Application-centric SSD Cache Allocation for Hadoop Applications

Zhen Tang
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
tangzhen12@otcaix.iscas.ac.cn

Wei Wang*
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
wangwei@otcaix.iscas.ac.cn

Yu Huang
State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University
yuhuang@nju.edu.cn

Heng Wu
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
wuheng@otcaix.iscas.ac.cn

Jun Wei
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
wj@otcaix.iscas.ac.cn

Tao Huang
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
tao@otcaix.iscas.ac.cn

ABSTRACT

Flash-based Solid State Drive (SSD) is widely used in the virtualization environment, usually as the cache of the hard disk drive-based Virtual Machine (VM) storage, to improve the IO performance. Existing SSD caching schemes are mainly driven by VM-centric metrics. They treat the VMs as independent units and focus on critical low-level performance metrics of individual VMs, such as the working set, the IO latency, or the throughput. However, for elastic Hadoop applications consisting of multiple VMs, the workload is rapidly changing, and the importance of differnet VMs may be different even if they have the same low-level IO pattern. In this situation, the VM-centric SSD caching schemes may not lead to the best performance, i.e., the shortest job completion time. Considering the importance of VMs and relationships among VMs inside the application may potentially better improve the performance, which we regard as the application-centric metrics. We propose the Application-Centric SSD caching for Hadoop applications (AC-SSD), which reduces the job completion time from the application level. AC-SSD uses the genetic algorithm based approach to calculate the nearly optimal weights of virtual machines for allocating SSD cache space and controlling the I/O Operations Per Second (IOPS) based on the importance of the VMs. Moreover, AC-SSD introduces the closed-loop adaptation to face the rapidly changing workload. The evaluation shows that AC-SSD reduces the job completion time by up to 39% for IO sensitive workloads, and up to 29% for rapidly changing workloads.

CCS CONCEPTS

• **Information systems → Cloud based storage**; *Storage virtualization*; *Flash memory*;

KEYWORDS

SSD; cache; virtualization; Hadoop

ACM Reference format:
Zhen Tang, Wei Wang, Yu Huang, Heng Wu, Jun Wei, and Tao Huang. 2017. Application-centric SSD Cache Allocation for Hadoop Applications. In *Proceedings of Internetware'17, Shanghai, China, September 23, 2017,* 10 pages.
https://doi.org/10.1145/3131704.3131708

1 INTRODUCTION

Virtualization technology is widely used, aiming to consolidate multiple Virtual Machines (VM) in one hypervisor to get better utilization of hardware resources[10]. Hypervisors, where VMs are hosted, are mostly attached with a slow-speed Hard Disk Drive (HDD) based back end shared storage to store VM images. In this architecture, it is widely recognized that the IO performance is vital to the efficient use of virtualized resources[25] [11]. Hadoop[2] applications are widely used in the virtualization environment, supporting the enterprises to complete tasks such as mining user behaviors and recommending products. These days, cloud providers also offer VM-based big data platforms, mostly for Hadoop applications. For example, Amazon Elastic Compute Cloud (EC2) provides Amazon Elastic MapReduce (EMR) [1] service to host big data applications. Also, Microsoft Azure provides HDInsight [19], a fully-managed Hadoop solution. For the open source community, Openstack foundation provides Sahara [21] to create Hadoop clusters dynamically.

*Corresponding author

Internetware'17, September 23, 2017, Shanghai, China

ACM ISBN 978-1-4503-5313-7/17/09…$15.00
https://doi.org/10.1145/3131704.3131708

For Hadoop applications, the most important performance metric is the job completion time (JCT). As Hadoop applications are supported by the HDFS [23], improving IO performance is vital to reducing the job completion time.

Flash-based Solid State Drive (SSD) has recently been a widespread solution to improving the IO performance [16] [14]. In the virtualization environment, SSD caching is usually deployed at the hypervisor-scale, i.e., VMs on the same hypervisor share the physical SSD storage for caching [7] [15] [20] [4]. The whole SSD is divided into multiple parts to be used as individual cache for different VMs. The IO operations will first arrive at the SSD cache. Upon a cache hit, the operation will be quickly served by the cached data. Otherwise, it will be served by the much slower back end storage system. By using SSD cache to improve the IO performance of VMs, the job completion time can be reduced.

Though SSD caching is vital for improving the IO performance, existing caching schemes face severe challenges in the virtualization environment, especially for Hadoop applications, as existing work is mostly *VM-centric*. The *VM-centric* approaches focus on important metrics of individual VMs such as the working set, the IO latency, or the throughput, aiming to improve the performance from the view of VM. The VM-centric approach can lead to the best IO performance from the view of virtual machines. However, it may not lead to the best performance from the application view, i.e., the job completion time of Hadoop applications. During the execution of a Hadoop job, the virtual machine not only read or write data from local storage, but also request data from other nodes via network communication. The importance of different virtual machines inside the cluster may be different due to the data locality. Thus, for VM centric approach, the optimized virtual machine may contribute little to the overall performance, which means the cache is not efficiently used. Table 1(a) demonstrates a simple Hadoop cluster consisting of 5 nodes, which runs specific jobs. The HDFS namenode is deployed on Node-2. Also, there is a background replication task running on one datanode (Node-5). The total size of the working set is 20GB and the total size of the SSD cache is 5GB. Each node in the cluster has different importance (ratio of IO operations related to the jobs) and working set. Furthermore, as the distributed file system is deployed on each node, the low-level IO patterns of VMs inside the cluster are similar. Table 1(b) shows the SSD cache allocation and the miss rate of the VM-centric approach and the App-centric approach. The former allocates the per VM SSD cache according to the working set of VMs, while the latter allocates according to the importance of VMs. Assuming that the average latency of SSD is 0.1x of HDD, if allocating SSD cache according to the importance of nodes from the application level, there may be significant performance improvement comparing to the VM-centric approach (the average latency is reduced by 38.3%), as shown in Table 1(c).

Additionally, tenants of cloud platforms usually apply for multiple virtual machines and use the cluster built from them. The virtual machines inside a cluster will face the similar workload. Furthermore, the cloud provider allocates the virtual machines using the load balance strategy, from the view of the virtual machine. Thus, virtual machines belonging to different Hadoop applications may be placed in the same hypervisor, and multiple applications may also be deployed on one hypervisor. In this architecture, the relationships between virtual machines inside a Hadoop application are important. Ignoring the relationships and considering the SSD cache allocation from the view of virtual machines may not lead to the best performance. Thus, we need *application-centric* SSD cache allocation, which takes the relationships inside an application into consideration to improve the application-level performance.

