O system becomes idle. This can speed up write requests. In this study, we select the read-only method because write-through and write-back are not preferred in virtualized environments due to reliability issues, as discussed in Section 4.

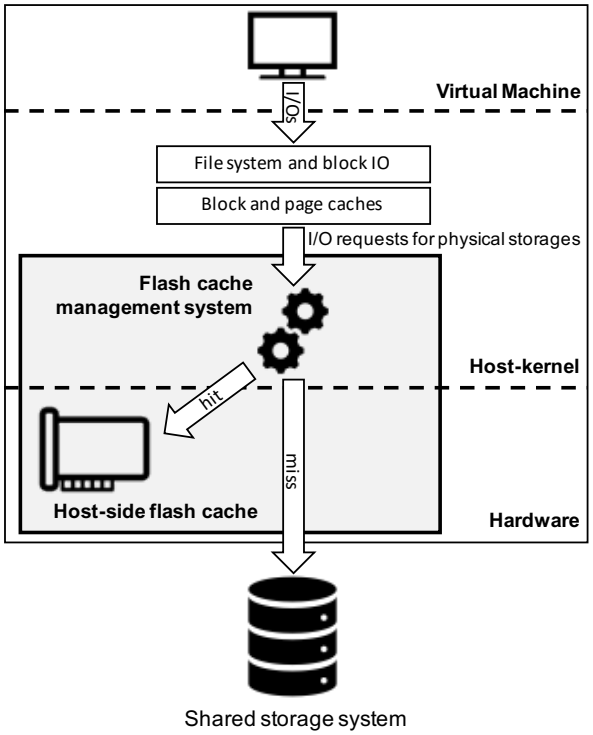


Figure 1. Linux System with EnhanceIO.

2.2. Related Work

Bonfire and Successor are representative examples for page cache warm-up when a VM is migrated [17,18]. They focused on warming up the page cache in the physical memory rather than

host-side flash caches, but the introduced mechanisms can also be applied to flash caches without major changes. Bonfire demonstrated that the migrated VM suffers from its cold page cache and proposes a rapid page cache warm-up technique with prefetching [17]. After VM migration, Bonfire transfers the list of cache entries from the source to the destination host. Then, Bonfire starts to warm up the page cache by loading the entries on the list from the shared storage system. Successor proposes an advanced prefetching technique that focuses on the start point of the page cache warm-up [18]. Successor performs page cache prefetching and VM migration in parallel so that the migrated VM can obtain the benefit of its hottest cache earlier than in the case of Bonfire. However, the parallel execution of VM migration and page cache warm-up leads to a cache inconsistency problem. When the migrated VM modifies the data on the page cache, following which the cache warm-up thread tries to fetch the data of the same address from the shared storage server, the latest data will be overwritten. To address the problem, Successor tracks dirty pages on the destination host and does not prefetch these pages.

Compared to Bonfire and Successor, the main difference with FirepanIF lies in the data source. Instead of using the shared storage, FirepanIF migrates all flash cache contents from the source to the destination host. Cache migration generally issues many I/O requests for transferring large bulks of data, and this consumes huge network bandwidth. As the network path between the source and destination hosts is less crowded compared to the network path to the shared storage, FirepanIF is expected to finish the warm-up process earlier than Bonfire and Successor. In our proposed models, Cachemior also uses the shared storage as the data source and shows worse performance than FirepanIF. FirepanIF also performs VM and cache migration in parallel, similar to Successor. FirepanIF uses the invalidation filter to solve the cache inconsistency problem.

Mortar and FVP also provide a flash cache warm-up technique with cache pooling [19,20]. Instead of transferring the data from the shared storage, upon a respective cache miss, cache pooling fetches the corresponding data from the flash cache of the source host. This technique is also called remote read or remote paging. The cache pooling system cannot warm up the cache on the destination host faster, but it avoids congestion on the shared storage. The disadvantage of cache pooling is that the flash cache on the source node cannot be freed and reallocated to another VM immediately. In addition, the access latency can be slowed down according to the network status.

Several studies addressed related issues with flash-based host-side caches. First, studies on FlashCache, Mercury, and vSphere flash read cache [21–23] demonstrated that the adoption of flash caches in the data center is effective. Several studies about flash cache management in terms of space management and cache policies were also conducted. S-Cave, vCacheShare, and CloudCache were proposed for efficient and flexible use of flash devices [11,24,25]. Because flash-based storage devices are expensive and present limited resources, the efficient allocation of flash cache space is the key factor that differentiates performance among multiple VMs. In addition, CacheDedup introduced an in-line deduplication technique to use the flash cache efficiently and reduce the number of cache insertions in order to solve the lifetime issue of flash-based devices.

Cache write policies are also important for performance improvement of flash caches. Byan et al. [22], Koller et al. [26], Holland et al. [27], and Qin et al. [28] analyzed several cache policies for flash caches with various workloads. They pointed out that write caching is very effective for providing predictable and high performance on flash caches. In addition, FVP proposed a fault-tolerant write caching technique by replicating write requests to another host and providing a mechanism to access the remote flash cache [20]. The flash cache migration techniques proposed in this study do not depend on such space management techniques and write cache policies.

3. Design

In this section, we introduce the designs of Cachemior, Firepan, and FirepanIF, which are our proposed models for efficient host-side flash cache warm-up for high performance VM migration. The proposed models are based on EnhanceIO [16]. However, our design is not specific to EnhanceIO and can be applied to other SSD caching frameworks such as dm-cache [29] as well.

3.1. Cachemior

Our previous work devised Cachemior (*Cache migrator*) to enable a hot start of the host-side flash cache [15]. As depicted in Figure 2, Cachemior consists of the following three components in each host server: hot cache monitor, hot cache messenger, and cache warmer.

