SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

May 2018, Vol. 61 050104:1–050104:17
https://doi.org/10.1007/s11432-017-9401-y

Special Focus on Internetware: Research Progress and Current Trend

IO dependent SSD cache allocation for elastic Hadoop applications

Zhen TANG[1,2], Wei WANG[1,2*], Lei SUN[3], Yu HUANG[4], Heng WU[1,2], Jun WEI[1,2] & Tao HUANG[1,2]

[1]*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing* 100190, *China;*
[2]*University of Chinese Academy of Sciences, Beijing* 100080, *China;*
[3]*Tianjin Massive Data Processing Technology Laboratory, Tianjin Shenzhou General Data Technology Co., Ltd., Tianjin* 300384, *China;*
[4]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210023, *China*

Received 15 November 2017/Revised 29 January 2018/Accepted 29 March 2018/Published online 18 April 2018

Abstract Elastic Hadoop applications consisting of multiple virtual machines (VMs) are widely used to support big data analysis and processing. In this scenario, flash-based solid state drive (SSD) is usually deployed on hypervisors and used as the cache to improve the IO performance. However, existing SSD caching schemes are mostly VM-centric, which focus on the low-level IO performance metrics of individual VMs. They may not lead to the optimized performance of elastic Hadoop applications, i.e., the job completion time (JCT), as the importance of VMs inside the application are different even though they have the similar low-level IO patterns. Considering the IO dependency among VMs and figuring out the importance, which we regard as the application-centric metrics, may potentially better improve the performance. We present IO dependency based requirement model, to characterize the requirement of SSD cache for each VM inside the elastic Hadoop application, and then use it in a genetic algorithm (GA) based approach to calculate the nearly optimal weights of VMs for allocating the per-VM SSD cache space and the capacity of the I/O operations per second (IOPS). Furthermore, we present a tool AC-SSD based on the approach and introduce the closed-loop adaptation to react to continuously changing workloads. The evaluation shows that by using AC-SSD, the JCT is reduced by up to 39% for IO sensitive workloads, up to 29% for continuously changing workloads, and over 12.5% for different scale of data comparing to the shared cache.

Keywords Hadoop, SSD, cache, resource management, virtualization

Citation Tang Z, Wang W, Sun L, et al. IO dependent SSD cache allocation for elastic Hadoop applications. Sci China Inf Sci, 2018, 61(5): 050104, https://doi.org/10.1007/s11432-017-9401-y

1 Introduction

Hadoop[1)] applications are widely used, supporting the enterprises to complete critical tasks. To face different scale of data and user workloads, cloud providers offer virtual machines (VMs) based elastic Hadoop applications. For example, Amazon EC2 provides Amazon EMR[2)], which allows tenants to run big data applications in the cloud. OpenStack foundation provides Sahara[3)] to host the data-intensive

* Corresponding author (email: wangwei@otcaix.iscas.ac.cn)
1) Apache Hadoop. http://hadoop.apache.org/.
2) Amazon Elastic MapReduce. https://aws.amazon.com/elasticmapreduce/.
3) OpenStack Sahara. https://docs.openstack.org/developer/sahara/.

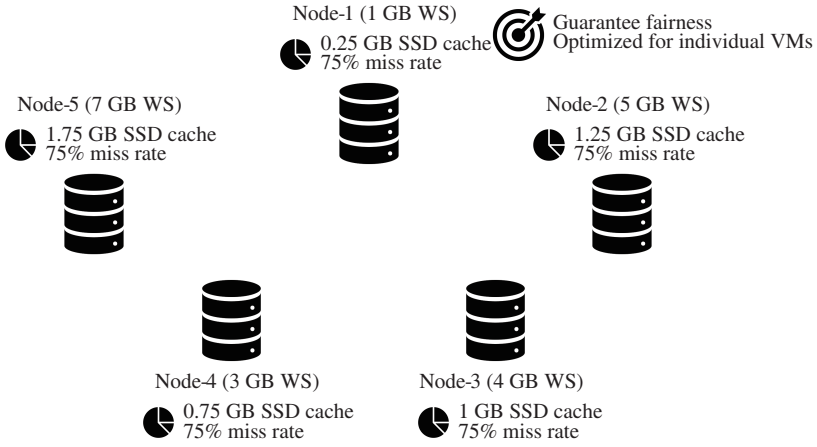

Figure 1 VM-centric SSD cache allocation.