Table 1: Demonstration of VM-centric and App-centric approach

(a) Characteristics of workloads

Node	Working Set	Importance
Node-1	5 GB	0.1
Node-2	1 GB	0.4
Node-3	4 GB	0.1
Node-4	3 GB	0.3
Node-5	7 GB	0.1

(b) SSD cache allocation and miss rate

Node	VM-centric	App-centric
Node-1	1.25 GB (75%)	0.5 GB (90%)
Node-2	0.25 GB (75%)	2 GB (0%)
Node-3	1 GB (75%)	0.5 GB (87.5%)
Node-4	0.75 GB (75%)	1.5 GB (50%)
Node-5	1.75 GB (75%)	0.5 GB (92.9%)

(c) Average Latency

Approach	Latency	Ratio
VM-centric	$0.775 \cdot l_{HDD}$	1
App-centric	$0.47836 \cdot l_{HDD}$	0.617

SSD has the significant advantage over HDD, especially for random IO operations. However, SSD is much more expensive than HDD. The size of SSD is limited, so we need to allocate per VM SSD caches according to requirements of VMs. Also, the IOPS (IO Operations Per Second) capacity of SSD is limited. Without controlling the IOPS capacity, there may be potential resource contention among VMs hosted on one hypervisor and leading to the waste of cache space. Thus, SSD needs to be efficiently used, by allocating the space and controlling the IOPS. We regard this as the allocation of **storage** and **IOPS** capacity of SSD cache. However, to get the optimized plan for allocating per VM storage and IOPS capacity of SSD cache is hard as the benefit for allocating different ratio of capacity maybe totally different, which cannot be calculated by a straight-forward approach. Furthermore, workloads of the Hadoop application is continuously changing. Also, for different stages of one Hadoop job, different VMs have their own IO patterns and the importance. The static SSD cache space allocation and IOPS control is far less enough. Thus, the SSD caching scheme must be adaptive.

The discussions above impulse two critical technical challenges:

(1) **How to allocate appropriate SSD cache space and control IOPS for virtual machines inside the Hadoop cluster from the application view?**

When allocating different cache space for different virtual machines, the benefits (the reduction of the job completion time) for different types of Hadoop jobs may be totally different.

(2) **How to dynamically change the plan to adapt to continuously and dynamically changing workloads?** By using an offline approach, allocating SSD cache space and controlling the IOPS may be easy and efficient. However, as the running jobs on the Hadoop cluster are changing rapidly, the static approach is far less enough.

We present AC-SSD, a SSD caching scheme, to reduce the job completion time of Hadoop applications for elastic Hadoop clusters provided by cloud platforms. After proving that the application-centric SSD cache allocation is NP complete, AC-SSD introduces a genetic algorithm based approach to get the nearly optimal weight for each virtual machine and then allocates SSD cache space and controls the IOPS based on the weight. Furthermore, AC-SSD uses closed-loop adaptation to handle the rapidly changing workloads. During the execution of the closed-loop, AC-SSD monitors the Hadoop application by gathering the performance of low-level IO operations and network interactions, as well as the detail of running jobs, and infers the requirement of SSD cache space and IOPS of each virtual machine. Then the genetic algorithm based approach is used to calculate the weights and finally be applied by resizing the per VM SSD cache and the IOPS capacity. The evaluation shows that AC-SSD reduces the job completion time by up to 39% for IO sensitive workloads, and up to 29% when the type of workload is rapidly changing.

The rest of the paper is organized as follows. In Section 2, we discuss the design details of AC-SSD. In Section 3, we discuss the execution of the closed-loop, which dynamically detects the workload of the Hadoop application and triggers the adjustment of SSD cache. In Section 4, we discuss the genetic algorithm based approach used in AC-SSD to allocate per VM SSD cache storage and IOPS capacity. In Section 5, we use several experiments to show the effect of the genetic algorithm based approach and closed-loop adaptation. We discuss the related work in section 6. The Section 7 concludes the paper.

2 OVERVIEW OF AC-SSD

The SSD cache benefits the performance of Hadoop applications by reducing the job completion time as it improves the IO performance of virtual machines. However, for SSD caches, the storage and the IOPS capacity are both limited. We need to allocate the cache space as well as control the IOPS for each virtual machine, according to the workloads running in the Hadoop application.

2.1 Illustrative Examples

In this paper, we focus on the famous big data platform, Apache Hadoop [2] ecosystem, including HDFS [23], YARN [24], HBase [9] and Mahout [3].

We have found two meta-level principles of Hadoop applications, as shown below:

(1) **Relationships among virtual machines:** As there are frequently interactions among virtual machines inside the cluster, the importance of each virtual machine may be different.

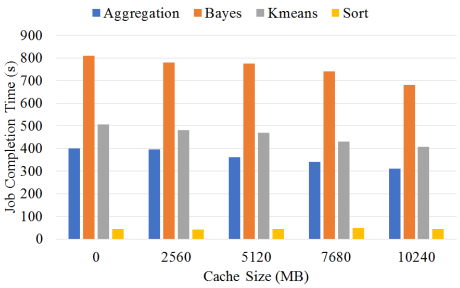


Figure 1: Job completion time of 4 workloads when allocating different size of SSD cache to the cluster.

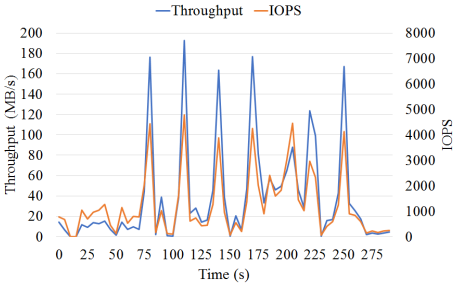


Figure 2: The changes of throughput and IOPS during the execution of the KMeans workload.

For example, the name node in HDFS and the resource manager node in YARN in the cluster are the fundamental nodes in the cluster, and have the great impact on job completion time, even though they may have the same IO pattern as the data nodes.

(2) **Temporal characteristics:** For different stages during the execution of the job, the IO patterns are different from the application view. For example, the hot spot and the important virtual machines in the map stage and the reduce stage may be different.