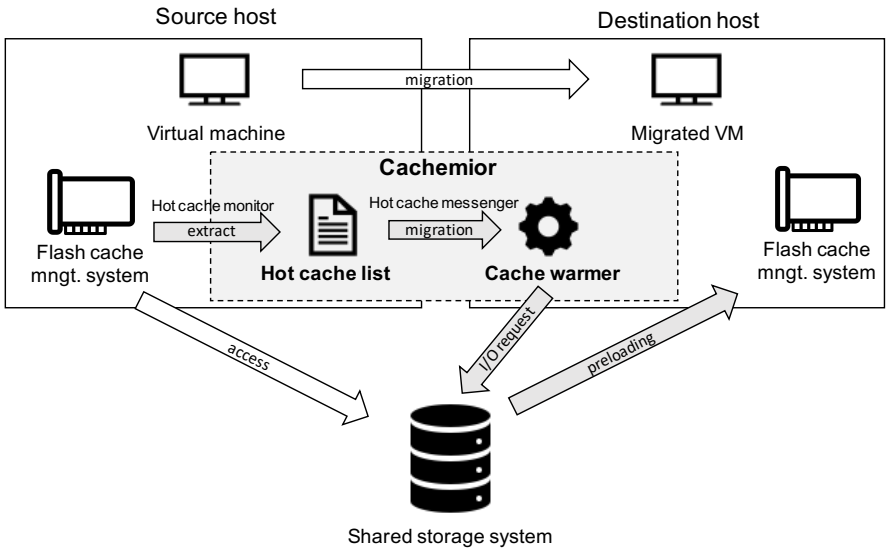


Figure 2. Components of Cachemior: Hot cache monitor, hot cache messenger, and cache warmer.

The hot cache monitor is implemented in the flash cache management system (i.e., EnhanceIO) and analyzes the host-side flash cache in the host machine. It identifies which cache block is frequently used or less recently used. It maintains a sorted list (i.e., the hot cache list in Figure 2) where frequently accessed cache blocks are placed toward the head of the list. Each cache block is associated with the logical block address of the shared storage server. As the list only includes the logical block addresses, the list size is relatively small compared to the cache size. For example, when the size of the host-side flash cache is 20 GB, the maximum size of the hot cache list is under 20 MB.

The hot cache messenger is in charge of transmission of the hot cache list between host servers. The messenger in the source node sends the sorted list to the destination node when the migration occurs, and the messenger in the destination node receives it. After receiving the list, the cache warmer in the destination node reconstructs or warms up the host-side flash cache by accessing the shared storage server in the background with a separate thread. As the cache warmer iterates the list from the head, frequently accessed blocks are restored earlier in time than the less recently used ones. If a target item in the list is already placed in the cache by the activity of the migrated VM, the corresponding block does not need to be accessed again. We call this procedure as preloading because the data will be loaded into the flash cache before they are accessed by the VM.

Limitations: After VM migration, Cachemior starts to warm up the host-side flash cache in the destination node. The separate thread in the cache warmer continuously copies cache items from the shared storage server using the hot cache list. However, simultaneously, the VM starts to access the shared storage server for executing its I/O-intensive workloads, and this situation may cause storage and network congestion on the shared storage server. As a result, the host-side flash cache would warm up slowly, and the performance of the I/O workloads on the VM may also deteriorate.

3.2. Firepan

To mitigate the limitation of Cachemior, we develop Firepan, which migrates the hot cache items in the flash cache management system of the source node to that of the destination node during VM migration. The basic mechanism of Firepan is simple. As depicted in Figure 3, Firepan copies all the

flash cache contents from the source's flash cache management system to the destination node during VM migration, which is similar to a memory migration method that copies all the memory pages from the source to the destination node. The VM will be stopped until the flash cache migration procedure is completed. When the VM migration is resumed, the VM can benefit from the same cache contents copied by Firepan. This enables the VM to maintain high I/O performance because it will not cause storage and network congestion on the shared storage server, as in the case of Cachemior.

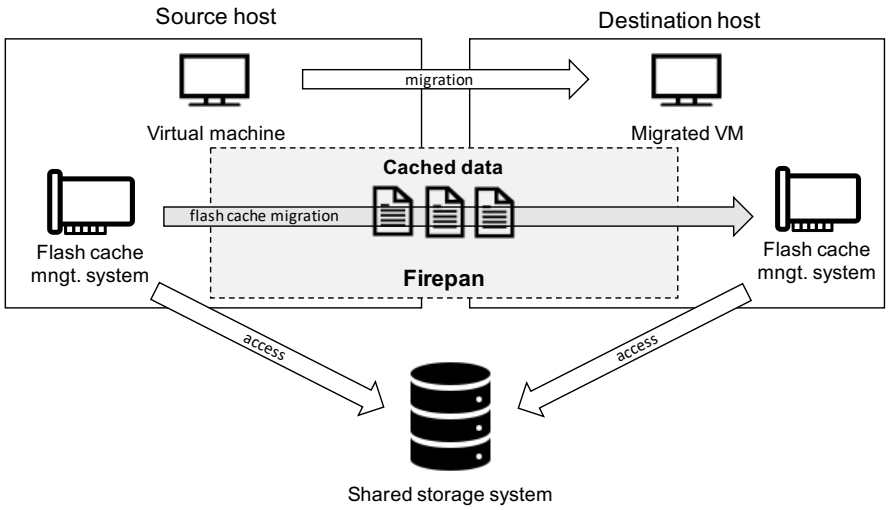


Figure 3. Flash cache migration procedure by Firepan.

Limitations: Although the I/O workloads on the migrated VM can exhibit high performance after the VM is resumed, Firepan does not allow the migrated VM to resume before the completion of flash cache migration. If the VM resumes before cache migration, a consistency problem will arise. Let us suppose that a cache block is not copied by Firepan yet, and the early resumed VM accesses the same cache block for updating the data. A compulsory cache miss will be then generated, and a new cache block will be inserted with new information. However, Firepan would overwrite the same cache block at a later time with old information. Therefore, the VM should be stopped for consistency until the cache migration is completed. As flash storage devices are becoming cheaper, the size of the flash cache will expectedly increase. Then, the required time for copying cache items will also increase. The lengthened VM suspension time by Firepan will eventually lead to a poor user experience.

3.3. FirepanIF

To overcome the limitation of Firepan, FirepanIF combines Firepan and with an IF. Compared with Firepan, FirepanIF resumes the migrated VM after memory migration even if the flash cache migration procedure is in progress. At this moment, the host-side flash cache is not given to the migrated VM; the migrated VM directly accesses the shared storage before the flash cache is entirely migrated by FirepanIF. The IF then traces all the I/O requests from the migrated VM and records only write accesses to the shared storage server in the invalidation table. After all the cache blocks are copied, the host-side flash cache is attached to the migrated VM.

From this point, the migrated VM is able to use the host-side flash cache. However, some cache blocks in the host-side flash cache may remain inconsistent with the corresponding storage blocks because the VM may have issued write requests to the storage blocks during flash cache migration. FirepanIF can identify which blocks are inconsistent by looking up the invalidation table. Therefore, when the VM issues an I/O request, the IF examines the invalidation table in order to know whether the request will access an inconsistent block in the host cache, as depicted in Figure 4. As the invalidation table is implemented as a hash table with separate chaining, the computational complexity of searching the inconsistent block is $O(logN)$, where N is several entries in the hash table. If the request is found in

the invalidation table, the IF redirects the request to the shared storage server. Then, the corresponding cache block will be invalidated by the filter so that a new cache entry can be inserted into the flash cache. With this mechanism, FirepanIF solves the consistency problem incurred by Firepan.