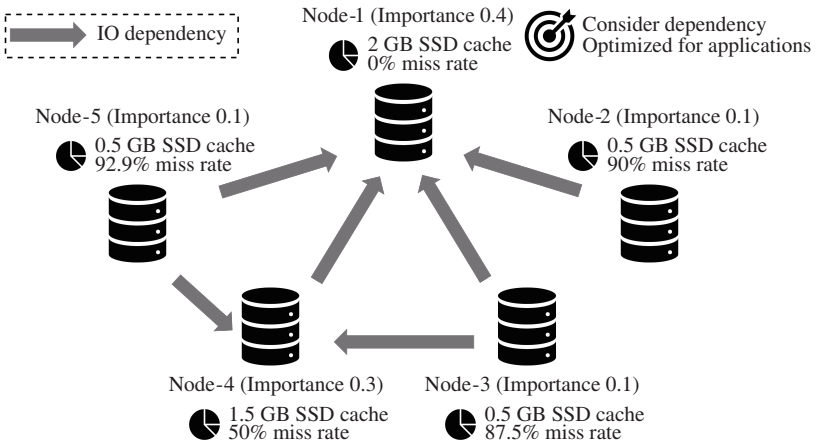

Figure 2 Application-centric SSD cache allocation.

application cluster easily. The most important performance metric of Hadoop applications is the job completion time (JCT). Supported by the HDFS [1], the Hadoop applications are data-intensive. Thus, apart from guaranteeing the data consistency [2,3], improving IO performance will help reduce the JCT.

Flash-based solid state drive (SSD) has significant advantages over traditional hard disk drive (HDD), delivering up to 100K I/O operations per second (IOPS) at low latency. Thus it has been a preferred solution to improving the IO performance [4,5]. Hypervisors, where VMs are hosted in the virtualization environment, are usually attached with a HDD based shared storage to save VM images. In this architecture, it is widely recognized that the IO performance is fundamental to the efficient use of virtualized resources [6,7]. SSD caching is a promising option to improve the IO performance of VMs as well as the hosted data-intensive applications such as elastic Hadoop applications.

Host-side SSD caching has received much attentions these years [8–11]. VMs hosted on the same hypervisor share one physical SSD for caching. The SSD is partitioned to support the per-VM SSD caching. However, existing host-side SSD caching schemes face severe challenges as existing study is mostly VM-centric. The VM-centric approaches manage the SSD cache based on low-level IO performance metrics of individual VMs, thus they can lead to the best IO performance from the view of VM. However, such approaches may not tune for the optimal performance from the application view, i.e., the JCT of Hadoop applications. During the execution of a Hadoop job, the VM requests for data from local storage and other nodes. Due to the data locality, the contribution to the JCT is different for VMs inside the cluster. For the VM centric approach, VMs allocated with high ratio of SSD cache may contribute little to the application-level performance, which leads to the waste of SSD cache.

Figures 1 and 2 demonstrate a simple 5-node Hadoop cluster. Node-1 is the HDFS NameNode while

Node-2 to 5 are DataNodes. Also, there is a background task running on Node-5. We assume that the total SSD cache space is 5 GB and the overall working set is 20 GB. For each node, the ratio of IO operations related to the jobs is different, which we regard as the importance. Furthermore, as the HDFS is deployed on each node, the low-level IO patterns of VMs are similar. The metadata node (Node-1) needs more SSD cache due to IO dependency from other nodes, while the datanode (Node-5) also needs more SSD cache space due to data locality. Figure 1 shows the result of the VM-centric approach, which allocates the SSD cache based on the working sets, while Figure 2 shows that of the App-centric approach, which allocates based on the importance of VMs. Assuming that the average latency of SSD is 0.1x of HDD, the average latency is $0.775 \cdot l_{\mathrm{HDD}}$ for the VM-centric approach and $0.47836 \cdot l_{\mathrm{HDD}}$ for the App-centric approach. Comparing to the VM-centric approach, the App-centric approach reduces the average latency by 38.3%. Thus, using App-centric approaches may lead to the optimal performance.