Next, we use two illustrative examples to show these meta-level principles. Firstly, we observe that different Hadoop applications may have different IO patterns, including the hot spot and the ratio of random IO operations. Furthermore, multiple nodes in the same application have the same low-level IO pattern during the execution of the job, but with different importance. The example is shown in Figure 1. We build a 5-VM cluster and choose 4 representative benchmarks from Hadoop ecosystem, including SQL Aggregation (database), Bayes (machine learning), Kmeans (machine learning) and Sort (CPU sensitive). The x axis represents the total SSD cache space allocated to the cluster. We observe that when allocating the different size of the SSD cache, the job completion time of Sort benchmarks changes slightly, while for Aggregation and Bayes benchmark, the job completion time is reduced significantly.

Secondly, we observe that for different stages in one job, the IO patterns may be different. For example, for the map and reduce stage in one job, the importance of virtual machines may be different, which results in different hot spots. The static allocation

of SSD cache space is far less enough to face the rapidly changing workloads and the IO patterns. The example is shown in Figure 2. We build a 5-VM cluster and run the KMeans benchmark from Apache Mahout, part of the Hadoop ecosystem. The x axis represents the time, while y axes represent the throughput and IOPS of the cluster during the execution. We observe that in different stages, IO patterns are different from the view of the application. In this scenario, the static SSD cache resource allocation approach may be incapable. Thus we need closed-loop adaptation to dynamically detect the change of workloads and IO patterns, to further allocate the storage and IOPS capacity of SSD cache efficiently.

2.2 System Architecture

AC-SSD provides the allocation of SSD cache storage and IOPS capacity on per VM basis. AC-SSD uses genetic algorithm based approach to get the nearly optimal results. Furthermore, to face the rapidly changing workloads and the different IO patterns for different stages during the execution of jobs, AC-SSD uses closed-loop adaptation, so as to dynamically detect the change of workloads and the performance of virtual machines, including IO patterns and network communications.

The closed-loop adaptation is inspired by MAPE-K [8], which consists of three steps, **Monitor**, **Solve** and **Apply**, as shown below. More details of the closed-loop adaptation can be found in Section 3.

(1) Monitor: In this step, AC-SSD monitors running processes and network communications of each virtual machine, and then builds the topology of the Hadoop application showing the relationships among VMs inside the application. Also, AC-SSD monitors the status of the running jobs for each Hadoop application, along with the low-level IO patterns of each virtual machine, including the throughput and the IOPS.
(2) Solve: In this step, a genetic algorithm is used to calculate the nearly optimal weights of storage and IOPS capacity for each VM inside each Hadoop application according to the importance of VMs. More detail can be found in Section 4.
(3) Apply: In this step, AC-SSD resizes the SSD cache space and controls IOPS for each VM, to apply the specific SSD cache resource allocation plan.

The architecture of AC-SSD is shown in Figure 3. To support the closed-loop adaptation and the nearly optimal SSD cache allocation, AC-SSD consists of the controller, monitor, solver, executor and agents on virtual machines and hypervisors, as shown below:

(1) Controller: The **Controller** triggers, traces and maintains the whole closed-loop, and provides the uniform RESTful API for management. The controller communicates with all the other components using the RESTful API and gathers the information from them. It also controls the execution of the closed-loop adaptation and starts the new round of SSD cache allocation frequently.
(2) Monitor: The **Monitor** gathers performance of CPU, disk and network from the agents deployed on virtual machines to figure out the requirement of SSD cache storage and IOPS capacity. Furthermore, the monitor detects running processes of each virtual machine to get the topology of the Hadoop cluster and then traces the execution of the jobs. Additionally, the monitor gathers the usage of SSD from the agent deployed on each hypervisor to figure out the resource supply and the limitation of SSD cahce storage and IOPS capacity. The gathered performance data will then be used by **Solver** to calculate the nearly optimal results.

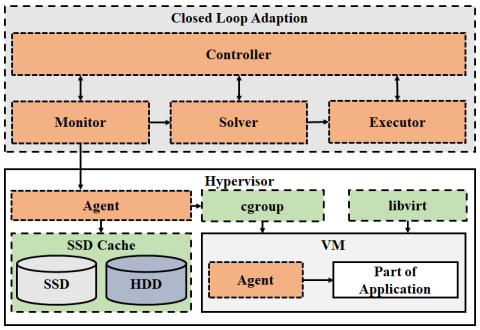


Figure 3: Architecture of AC-SSD.

(3) Solver: The **Solver** calculates weights of SSD cache storage and IOPS capacity for each virtual machine inside each application, for current running workloads. A genetic algorithm based approach is used to calculate the nearly optimal results based on the importance of virtual machines. Finally, a SSD cache allocation plan will be generated for the **Executor** to apply.
(4) Executor: The **Executor** sends messages to the agent on each hypervisor to apply the plan, including resizing the SSD cache space and controlling the IOPS for each virtual machine. For the prototype of AC-SSD, we embedded an LRU-based SSD cache in QEMU, and use the cgroup to control the IOPS. We use the libvirt library to control the lifecycle of virtual machine to help resize the SSD cache and control the IOPS. More details of the implementation of AC-SSD can be found in Section 5.
(5) Agent: The **Agent** is deployed on each virtual machine and hypervisor. For virtual machines, the agent aims to gather performance data along with the information of running processes, in order to build the topology of application, to figure out the role of the virtual machine, and to identify the IO pattern. For hypervisors, the agent resizes the SSD cache and controls the IOPS for virtual machines hosted on them.

3 DYNAMIC SSD CACHE SPACE ALLOCATION AND IOPS CONTROL

For Hadoop applications, we observed that the importance of different VMs are different when facing different workloads or for different stages during the execution of one workload. The SSD cache requirements of storage and IOPS capacity for each virtual machine are different, even for those with the similar disk IO usage and IO patterns. Thus we need the application-centric SSD cache allocation. Furthermore, we use the closed-loop adaptation in AC-SSD to react to the change of workloads. The closed-loop adaptation is inspired by MAPE-K[8] feedback loop, which consists of three steps: monitor, solve and apply.

3.1 Three Levels of Monitoring

We monitor the performance and the status of the Hadoop application from the hypervisor, the application and the virtual machine level, so as to figure out the importance of each virtual machine.

For the hypervisor level, AC-SSD uses the agent deployed on each hypervisor to monitor the usage of SSD space and IOPS, along with the usage of per VM SSD cache.