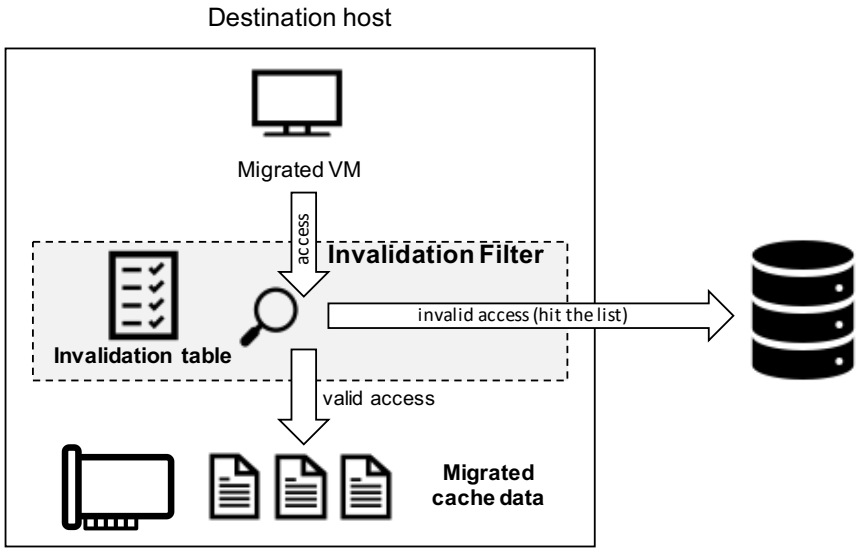


Figure 4. Invalidation procedure by FirepanIF.

Algorithms 1 and 2 describe the detailed behavior of the invalidation filter and the management of the I/O requests during cache migration. *dispatch_with_IF*() is the main function that processes the I/O requests in the flash cache management system. During cache migration, every request will be served by the shared storage. If the request is a write request, the function inserts the physical address of the data in a hash chain. After the migration is completed, every request will be inspected whether it is in the hash table. If it is, the hash entry is removed, and the request is dispatched to either the flash cache for writing or the shared storage for reading. This is the key behavior of invalidation filtering to keep the integrity between old data in the migrated cache and updated data in the shared storage during cache migration. The *dispatch_to_flash_cache*() and *dispatch_to_shared_storage*() functions work as an ordinary cache management system. The function looks up its cache for a read request and serves it on a cache hit or dispatches the request to the shared storage. The *dispatch_to_shared_storage*() function sends the request to the shared storage while updating the flash cache entries.

FirepanIF can reduce the downtime of the VM during migration compared with Firepan. FirepanIF resumes the VM while the cache items are copied from the source to the destination node. As the VM suspension time becomes much shorter, the user can continue to perform the workloads inside the VM. However, during cache migration, the VM will access the shared storage server instead of the flash cache. This can decrease the performance of the I/O workload in the VM for a moment. Despite this decline in performance, we reveal that FirepanIF significantly outperforms Firepan because the I/O workloads can be executed during cache migration, as discussed in Section 4.

Algorithm 1: Dispatch with invalidation filtering.

Input: An I/O request (*req*) with an operation mode (*op*), physical block address (*pba*), and data to write or to be filled by read (*data*).
Output: The length of processed data. Zero on error.
Data: *hash_size* is the size of the hash table.;
IF_chains[*hash_size*] is an array of hash chains. Each chain has no entry at the start.;
IF_entries is several entries in the hash table. Initialized as 0;
Function *dispatch_with_IF*(*req*)**:**

```
if the flash cache migration is not done then
    if req.op = write then
        chain ← IF_chains[req.pba%hash_size];
        entry ← allocate a new entry at the end of the chain;
        entry.pba ← req.pba;
        Increase IF_entries by one;
    end
    return dispatch_to_shared_storage(req);
end
else if IF_entries ≠ 0 then
    chain ← IF_chains[req.pba%hash_size];
    for every entry in the chain do
        entry = next entry of the chain;
        if req.pba = entry.pba then
            free and remove the entry from the chain;
            decrease IF_entries by one;
            if req.op = read then
                return dispatch_to_shared_storage(req);
            end
            else
                return dispatch_to_flash_cache(req);
            end
        end
    end
    return dispatch_to_flash_cache(req);
    else
        return dispatch_to_flash_cache(req);
    end
end
```

Algorithm 2: Dispatch functions for the flash cache and the shared storage.

```
Function dispatch_to_flash_cache(req):
    if req.op = read then
        if req.pba is found in the flash cache then
            data ← the cache entry correspond to req.pba;
            return the length of data
        end
        else
            return dispatch_to_shared_storage(req)
        end
    end
    else
        return dispatch_to_shared_storage(req)
    end
Function dispatch_to_shared_storage(req):
    if req.op = read then
        data ← dispatch req for the shared storage;
        if flash cache is available then
            update the flash cache with req.pba and data;
        end
        return the length of data
    end
    else
        if flash cache is available then
            update the flash cache with req.pba and req.data;
        end
        dispatch req for the shared storage;
        return the length of req.data
    end
```

4. Evaluation

In this section, we evaluate Cachemior, Firepan, and FirepanIF with realistic experiments. After describing the test equipment and settings, we present the evaluation results.

4.1. Experimental Setup

We use two computing nodes and one shared storage node. Each of the two computing nodes equip six physical cores, 8 GB main memory, a 250 GB SATA SSD for a host operating system, and a 512 GB NVMe SSD for a host cache. The shared storage node equips four physical cores, 8 GB main memory, a 120 GB SATA SSD for an operating system, and a 480 GB NVMe SSD, which is used for the shared storage. Linux kernel 4.15.0-43 is used for all the nodes, and Kernel-base Virtual Machine (KVM) is used to virtualize the computation and storage resources. Three nodes are connected via 10 Gigabit Ethernet.

Each VM has four virtual cores, 1 GB memory, and 50 GB storage. For the host-side flash cache, each VM is given 10 GB of NVMe SSD. The host-side cache size is 20% of the VM storage that that is enough to store the operating system and workload data of the VM. We configure the workload data size as 20 GB for all the experiments so that the host-side flash cache can store 50% of the VM's workload data.