Additionally, cloud providers usually build elastic Hadoop clusters from multiple VMs and place the VMs using the load balance strategy. VMs from different elastic Hadoop applications may be placed on the same hypervisor, while multiple elastic Hadoop applications may be placed on one hypervisor. In this architecture, the relationships among VMs inside the elastic Hadoop application are also important. Thus, we need the application-centric SSD cache allocation, which allocates the SSD cache according to the requirement of VMs and takes the relationships among VMs into consideration to get the optimal application-level performance.

There are still some limitations to the effective usage of SSD caching to improve the performance of elastic Hadoop applications. The amount of SSD cache which can be allocated to VMs is limited comparing to the working set, so we need to make full use of SSD cache by allocating the per-VM SSD cache according to the demand of the VM. Furthermore, the IOPS capacity of SSD is also limited. There may be potential resource contention of IOPS among VMs hosted on the same hypervisor. Thus, we need to both allocate the cache space and the IOPS capacity for each VM to get the optimal performance. Furthermore, for elastic Hadoop applications, allocating different amount of cache space and IOPS capacity may result in different reduction of the JCT. A straight-forward approach is far from enough for calculating the optimized cache allocation plan. Also, the workloads of different Hadoop application as well as the importance of VMs during the execution of jobs are continuously changing. We regard the importance of VMs as the contribution to the overall performance of the elastic Hadoop application, i.e., the reduction of the JCT when allocating the same SSD cache to the VM. In this scenario, the SSD caching approach must be adaptive to react to continuously changing workloads.

We overcome two technical challenges to meet the goal of optimizing the application-level performance.

(1) How to characterize the importance of VMs from the application view, so as to guide the allocation of cache space and IOPS capacity? For different elastic Hadoop applications, the reduction of the JCT when allocating different cache space and IOPS capacity may be different. We need to figure out the performance metrics related to the JCT to characterize the importance and help find out the optimized SSD cache allocation plan.

(2) How to efficiently allocate the SSD cache space and control the IOPS according to the requirement of VMs and react to continuously changing workloads? As the workloads of elastic Hadoop applications and the requirement of VMs during the execution of jobs are continuously changing, a static or off-line approach is not capable.

This paper makes the following contributions:

(1) We present IO dependency based requirement model, to characterize the importance of VMs inside elastic Hadoop applications, to help figure out the requirement of SSD cache from application view.

(2) Based on this model, we introduce a genetic algorithm (GA) based approach which calculates the nearly optimal weight for each VM and then allocates the capacity of SSD cache space and IOPS based on the weight.

(3) We present AC-SSD, a tool to dynamically allocate the SSD cache based on the approach. We also introduce the closed-loop adaptation to react to continuously changing workloads.

The evaluation shows that by using AC-SSD, the JCT is reduced by up to 39% for IO sensitive workloads, up to 29% for continuously changing workloads, and over 12.5% for different scale of data

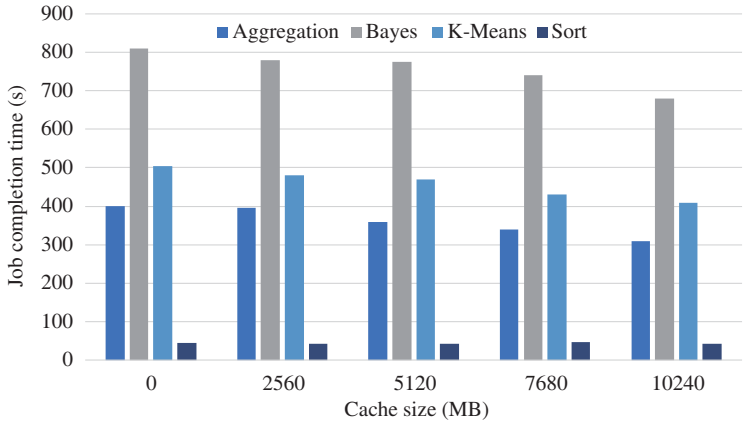

Figure 3 (Color online) Job completion time of 4 benchmarks when allocating different size of SSD cache.

comparing to the shared cache.