For the application level, AC-SSD uses the agent deployed on each virtual machine to detect the "borderline" of the Hadoop application, i.e., to figure out each virtual machine inside a Hadoop application, and then build the topology for each application. Firstly, we monitor the network communication among virtual machines. We record the network connections and then build directed graphs based on them. Each graph represents a Hadoop application. Furthermore, we detect the processes of virtual machines, to get the role of different virtual machines, such as the master node and the slave nodes. We also trace the execution of jobs to figure out the critical virtual machines inside the cluster. The relationships among VMs inside the application and the job details will help infer the importance of VMs.

For the virtual machine level, AC-SSD uses the agent deployed on each virtual machine to monitor the CPU usage, network throughput and IO patterns. SSD can benefit the IO performance significantly, especially for random IO operations. Thus, the agent will monitor the average size of IO requests to figure out the ratio of random IO operations for each virtual machine. With the help of the application topology, the agent monitors the network communication for each virtual machine, especially for the incoming throughput so as to get the importance of VMs inside an application.

3.2 Execution of the Closed-Loop

The execution of the closed-loop and the main steps are shown in Figure 4. AC-SSD triggers the execution of the closed-loop for every minute. Such interval is also the window size for monitoring the performance and application status. The one-minute interval is sufficient in practice, and get a balance between accuracy of monitoring and system overhead.

For one round of the closed-loop adaptation, AC-SSD firstly monitors the low-level performance from virtual machine, including the CPU usage, the IO overload and the network throughput among other VMs (Step 1.1). Also, AC-SSD figures out the "borderline" of each application along with the relationships among the virtual machines (Step 1.2). Furthermore, AC-SSD traces the execution of jobs for each cluster (Step 1.3). After monitoring the performance data and the status of the application, the gathered information will be used by the genetic algorithm based approach to infer the importance for each virtual machine inside the application (Step 2.1). We aim to calculate the weight of storage and IOPS capacity according to the importance of each virtual machine inside the application (Step 2.2). More detailed information of the **Solve** step can be found in Section 4. Finally, AC-SSD triggers the Executor to use the agent deployed on each hypervisor to change the SSD cache storage and IOPS capacity for each virtual machine (Step 3). The closed-loop adaptation is triggered for every minute, which is the window size for the sliding window based monitoring (Step 4).

4 NEARLY OPTIMAL SSD CACHE ALLOCATION OF SPACE AND IOPS CAPACITY

We use the genetic algorithm based approach to calculate the nearly optimal weights of SSD cache storage and IOPS capacity for each virtual machine, from the view of the cluster.

4.1 Problem Definition

For the nearly optimal SSD cache allocation, we aim to calculate the weights of SSD cache storage and IOPS capacity for each virtual machine hosted on each hypervisor. The SSD cache storage and IOPS capacity are then allocated according to the weights.

Firstly, we regard the Hadoop application A as the set of n virtual machines v, $A_i = \{v_1, v_2, \ldots, v_n\}$. Also, each virtual machine is bound to one specific hypervisor H. We have $H_i = \{v_1, v_2, \ldots, v_n\}$. From the view of the application, when facing a specific workload, the interactions among different virtual machines are different. We use the weight w to describe the importance of virtual machine, which shows the contribution to the job completion time bringing by the IO operations. Thus, we have the weight vector $W = \{w_1, w_2, \ldots, w_n\}$, where w_i is the weight of virtual machine v_i.

Furthermore, for different types of IO patterns, for example, the sequential IO operations and random IO operations, the requirement of SSD cache storage and IOPS capacity may be totally different. Allocating the SSD cache space and controlling the IOPS due to the importance and the IO access pattern of the virtual machine will help improve the performance and reduce the job completion time.

Our goal is to calculate the weight of SSD cache storage and IOPS capacity for each virtual machine, by using the performance data and the application-level runtime status gathered in the **Monitor** step in the closed-loop adaptation, so as to manage the SSD cache from the application view.

4.2 Proof of NP Completeness

Next, we show that the 0-1 knapsack problem is a special instance of the application-centric SSD cache allocation problem, to prove that the problem is NP hard. Thus it is reasonable to use the genetic algorithm to get the nearly optimal results. To allocate SSD cache and control the IOPS for each virtual machine to get the best performance, we need to allocate appropriate size of SSD cache space and set the IOPS limitation for selected virtual machines, so as to minimize the job completion time. We assume that for a specific workload, the cost (usage of SSD cache storage and IOPS) for each VM is constant, and so does the value (the reduction of job completion time).

Then, we add restrictions to the problem as following to build a special case of 0-1 knapsack problem. The item set is the VM set $V = \{v_1, v_2, \ldots, v_n\}$, while the total number of the items is $|V| = n$. Value for each item $W = \{w_1, w_2, \ldots, w_n\}$ is the reduction of job completion time when the virtual machine is selected to allocate specific size of SSD cache space and set the IOPS limitation. The cost for each item $C = \{c_1, c_2, \ldots, c_n\}$ is the usage of cache size and IOPS limitation for each virtual machine. The capacity of the knapsack is the total cache storage and IOPS capacity for the SSD.

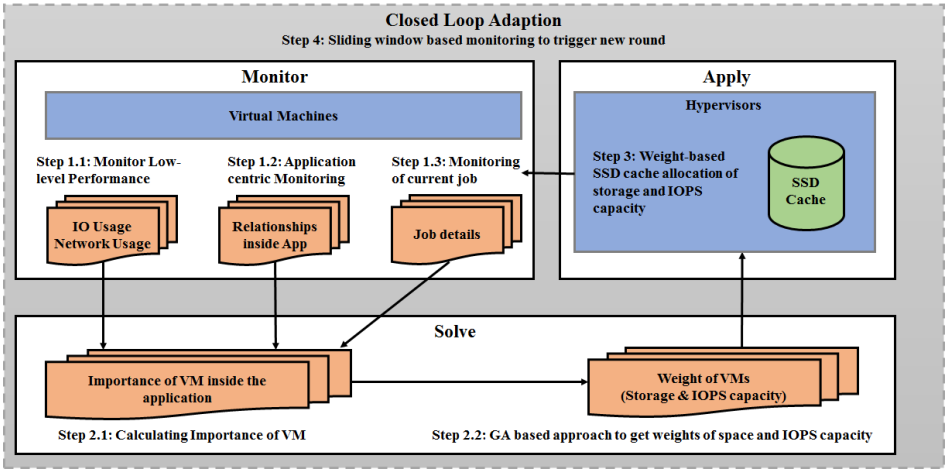


Figure 4: Execution of the closed-loop.