EnhanceIO is used for flash cache management. As explained in Section 2.1, EnhanceIO provides three cache write policies: read-only, write-through, and write-back. We use the read-only policy for

our experiments because write caching is not preferred in the virtualized environment due to the reliability issue. When the flash cache management subsystem in the host crashes, the data of the VM can be lost with write caching. Please note that although we apply the read-only policy, our models do not depend on the cache policy.

We use flexible I/O tester (FIO) as a benchmark suite in our experiments [30]. To create a realistic workload with spatial locality, we configure the FIO benchmark as follows: four threads issue I/O requests concurrently, and each thread issues random reads and writes with an 8:2 ratio for an exclusive 5 GB file. The requests are directly delivered to the storage device, and the maximum number of concurrently issued requests is 32 per thread. Because the FIO always tries to issue as many requests as possible, 128 requests from the four threads wait in the storage device simultaneously while the benchmark program is running.

In addition, we carefully configure the randomness of the FIO. Because the default configuration of the FIO uses the same number for a random seed across executions, the FIO generates the same series of requests for each execution. The behavior is not similar to that of a real file system. Therefore, we configure the FIO to use fully random numbers for the target addresses of the I/O requests. Moreover, we configure the FIO such that 80% of the requests are issued for the first 20% of addresses in the target file to simulate spatial locality. We provide the full configuration of the FIO workload in Figure 5 to help the reproduction of our experiments.

```
[global]
name=workload
rw=randrw
rwmixread=80
rwmixwrite=20
direct=1
time_based=1
norandommap
random_distribution=zoned:50/5:30/15:20/80
ioengine=libaio
iodepth=32
size=5G
randrepeat=0

[file1]
bs=4k
[file2]
bs=4k
[file3]
bs=4k
[file4]
bs=4k
```

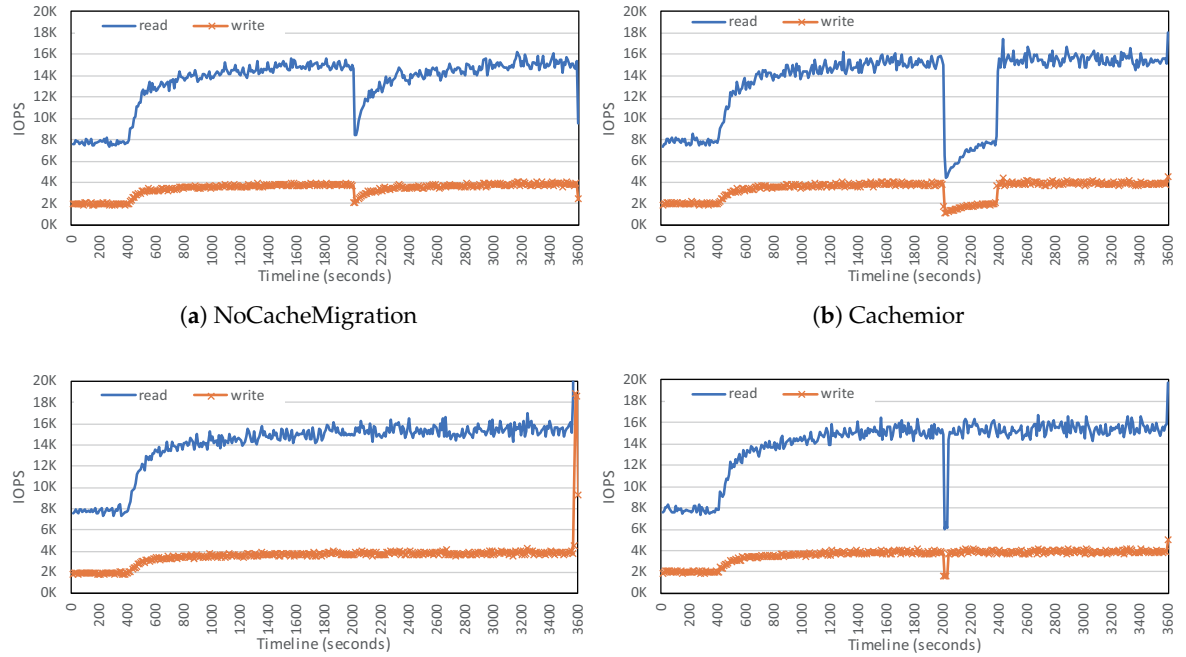
Figure 5. Configuration for the FIO workload.

4.2. Effectiveness of Firepan and FirepanIF

The first experiment shows the effectiveness of Firepan and FirepanIF with a realistic I/O workload compared to Cachemior. Firepan and FirepanIF directly copy the cache contents from the source to the destination node during migration, whereas Cachemior only transfers the hot cache list to the cache warmer in the destination node, as explained in Section 3.1. At the beginning of the experiment, four VMs run on the source host, and three VMs, on the destination host. Each VM runs the FIO for 3600 s. Then, we migrate a VM from the source to the destination host. We call this VM the target VM. The six other VMs run the FIO workload with a flash cache for the entire experiment.

The experimental procedure is as follows: At first, the target VM runs a workload without a flash cache for the first 400 s. Then, we attach a flash cache to the VM. After 1600 s, the target VM is migrated to the destination host, and then, we immediately allocate a flash cache to the VM. Next, the VM continues running a workload during the next 1600 s.

We ran this experiment with four methods: (1) No cache migration (NoCacheMigration), and flash cache migration using (2) Cachemior, (3) Firepan, and (4) FirepanIF. Figure 6 shows the results of the I/O operations per second (IOPS) with the FIO benchmark.



(**a**) NoCacheMigration (**b**) Cachemior

(**c**) Firepan (**d**) FirepanIF

Figure 6. Performance of the target VM with NoCacheMigration, Cachemior, Firepan, and FirepanIF.

First, we explain the result of NoCacheMigration shown in Figure 6a. For the first 400 s, FIO performs approximately 8K IOPS for read operations without a flash cache, which is akin to the baseline performance. After the flash cache is provided in the source node, the cache is warmed up for the next 800 s approximately. After 1400 s of running, the IOPS result seems to stabilize, and the flash cache hit ratio at this time is approximately 60%. The average IOPS value between 1400 s and the migration moment is more than 15K, which exceeds a 90% increment from the baseline performance. At 2000 s, the target VM is migrated, and this operation takes approximately 3 s. After the target VM is migrated, the flash cache on another host should be warmed up again. Thus, the IOPS value dramatically drops after the migration, and thereafter, the performance increases again when the flash cache is attached to the migrated VM. It also takes approximately 800 s for the benchmark to provide the highest performance. The result of the write operation is similar to that of the read case.

Second, Figure 6b shows the result with Cachemior. Until VM migration at 2000 s, the result is similar to that seen in Figure 6a because there is no difference in the procedure. After the VM migration, the cache warmer starts to warm up the flash cache at the destination host. The flash cache migration continuously loads the hot data from the shared storage by using the cache metadata transferred from the flash cache management system running on the source host. In this experiment, we use one asynchronous I/O thread for data preloading. The thread creates up to 128 requests concurrently for the shared storage. Thus, after 2000 s, the FIO and warm-up thread compete for the shared storage until the thread ends at 2360 s. During the preloading period, the target VM suffers from low performance because of the preloading thread's activity. This is the main drawback of the

preloading approach. When we analyze the raw data, the VM and preloading thread compete with each other for the bandwidth allocated for the shared storage. After the warm-up ends, the IOPS value immediately increases to above 15K, which means that the flash cache migration was successfully completed. As shown in Figure 6b, however, the performance loss by data preloading is too large and outweighs the benefit. The average IOPS values after VM migration are 13,946 and 13,672 for NoCacheMigration and Cachemior, respectively. Thus, flash cache migration with Cachemior does not provide any benefit.

Next, Figure 6c shows the result of Firepan. Compared to the previous experiments, the IOPS value does not drop during the migration. However, this is only an illusion for the user-level application. As the target VM was paused during VM and flash cache migration and starts with its copied flash cache, the measured IOPS value in the VM shows high performance, as if no migration occurred. In the real world, the user suffers due to the VM and flash cache migration, which takes approximately 3 and 30 s, respectively. Thus, we further investigate the real effectiveness of Firepan by uncovering the I/O usages of the host machines.

We trace the bandwidth (MB/s) of the read requests issued by the target VM on the two host systems, and show the results before and after VM migration in Figure 7. We use the bandwidth metric instead of the IOPS in this analysis because I/O requests from the VM are fragmented in the host system, which results in inaccurate measurement. The bandwidth is the sum of two I/O streams for the shared storage and flash cache device. The results of NoCacheMigration and Cachemior show similar trends. The performance drops after the migration and then increases slowly while the flash cache is warmed up. The results are similar to those in Figure 6a,b. In the case of Firepan, the bandwidth was zero during flash cache migration, which takes approximately 30 s. Then, the bandwidth immediately increases to the same level as that before the VM migration. We exclude the bandwidth used to migrate the flash cache in the results of Firepan and FirepanIF.

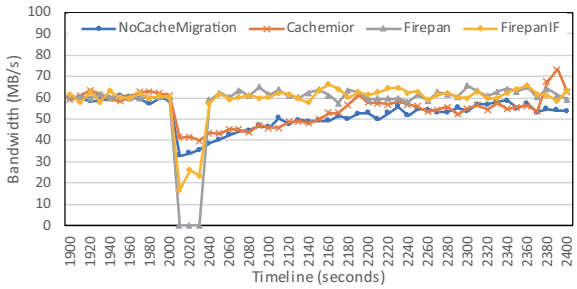


Figure 7. Bandwidth before and after VM migration with NoCacheMigration, Cachemior, Firepan, and FirepanIF.

We provide more detailed results in Table 1. When we compare the average bandwidth for 400 s after VM migration, Firepan provides 13.49% more bandwidth for the target VM despite the suspension for 30 s compared to NoCacheMigration. Even if we compare 800 s after VM migration, which is sufficient time for the cache warm up in the other methods, Firepan provides 9.93% and 2.73% more bandwidth compared to NoCacheMigration and Cachemior, respectively. The benefit is derived from the immediate restoration of the entire cache state while the other methods gradually restore the cache state.

Finally, the result for FirepanIF is shown in Figure 6d. After the VM migration, the target VM is executed immediately without its host-side flash cache while the flash cache migration is in progress. Thus, the performance drops for approximately 30 s. The average IOPS value during this period is approximately 6K, which is lower than the performance of NoCacheMigration immediately after the VM migration (i.e., 8K) in Figure 6a. As flash cache migration requires additional bandwidth for the network and host-side flash cache, the four FIO threads on the target VM are affected. After the flash cache migration is completed, the performance immediately recovers to the same level as that when

the flash cache is fully warmed up. As shown in Figure 7 and Table 1, FirepanIF provides 16.68% more bandwidth for the target VM after 400 s following the VM migration and 12.00% after 800 s. This result shows that FirepanIF is the most effective method for host-side flash cache warm-up among the four methods.

Table 1. Average bandwidth between 400 and 800 s after VM migration.

	NoCache Migration (Baseline)	Cachemior	Firepan	FirepanIF
2000s	41.89	45.25 (+8.02%)	43.11 (+2.93%)	48.67 (+16.18%)
2100s	49.58	51.16 (+3.19%)	61.69 (+24.44%)	61.88 (+24.82%)
2200s	53.48	55.90 (+4.53%)	60.13 (+12.44%)	62.18 (+16.26%)
2300s	55.88	58.46 (+4.62%)	62.98 (+12.71%)	61.59 (+10.22%)
avg. (first 400 s)	50.21	52.69 (+4.94%)	56.98 (+13.49%)	58.58 (+16.68%)
2400s	56.09	62.21 (+10.91%)	61.23 (+9.17%)	60.72 (+8.26%)
2500s	57.28	61.66 (+7.65%)	61.04 (+6.57%)	60.42 (+5.48%)
2600s	56.99	63.26 (+11.00%)	62.36 (+9.41%)	61.39 (+7.73%)
2700s	58.57	61.91 (+5.70%)	59.89 (+2.25%)	64.47 (+10.07%)
avg. (total 800 s)	53.72	57.48 (+7.00%)	59.05 (+9.93%)	60.16 (+12.00%)

4.3. Neighborhood Effect

In addition, we examine the neighborhood effect of Cachemior, Firepan, and FirepanIF. After providing performance models to theoretically analyze the neighborhood effect, we present the experimental results in this section.

4.3.1. Performance Model for Neighborhood Effect

At first, we provide a performance model that represents each VM's I/O performance related to the number of co-running VMs. In the cloud system, VMs share the following resources that influence the I/O performance: a flash cache, shared storage, and network. These resources are fairly allocated to the VMs in general cloud systems. In addition, there is another important consideration about I/O performance when VMs share the flash-based storage: partitioning. Usually, each VM is allocated limited space of the storage device with a limited partition. Because an SSD cannot maximize the internal parallelism among multiple flash-chips with the limited partition, each VM receives restricted performance from the allocated space. For example, we allocate 50 GB space of the 480 GB shared storage and 10 GB of the 512 GB flash cache for each VM. A 10 GB partition of the SSD cannot provide the same level of performance that the entire 512 GB SSD can provide. Therefore, the first reason that limits the I/O performance of each VM is not sharing of the storage with other VMs, but partitioning of the storage. In other words, the performance of each VM is isolated from the neighbor VM by its partition.