As it is straight-forward that the 0-1 knapsack problem is NP complete, the allocation of SSD cache storage and IOPS capacity is NP complete.

4.3 Genetic Algorithm

Thus, we choose to use genetic algorithm to calculate the nearly optimal weights for SSD cache allocation. We aim to create connection between high-level job completion time of Hadoop application and the low-level characteristics of IO operations through the fitness. We calculate the fitness by considering the type of the job, the importance of different VMs, and the characteristics of IO operations, so as to give a value which is highly relative to the job completion time of the application.

There are some critical parameters in the genetic algorithm which we need to introduce here. The chromosome indicates the weight tuple $\{w_{storage}, w_{IOPS}\}$ of a specific VM, which is the ratio of storage and IOPS capacity. Thus, The genome indicates a specific SSD cache allocation plan of the whole platform, including all virtual machines belong to different applications along with the weights. We define the selection operation as selecting genomes randomly by their fitnesses. We define the crossover operation as selecting random numbers of chromosome of two genomes and swap them, and then the weights will be normalized again. We define the mutation operation as changing the weight of random numbers of chromosomes of a genome inside a specific range.

For the most important part, the fitness aims to help predict the job completion time. Firstly, we use the IO patterns of the job, the contribution of the network throughput from IO operations and the characteristics of IO operations to characterize the requirement of SSD cache for each VM, including the storage and IOPS. The importance of the VM can be inferred by figuring out the intensity of specific VM by the the contribution of network throughput from IO operations. Secondly, we need to consider the low-level IO, which is the IO load of the specific VM, as we called the "IO sensitivity". We regard the IO sensitivity as the ratio of IO and non-IO operations. Thirdly, we consider the intensity of the random access of the VMs. We regard the random access intensity as the reciprocal

Algorithm 1: Fitness calculation

Input: Target genome; CPU time, IO usage (IO time, Bandwidth and IOPS), network throughput for each VM
Output: Fitness for target genome
Initialization;
foreach *Chromosome c in target genome* **do**
  VM $c.v$:= VM bound to c;
  Application $c.app$:= Application bound to c;
  Weight $c.ws$:= Weight of storage capacity bound to c;
  Weight $c.wi$:= Weight of IOPS capacity bound to c;
  IO sensitivity $c.s$:= $c.v$.IOTime / $c.v$.CPUTime;
  Random access intensity $c.r$:= $c.v$.TotalIOPS / $c.v$.TotalBandwidth;
  // Calculating the VM intensity
  VM importance $c.i$:= 0;
  foreach *Network throughput t between c.v and VMs in c.app* **do**
    $c.i$:= $c.i$ + t / $c.v$.TotalBandwidth ;
  end
end
// Normalization
foreach *Chromosome c in target genome* **do**
  $c.s$:= $c.s$ / Total $c.s$;
  $c.r$:= $c.r$ / Total $c.r$;
  $c.i$:= $c.i$ / Total $c.i$;
end
// Calculate the match degree
foreach *Chromosome c in target genome* **do**
  Match degree $c.m$:= $(c.ws - c.s)^2 + (c.wi - c.r)^2 + (c.ws - c.i)^2 + (c.wi - c.i)^2$;
end
Fitness $f := 4 \times (count_{VM}) - \sum c.m$;
return f;

of the average size of IO operations. Finally, the fitness is calculated by comparing the match degree of the weight with the IO sensitivity, the VM importance and the random access intensity of the VM. We calculate the match degree by summing up the euclidean metric between the weight, the normalized IO sensitivity, the VM importance and the random access intensity. We finally adjust the match degree to a positive relative value to the job completion time. The algorithm for calculating fitness is shown in algorithm 1.

5 EVALUATION

In this section, we seek to answer the following questions:

(1) How does AC-SSD impact the performance of Hadoop applications? Can AC-SSD reduce the job completion time of Hadoop application better comparing to the shared cache?
(2) Can AC-SSD adapt to different types of changing workloads? Can AC-SSD resize the SSD cache and control the IOPS for VM dynamically?

5.1 Experiment Design

5.1.1 Implementation of AC-SSD. We implement AC-SSD on Xen[5], a widely used hypervisor. The components of the closed-loop adaptation are implemented in Java. For the SSD cache, we implemented an LRU cache on QEMU, an emulator used by Xen. The cache for each virtual machine is stored on SSD as separate file. For the IOPS control, we use Linux cgroup to change the weight of different virtual machines. For the control of virtual machine, we use the library **libvirt**, which used by the agent deployed on the hypervisors to safely suspend the virtual machine before changing the cache space and IOPS capacity. Furthermore, we use the Linux proc filesystem and the sysstat package to monitor the usage of CPU, disk IO and network communication.

5.1.2 Experiment Setup. Our experiments are performed on four Inspur blade servers, each with two 8-core Intel Xeon CPU and 16GB memory. For each server, a 240GB Intel 535 SSD is attached, along with one Toshiba AL13SEB300 300GB SAS hard disk drive. We create the shared storage based on the HDDs.

To simulate the behavior of the real world cloud provider, we use 3 clusters rather than one cluster with three tenants. The deployment of the three clusters in the experiment environment is shown in Figure 5. The cluster 1 consists of five virtual machines, while cluster 2 consists of ten and cluster 3 consists of five. For cluster 1, the master node does not interfere with any others, while the master nodes of cluster 2 and 3 are placed on the same hypervisor, though they have the same number of nodes. As mentioned before, the cloud providers usually place the virtual machine due to the load balancing of the CPU and memory resource of the hypervisors. We place the three clusters fairly on four hypervisors, which means each application consists of virtual machines hosted on all four hypervisors. We deployed Hadoop 2.7.2 for each cluster, while the replication of HDFS is set to 2.

5.1.3 Benchmarks. We use micro benchmarks from the Hadoop ecosystem, including Apache Hadoop, Apache HBase, and Apache Mahout. We aim to simulate the common scenarios in the real world:

(1) IO sensitive: the **TestDFSIO** benchmark. The pattern is just like the daily backup tasks for enterprises.
(2) Parallel algorithms: Including the **Sort**, **Terasort** and **Wordcount** benchmarks. The pattern is just like the processing of the intermediate data.
(3) SQL: Including the **Aggregation**, **Join** and **Scan**s benchmarks. The pattern is just like the query and search in a huge database.
(4) Data mining and websearch: Including the **Bayes**, **KMeans** and **Pagerank** benchmarks. The pattern is just like the daily analysis of user behavior.