Equation (1) presents the storage performance in IOPS that can be provided for the kth VM, VM_i^k, on the ith node based on the above explanation. S_i^k is the maximum performance of the shared storage partition that is allocated for VM^k on the ith node. F_i^k is the maximum performance of the flash cache partition, and H_i^k is a hit ratio for the cache of VM^k on the ith node.

$$IOPS_i^k = H_i^k F_i^k + (1 - H_i^k) S_i^k \tag{1}$$

Equation (2) revises Equation (1) taking into consideration of sharing of the storage and the cache with other VMs. When the overall performance provided by all the partitions exceeds the maximum performance of the device, the storage performance will be throttled down. S and F_i are the maximum performance provided by the shared storage system and the flash cache on the ith node. E_i is the

number of VMs running on the ith node, and the total number of host nodes is n. We assume that every VM fairly shares the total bandwidth of the shared storage system and the flash cache.

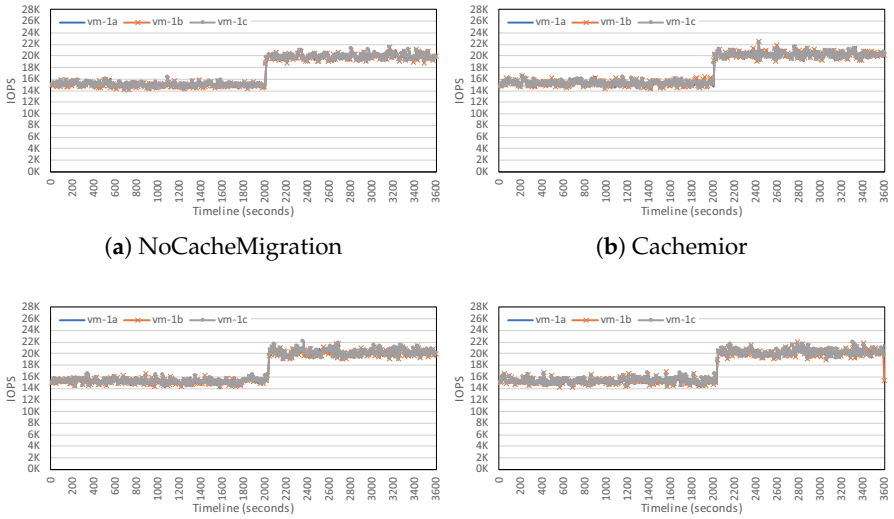
$$IOPS_i^k = \max\left\{\frac{F_i}{E_i}, H_i^k F_i^k\right\} + \max\left\{\frac{S}{\sum_{i=0}^{n} E_i}, (1 - H_i^k) S_i^k\right\} \quad (2)$$

The last factor is the network bandwidth, which can throttle the storage performance. Equation (3) represents the complete performance model for VM_i^k, where N_i is the maximum network bandwidth between the ith node and the shared storage system, and B_i is the average size of I/O requests in bytes from the ith node.

$$IOPS_i^k = \max\left\{\frac{F_i}{E_i}, H_i^k F_i^k\right\} + \max\left\{\frac{S}{\frac{N_i}{B_i}, \sum_{i=0}^{n} E_i}, (1 - H_i^k) S_i^k\right\} \quad (3)$$

4.3.2. Experimental Result

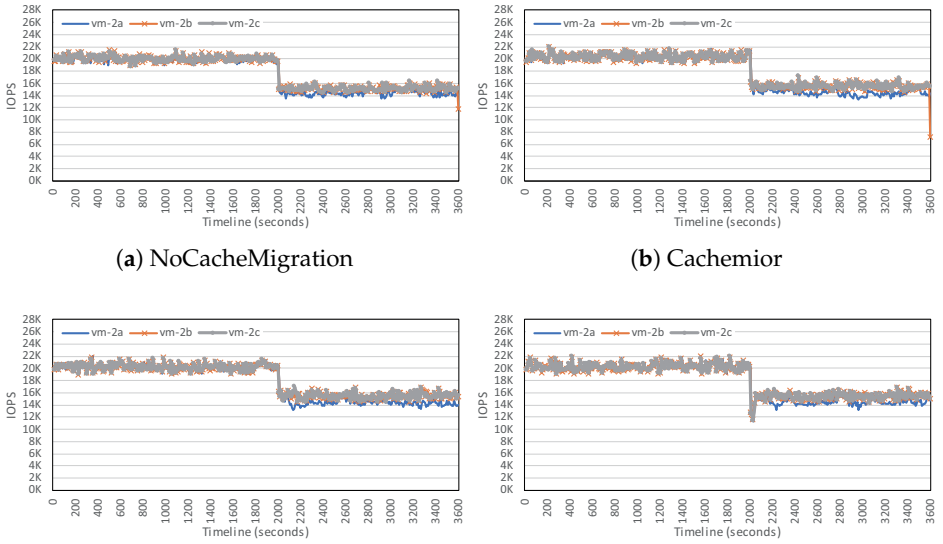
Second, we present the experimental result and analysis. Figures 8 and 9 show the IOPS results of six other VMs on the source and destination hosts during the experiment in Section 4.2. Each VM runs the FIO with the same workload and 10 GB of a host-side flash cache. Before the experiment, we execute the VMs for 30 min to warm up the flash caches. As shown in Figure 8a, when four VMs are running on the source host, the IOPS results are approximately 15K for each VM until 2000 s. Although a flash cache is attached to the target VM after the first 400 s, as explained in Section 4.2, the performance of the three VMs remains steady whether the target VM shares the NVMe SSD or not. This is because the NVMe SSD for flash caches has already been providing the maximum performance for the three 10 GB partitions of the three VMs as explained in Section 4.3.1. After the target VM is migrated, the IOPS results of the three VMs increase to approximately 20K because of the competition for the shared storage drops. The IOPS changes in the two hosts are similar for every case. When three and four VMs compete, the average IOPS values are 20K and 15K, respectively.



(**a**) NoCacheMigration (**b**) Cachemior

(**c**) Firepan (**d**) FirepanIF

Figure 8. Performance of neighbor VMs on the source host with NoCacheMigration, Cachemior, Firepan, and FirepanIF.