The SQL benchmarks (Aggregation, Join and Scan) and the Pagerank benchmark are from HiBench [12].

5.2 The Effect of GA based SSD Cache Allocation

In this section, we compare the IO performance and job completion time to the shared SSD cache, which means the virtual machines deployed on the same hypervisor share the total SSD cache space.

As mentioned before, we use 3 cluster in the evaluation. The 3 clusters will run the selected workloads continuously and simultaneously. We use total cache size of 2560MB for 20 virtual machines, which means 640MB for each hypervisor.

Firstly, we use the IO sensitive workload, TestDFSIO, to evaluate AC-SSD by comparing the job completion time and the throughput to the shared cache. The number of files is set to 10, while the size of each file is 256MB. We choose the write, read and random read mode in this experiment. Figure 6 shows the reduction of job completion time for all three clusters. We observe that AC-SSD can reduce the job completion time of IO sensitive workloads by up to 39% comparing to the shared cache. In average, AC-SSD reduces the job completion time by 31%, which means the cluster can run 45% more jobs.

Figure 7 shows the improvement of throughput. The results shows that AC-SSD can improve the performance by up to 57% comparing to the shared cache, and 35% in average. We observe that AC-SSD works better for large cluster. Also, we observe that AC-SSD works better for workloads with heavy read operations, especially for random reads.

AC-SSD works well with the larger clusters and the random read workload, as AC-SSD allocates more SSD cache storage and IOPS capacity to the hot spot in the application, and considers the ratio of random IO operations.

Secondly, we run micro benchmarks of real world algorithms on separate clusters to evaluate AC-SSD. Still we use 640MB SSD cache for each hypervisor, which is the about 50% of the working set. We run the benchmarks on all three clusters simultaneously. Similarly, we compare our approach with the shared SSD cache. Also, for shared SSD cache, we do not control the IOPS.

The results are shown in Figure 8. The results shows that for workloads with frequently IO operations, such as Aggregation, Scan and Join, AC-SSD provides up to 19% reduction on job completion time comparing to the shared cache. For the workloads with low ratio of IO operations, such as Sort, Terasort and Wordcount, the job completion time is similar to the one using the shared cache. Additionally, for workloads with different stages, such as the KMeans

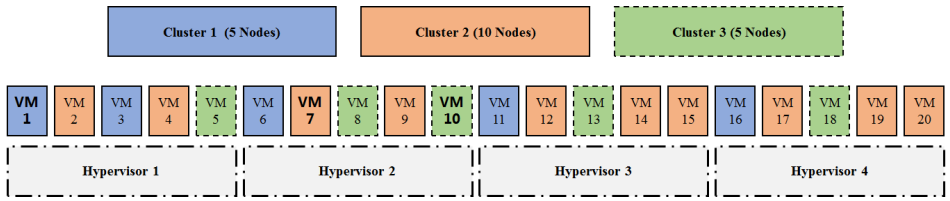


Figure 5: Experiment setup. The virtual machines with number 1, 3, 6, 11 and 16 are belonging to cluster 1, and the master node is VM1. Similarly, VM with number 2, 4, 7, 9, 12, 14, 15, 17, 19, 20 are belong to cluster 2, and the master node is VM7. The rest virtual machines forms the cluster 3, and the master node is VM10.

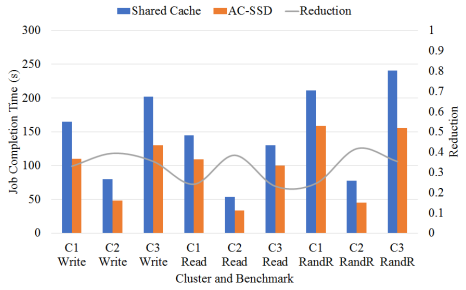


Figure 6: Job completion time of TestDFSIO for 3 clusters under 3 modes.

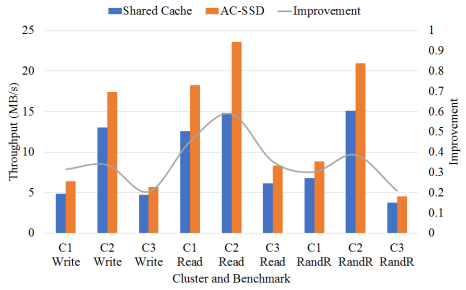


Figure 7: Throughput of TestDFSIO for 3 clusters under 3 modes.

and Bayes benchmarks, the job completion time is reduced by up to 27%, which shows the advantages of closed-loop adaptation used in AC-SSD, as AC-SSD changes the SSD cache allocation during the execution of jobs. In average, AC-SSD reduces the job completion by 14.3% for cluster 1, 17.8% for cluster 2, and 16% for cluster 3. Similarly to the evaluation on IO sensitive workloads, the reduction of job completion time for larger clusters is a little better than that for larger clusters, as AC-SSD allocates SSD cache according to the hot spot in the application. The results of cluster 1 and cluster 3 are different because the master node of cluster 2 and cluster 3 are hosted on the same hypervisor, which may result in the resource contention of IOPS. As AC-SSD controls the IOPS for each virtual machine, more storage and IOPS will be allocated to the 2 master nodes, which results in the reduction of job completion time.

Furthermore, we observed 3% more CPU usage in average when the agents deployed on VMs are enabled. The overhead introduced by the monitoring component is considerable.

5.3 Using AC-SSD in the Dynamic Environment

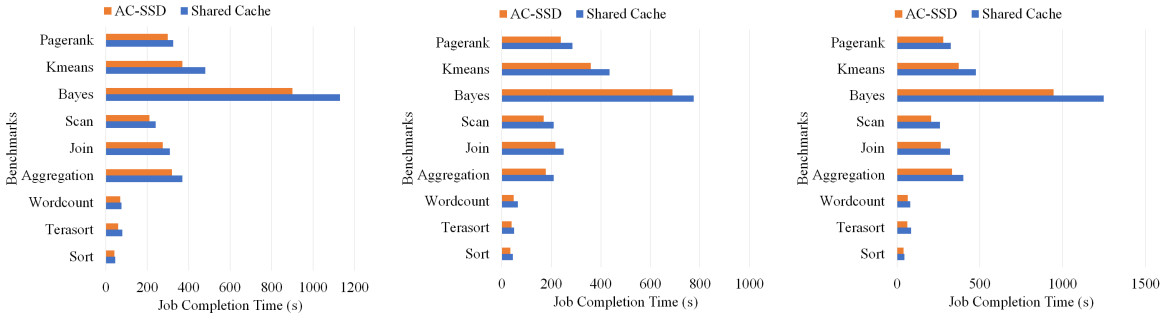
In this section, we evaluate AC-SSD by changing the type of workloads running on each cluster, in order to confirm that AC-SSD can adapt to different, rapidly changing workloads. We still use the three cluster consisting of 20 virtual machines in total. However, we change the jobs running on the cluster by the order listed in Table 2 by every 10 minutes, to simulate the changing of workloads. We aim to simulate that multiple tenants of the cloud platform are running different workloads and change the workloads if they need.