(**a**) NoCacheMigration (**b**) Cachemior

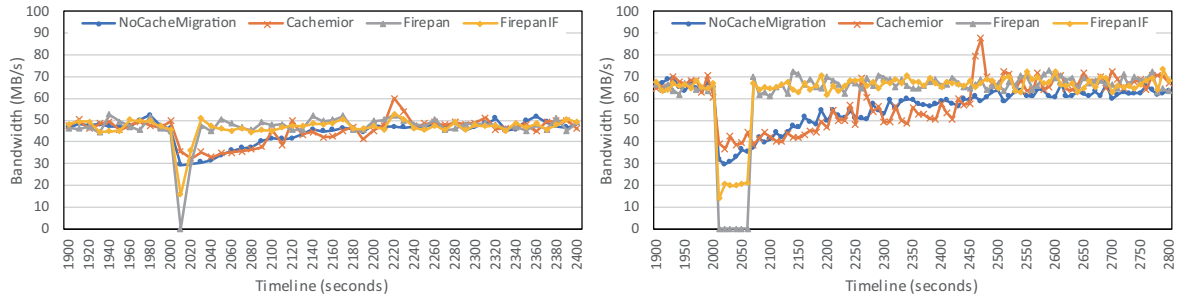
(**c**) Firepan (**d**) FirepanIF

Figure 9. Performance of neighbor VMs on the destination host with NoCacheMigration, Cachemior, Firepan, and FirepanIF.

4.4. Effect of Flash Cache Capacity

In this section, we investigate the performance impacts of Firepan and FirepanIF with various flash cache capacities in more detail. Firepan and FirepanIF copy cache items from the source node, and the required time for copying may impact the I/O performance of the target VM. To reflect this situation, we conduct additional experiments with 5 and 20 GB capacities of flash caches. The procedure and workload of the experiment are the same as in the previous experiments.

We provide the bandwidth results before and after the VM migration for the 5 and 20 GB flash caches in Figure 10a,b, respectively. Please note that the scales of the x-axes differ in the figures. The results are similar to that of the previous experiment depicted in Figure 7. NoCacheMigration and Cachemior show a gradual performance increase after the VM migration. For the 20 GB flash cache, approximately more than 400 s are required to reach the highest bandwidth. Firepan and FirepanIF require 15 and 60 s, respectively, for flash cache migration, which is similar to the result in Figure 7. After these periods, the bandwidth is immediately restored to the level before the VM migration.



(**a**) With a 5 GB flash cache (**b**) With a 20 GB flash cache

Figure 10. Bandwidths before and after VM migration with various flash cache capacities.

Table 2 and 3 summarize the average bandwidth values during 400 and 800 s after VM migration, respectively. Firepan and FirepanIF provide better performance for every flash cache capacity. With the 20 GB flash cache, FirepanIF exhibits 59.41 MB/s as the average bandwidth for 400 s, which translates into an improvement of 21.87% over NoCacheMigration.

Table 2. Average bandwidth during 400 s after VM migration.

	NoCache Migration (Baseline)	Cachemior	Firepan	FirepanIF
5 GB	43.62	44.55 (+2.14%)	46.23 (+5.99%)	46.23 (+5.97%)
10 GB	50.21	52.69 (+4.94%)	56.98(+13.49%)	58.58 (+16.68%)
20 GB	48.75	48.02 (-1.51%)	56.61 (+16.11%)	59.41 (+21.87%)

Table 3. Average bandwidth during 800 s after VM migration.

	NoCache Migration (Baseline)	Cachemior	Firepan	FirepanIF
5 GB	45.64	46.36 (+1.57%)	47.17 (+3.34%)	46.97 (+2.92%)
10 GB	53.72	57.48 (+7.00%)	59.05 (+9.93%)	60.16 (+12.00%)
20 GB	54.86	57.93 (+5.60%)	61.89 (+12.80%)	62.79 (+14.45%)

The results show that the benefits of Firepan and FirepanIF increase with flash cache capacities. As the size increases, NoCacheMigration and Cachemior require a much longer time to fully warm up the increased host-side flash cache, as shown in Figure 10b. In the case of Firepan or FirepanIF, although flash cache migration takes more time than that in the case of a small flash cache, the migration time is much smaller than the warm up times of NoCacheMigration and Cachemior. Therefore, the overall performance of Firepan and FirepanIF shows an improvement in this experiment.

5. Conclusions

This study developed and proposed the use of Cachemior, Firepan, and FirepanIF to discover an efficient method for host-side flash cache warm-up in order to enable high performance VM migration. Cachemior warms up the host-side flash cache in the destination node with a separate thread after VM migration. However, it causes storage and network congestion on the shared storage server, thus achieving a marginal performance gain. As Firepan copies all the cache contents from the source to the destination node, it exhibits higher performance than Cachemior. However, for reasons of synchronization, Firepan does not allow the migrated VM to resume before the completion of flash cache migration. Finally, FirepanIF overcomes the limitation of Firepan by resuming the VM migration while the flash cache migration procedure is in progress. This is possible due to the IF, which traces I/O requests from the migrated VM and records write accesses in the invalidation table. FirepanIF can improve the performance of the I/O workload by up to 21.87% when the size of the flash cache is 20 GB.

Author Contributions: The work presented here was completed in collaboration between all authors. conceptualization, H.P.; validation, M.L.; formal analysis, H.P. and C.-H.H.; investigation, H.P. and C.-H.H.; writing–original draft preparation, H.P., M.L., and C.-H.H.; writing–review and editing, C.-H.H.; supervision, C.-H.H.; funding acquisition, H.P. and C.-H.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIP) (No. NRF-2019R1C1C1011068 and No. NRF-2017R1C1B5016000) and research funds for newly appointed professors of Jeonbuk National University in 2016.