The results are shown in Figure 9. The x axis represents the number of group, while the y axis represents the job completion time. As AC-SSD decreases the cache space for cluster running CPU sensitive workloads, we observe the significant reduction of job completion time for some group of workloads, up to 29% reduction. This shows that AC-SSD resizes the SSD cache for specific virtual machines when the change of hot spot is detected, which improves the utility of SSD and performance. In average, AC-SSD reduces the job completion time by 19% for cluster 1, 22% for cluster 2, and 18% for cluster 3. This situation is better than the static workloads, especially for bigger clusters. Additionally, AC-SSD allocates more SSD cache storage and IOPS capacity to the cluster running IO sensitive workload rather than the shared cache, and quickly resizes the per VM SSD cache rather than waiting for the whole SSD to warm up when using shared cache. This may explain the significant reduction of job completion time when the workload of some clusters changes between CPU sensitive and IO sensitive, such as changing from group 3 to group 4, and from group 6 to group 7.

The results shows that AC-SSD can adapt to rapidly changing workloads, especially for bigger clusters and for the situation that the ratio of IO operations changes significantly.

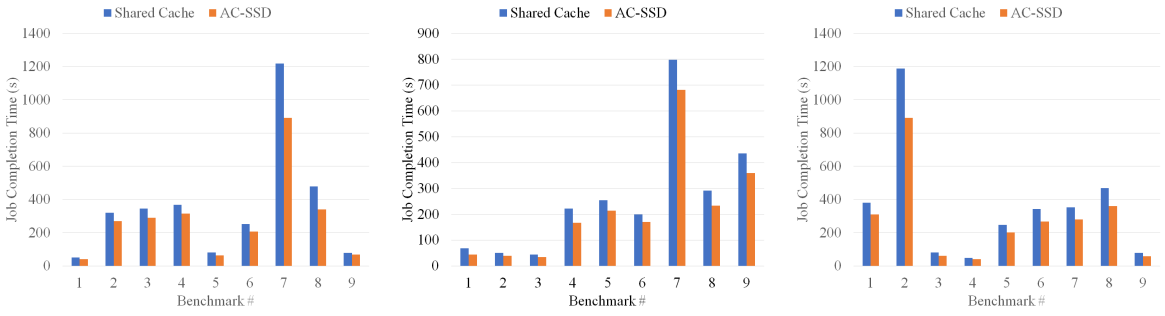
5.4 Lessons Learned

In the evaluation, we observe that AC-SSD can reduce the job completion time for Hadoop applications, especially for those workloads with heavy random IO operations. As AC-SSD controls per VM IOPS capacity, it is capable to be used in multi-tenants cloud platforms. We choose to use the small clusters and the caching system consisting of HDD and SSD in the evaluation, but AC-SSD can also be applied to the big cluster consisting of hundreds of virtual



(a) Cluster 1 (b) Cluster 2 (c) Cluster 3

Figure 8: Job completion time for 3 cluster running different workloads.



(a) Cluster 1 (b) Cluster 2 (c) Cluster 3

Figure 9: Reduction of job completion in the dynamic environment.

Table 2: Order of workloads

No.	Workload - C1	Workload - C2	Workload - C3
1	Sort	Wordcount	Aggregation
2	Join	Terasort	Bayes
3	Pagerank	Sort	Wordcount
4	Aggregation	Aggregation	Sort
5	Terasort	Join	Scan
6	Scan	Scan	Join
7	Bayes	Bayes	Pagerank
8	KMeans	Pagerank	KMeans
9	Wordcount	KMeans	Terasort

machines easily. Additionally, AC-SSD can also be applied to the caching system consisting of high speed SSD (such as NVMe SSD or Intel Optane [13]) and low speed SSD or network attached storage.

Moreover, the genetic algorithm based approach can also be applied to other types of applications consisting of multiple virtual machines, as it takes the relationships among virtual machines into consideration.

However, there are still some limitations for AC-SSD. For multiple virtual machines belong to different applications while hosted on the same hypervisor, the resource contention of IOPS cannot be ignored if such VMs are facing heavy random IO operations. This problem can only be solved by introducing the live migration of virtual machine. Also, we only monitor the virtual machine at low overhead and infer the importance by the performance data and application status. We do not use the instrumentation to monitor the behavior of Hadoop application and the change of IO hot spot accurately as it may lead to high overhead and are not capable to be used in online environment.

6 RELATED WORK

In this section, we discuss the related work for the SSD cache allocation.

Workload characterization. Some related work aims to map high-level workloads to low-level read and write [6]. The characteristics of the workload is identified by the workload mix and the request access pattern and aims to connect the low-level and high-level IO operations by recognize the access pattern and map to the operations on file system. However, the approach is not capable for the Hadoop application as it cannot create the connection between high-level application and low-level IO performance.

SSD cache allocation. Capo [22] aims to improve the performance of virtual desktops. A technique named differential durability is proposed to apply different cache policies on different path of virtual machines. Unlike Capo, AC-SSD provides a high-level mechanism for allocating the storage and IOPS capacity of SSD cache, along with the algorithm to get the nearly optimal planning, which

is capable for different types of jobs in the Hadoop application. **S-CAVE** [17] identifies the cache space demand of each VM and tries to allocate the appropriate amount of cache space. It uses ratios of effective cache space (rECS) and decides the change of cache allocation according to the four combinations. Unlike S-CAVE, AC-SSD takes the characteristics of Hadoop applications into consideration to help allocate the storage and IOPS capacity of SSD cache. **vCacheShare** [18] provides a mathematical optimization solution for server flash cache management. It calculates the optimized cache space allocation from the IO trace. Unlike vCacheShare, AC-SSD provides a weight-based cache allocation mechanism and considers the importance of virtual machines inside the Hadoop application rather than treat them individually. **Centaur** [15] focuses on the host-side SSD cache partitioning. The adjustment of cache partition only depends on single and simple target: the miss rate, the IO latency, the throughput or curves which can be measured directly from cache. Unlike Centaur, AC-SSD takes the relationship of different virtual machines inside the applications hosted on hypervisors into consideration, so that it is capable for meeting more complex requirements.

7 CONCLUSION AND FUTURE WORK

We present AC-SSD, a novel tool to allocate the storage and IOPS capacity of SSD cache in the virtualization environment, in order to reduce the job completion time of the Hadoop application. We use the genetic algorithm based approach to calculate a nearly optimal result quickly and efficiently. Moreover, we introduced the closed-loop adaptation inspired by the MAPE-K [8], to automatically detect the change of the workload and trigger the cache adjustment. The evaluation shows that AC-SSD has the significant advantage over the shared cache.

For the future work, we need to detect the change of workload more accurately, rather than using simple sliding window. Moreover, we need to decide when to apply the allocation, aiming to reduce the performance degradation during adjustment, as the change of storage and IOPS capacity will have a slightly impact on the VM involved, which may be unacceptable when the application is handling some critical tasks.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China (2016YFB1000103), the National Natural Science Foundation of China (61572480) and Youth Innovation Promotion Association, CAS (No. 2015088).

REFERENCES

[1] Amazon. 2017. Amazon Elastic MapReduce. (2017). https://aws.amazon.com/elasticmapreduce/
[2] Apache. 2017. Apache Hadoop. (2017). http://hadoop.apache.org/
[3] Apache. 2017. Apache Mahout: Scalable machine learning and data mining. (2017). http://mahout.apache.org/
[4] Dulcardo Arteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, and Ming Zhao. 2016. CloudCache: On-demand Flash Cache Management for Cloud Computing. In *Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST'16)*. USENIX Association, Berkeley, CA, USA, 355–369.
[5] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*. ACM, New York, NY, USA, 164–177.
[6] Axel Busch, Qais Noorshams, Samuel Kounev, Anne Koziolek, Ralf Reussner, and Erich Amrehn. 2015. Automated Workload Characterization for I/O Performance Analysis in Virtualized Environments. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE '15)*. ACM, New York, NY, USA, 265–276.
[7] S. Byan, J. Lentini, A. Madan, L. Pabon, M. Condict, J. Kimmel, S. Kleiman, C. Small, and M. Storer. 2012. Mercury: Host-side flash caching for the data center. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*. 1–12.
[8] Autonomic Computing et al. 2006. An architectural blueprint for autonomic computing. *IBM White Paper* (2006).
[9] Lars George. 2011. *HBase: The Definitive Guide: Random Access to Your Planet-Size Data.* " O'Reilly Media, Inc.".
[10] Ajay Gulati, Ganesha Shanmuganathan, Xuechen Zhang, and Peter Varman. 2012. Demand Based Hierarchical QoS Using Storage Resource Pools. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (USENIX ATC'12)*. USENIX Association, Berkeley, CA, USA, 1–1.
[11] Jacob Gorm Hansen and Eric Jul. 2010. Lithium: Virtual Machine Storage for the Cloud. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10)*. ACM, New York, NY, USA, 15–26. https://doi.org/10.1145/1807128.1807134
[12] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*. IEEE, 41–51.
[13] Intel. 2017. Intel Optane Technology. (2017). http://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html
[14] Jaeho Kim, Donghee Lee, and Sam H. Noh. 2015. Towards SLO Complying SSDs Through OPS Isolation. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15)*. USENIX Association, Berkeley, CA, USA, 183–189.
[15] Ricardo Koller, Ali Jose Mashtizadeh, and Raju Rangaswami. 2015. Centaur: Host-Side SSD Caching for Storage Performance Control. In *Autonomic Computing (ICAC), 2015 IEEE International Conference on*. 51–60. https://doi.org/10.1109/ICAC.2015.44
[16] Lanyue Lu, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2016. WiscKey: Separating Keys from Values in SSD-conscious Storage. In *Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST'16)*. USENIX Association, Berkeley, CA, USA, 133–148.
[17] Tian Luo, Siyuan Ma, Rubao Lee, Xiaodong Zhang, Deng Liu, and Li Zhou. 2013. S-CAVE: Effective SSD Caching to Improve Virtual Machine Storage Performance. In *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques (PACT '13)*. IEEE Press, Piscataway, NJ, USA, 103–112.
[18] Fei Meng, Li Zhou, Xiaosong Ma, Sandeep Uttamchandani, and Deng Liu. 2014. vCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC'14)*. USENIX Association, Berkeley, CA, USA, 133–144.
[19] Microsoft. 2017. HDInsight - Hadoop, Spark and R Solution for the Cloud. (2017). https://azure.microsoft.com/services/hdinsight/
[20] Yongseok Oh, Eunjae Lee, Choulseung Hyun, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2015. Enabling Cost-Effective Flash Based Caching with an Array of Commodity SSDs. In *Proceedings of the 16th Annual Middleware Conference (Middleware '15)*. ACM, New York, NY, USA, 63–74. https://doi.org/10.1145/2814576.2814814
[21] OpenStack. 2017. OpenStack Sahara. (2017). https://docs.openstack.org/developer/sahara/
[22] Mohammad Shamma, Dutch T. Meyer, Jake Wires, Maria Ivanova, Norman C. Hutchinson, and Andrew Warfield. 2011. Capo: Recapitulating Storage for Virtual Desktops. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies (FAST'11)*. USENIX Association, Berkeley, CA, USA, 3–3.
[23] K. Shvachko, Hairong Kuang, S. Radia, and R. Chansler. 2010. The Hadoop Distributed File System. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. 1–10. https://doi.org/10.1109/MSST.2010.5496972
[24] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC '13)*. ACM, New York, NY, USA, Article 5, 16 pages. https://doi.org/10.1145/2523616.2523633
[25] Lei Ye, Gen Lu, Sushanth Kumar, Chris Gniady, and John H. Hartman. 2010. Energy-efficient Storage in Virtual Machine Environments. In *Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '10)*. ACM, New York, NY, USA, 75–84. https://doi.org/10.1145/1735997.1736009