Acknowledgments: The authors would like to thank the anonymous reviewers of Applied Sciences journal for their valuable comments and suggestions to improve the quality of the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Milojičić, D.; Llorente, I.M.; Montero, R.S. Opennebula: A cloud management tool. *IEEE Internet Comput.* **2011**, *15*, 11–14. [CrossRef]
2. Sefraoui, O.; Aissaoui, M.; Eleuldj, M. OpenStack: toward an open-source solution for cloud computing. *Int. J. Comput. Appl.* **2012**, *55*, 38–42. [CrossRef]
3. López, L.; Nieto, F.J.; Velivassaki, T.H.; Kosta, S.; Hong, C.H.; Montella, R.; Mavroidis, I.; Fernández, C. Heterogeneous secure multi-level remote acceleration service for low-power integrated systems and devices. *Procedia Comput. Sci.* **2016**, *97*, 118–121. [CrossRef]
4. Varghese, B.; Leitner, P.; Ray, S.; Chard, K.; Barker, A.; Elkhatib, Y.; Herry, H.; Hong, C.; Singer, J.; Tso, F.; et al. Cloud Futurology. *Computer* **2019**, *52*, 68–77. [CrossRef]
5. Park, H.; Yoo, S.; Hong, C.H.; Yoo, C. Storage SLA guarantee with novel ssd i/o scheduler in virtualized data centers. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *27*, 2422–2434. [CrossRef]
6. Hong, C.H.; Lee, K.; Kang, M.; Yoo, C. qCon: QoS-Aware Network Resource Management for Fog Computing. *Sensors* **2018**, *18*, 3444. [CrossRef] [PubMed]
7. Badshah, A.; Ghani, A.; Shamshirband, S.; Chronopoulos, A.T. Optimising infrastructure as a service provider revenue through customer satisfaction and efficient resource provisioning in cloud computing. *IET Commun.* **2019**, *13*, 2913–2922. [CrossRef]
8. Rehman, A.; Hussain, S.S.; ur Rehman, Z.; Zia, S.; Shamshirband, S. Multi-objective approach of energy efficient workflow scheduling in cloud environments. *Concurr. Comput. Pract. Exp.* **2019**, *31*, e4949. [CrossRef]
9. Osanaiye, O.; Chen, S.; Yan, Z.; Lu, R.; Choo, K.K.R.; Dlodlo, M. From cloud to fog computing: A review and a conceptual live VM migration framework. *IEEE Access* **2017**, *5*, 8284–8300. [CrossRef]
10. Jo, C.; Gustafsson, E.; Son, J.; Egger, B. Efficient live migration of virtual machines using shared storage. *ACM Sigplan Not.* **2013**, *48*, 41–50. [CrossRef]
11. Luo, T.; Ma, S.; Lee, R.; Zhang, X.; Liu, D.; Zhou, L. S-cave: Effective ssd caching to improve virtual machine storage performance. In Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, Edinburgh, UK, 7–11 September 2013; pp. 103–112.
12. Clark, C.; Fraser, K.; Hand, S.; Hansen, J.G.; Jul, E.; Limpach, C.; Pratt, I.; Warfield, A. Live migration of virtual machines. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, Berkeley, CA, USA, 2–4 May 2005; Volume 2, pp. 273–286.
13. Meng, X.; Pappas, V.; Zhang, L. Improving the scalability of data center networks with traffic-aware virtual machine placement. In Proceedings of the 2010 Proceedings IEEE INFOCOM, San Diego, CA, USA, 14–19 March 2010; pp. 1–9.
14. Arteaga, D.; Zhao, M. Client-side flash caching for cloud systems. In Proceedings of the International Conference on Systems and Storage, Haifa, Israel, 30 June–2 July 2014; pp. 1–11.
15. Park, J.Y.; Park, H.; Yoo, C. Design and Implementation of Host-side Cache Migration Engine for High Performance Storage in A Virtualization Environment. *KIISE Trans. Comput. Pract.* **2016**, *22*, 278–283. [CrossRef]
16. Koutoupis, P. Advanced hard drive caching techniques. *Linux J.* **2013**, *2013*, 2.
17. Zhang, Y.; Soundararajan, G.; Storer, M.W.; Bairavasundaram, L.N.; Subbiah, S.; Arpaci-Dusseau, A.C.; Arpaci-Dusseau, R.H. Warming Up Storage-Level Caches with Bonfire. In Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST 13), San Jose, CA, USA, 12–15 February 2013; pp. 59–72.
18. Lu, T.; Huang, P.; Stuart, M.; Guo, Y.; He, X.; Zhang, M. Successor: Proactive cache warm-up of destination hosts in virtual machine migration contexts. In Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016; pp. 1–9.
19. Hwang, J.; Uppal, A.; Wood, T.; Huang, H. Mortar: Filling the gaps in data center memory. In Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Salt Lake City, UT, USA, 1–2 March 2014; pp. 53–64.
20. Bhagwat, D.; Patil, M.; Ostrowski, M.; Vilayannur, M.; Jung, W.; Kumar, C. A practical implementation of clustered fault tolerant write acceleration in a virtualized environment. In Proceedings of the 13th

USENIX Conference on File and Storage Technologies (FAST 15), Santa Clara, CA, USA, 16–19 February 2015; pp. 287–300.
21. Kgil, T.; Mudge, T. FlashCache: a NAND flash memory file cache for low power web servers. In Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, Seoul, Korea, 22–25 October 2006; pp. 103–112.
22. Byan, S.; Lentini, J.; Madan, A.; Pabon, L.; Condict, M.; Kimmel, J.; Kleiman, S.; Small, C.; Storer, M. Mercury: Host-side flash caching for the data center. In Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), San Diego, CA, USA, 16–20 April 2012; pp. 1–12.
23. Documentation of the vSphere Flash Read Cache v6.5. Available online: https://docs.vmware.com/en/VMware-vSphere/6.5/com.vmware.vsphere.storage.doc/GUID-07ADB946-2337-4642-B660-34212F237E71.html (accessed on 17 December 2019).
24. Meng, F.; Zhou, L.; Ma, X.; Uttamchandani, S.; Liu, D. vCacheShare: Automated server flash cache space management in a virtualization environment. In Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 14), Philadelphia, PA, USA, 19–20 June 2014; pp. 133–144.
25. Lee, H.; Cho, S.; Childers, B.R. CloudCache: Expanding and shrinking private caches. In Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture, San Antonio, TX, USA, 12–16 February 2011; pp. 219–230.
26. Koller, R.; Marmol, L.; Rangaswami, R.; Sundararaman, S.; Talagala, N.; Zhao, M. Write policies for host-side flash caches. In Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST 13), San Jose, CA, USA, 12–15 February 2013; pp. 45–58.
27. Holland, D.A.; Angelino, E.; Wald, G.; Seltzer, M.I. Flash caching on the storage client. In Proceedings of the 2013 USENIX Annual Technical Conference (USENIX ATC 13), San Jose, CA, USA, 26–28 June 2013; pp. 127–138.
28. Qin, D.; Brown, A.D.; Goel, A. Reliable writeback for client-side flash caches. In Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 14), Philadelphia, PA, USA, 19–20 June 2014; pp. 451–462.
29. Arteaga, D.; Otstott, D.; Zhao, M. Dynamic block-level cache management for cloud computing systems. In Proceedings of the Conference on File and Storage Technologies, San Jose, CA, USA, 14–17 February 2012.
30. Axboe, J. Flexible i/o Tester. 2016. Available online: https://github.com/axboe/fio (accessed on 17 December 2019).