A preliminary version of this paper is presented in [12]. The rest of the paper is organized as follows. In Section 2, we discuss two meta level principles we found in elastic Hadoop applications, which motivate us to build the model. In Section 3, we discuss IO dependency based requirement model, which characterizes the importance of VMs inside the Hadoop application. In Section 4, we discuss the GA based approach and the closed-loop adaptation we used to dynamically allocate the SSD cache with the nearly optimal weights. In Section 5, we discuss the implementation of the tool, AC-SSD. In Section 6, we experiment with AC-SSD to validate that it can reduce the JCT of Hadoop applications and react to continuously changing workloads under different data scales. We discuss the related work in Section 7. Section 8 concludes the paper.

2 Meta-level principles

We focus on HDFS [1], YARN [13], and some famous frameworks based on Apache Hadoop, including HBase [14] and Mahout[4)]. As elastic Hadoop applications are data intensive, the JCT can be reduced by improving the IO performance of VMs. Managing per VM SSD cache space and IOPS capacity may be the preferred solution. We have found two meta-level principles of elastic Hadoop applications, which guide the allocation of cache, as shown below:

(1) Relationships among VMs. The importance of each VM inside the elastic Hadoop application may be different due to data locality and the difference of roles. For example, HDFS NameNode and the YARN ResourceManager are the fundamental nodes, which have the significant impact on the JCT, even though they may have the same IO pattern as other nodes with high ratio of IO operations.

(2) Temporal characteristics. The IO patterns for VMs inside the elastic Hadoop application are changing during the execution of the job. For example, the hot spot in the map and reduce stage of the job may be different.

Next, we use two illustrative examples to show the meta-level principles. First, different types of Hadoop applications may have different sensitivity on IO operations due to different behavior of applications and the relationships among VMs inside applications. Furthermore, multiple VMs inside the same application may have the same IO pattern but with different importance. The example is shown in Figure 3. We run 4 representative benchmarks (Aggregation, Bayes, K-Means and Sort) on a 5-VM Hadoop cluster. The x axis represents the total SSD cache space allocated to the application. We observe that when allocating different size of the SSD cache, the JCT of Aggregation and Bayes benchmarks is reduced significantly. However, for CPU sensitive benchmark (Sort), the JCT changes slightly. Thus, allocating more cache space and IOPS capacity to the data intensive applications will make efficient use of SSD cache.

4) Apache Mahout: scalable machine learning and data mining. http://mahout.apache.org/.

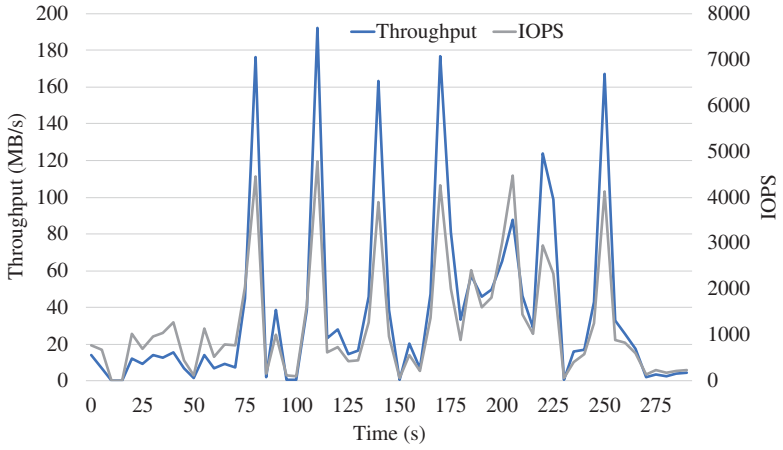

Figure 4 (Color online) The changes of throughput and IOPS during the execution of the K-Means benchmark.

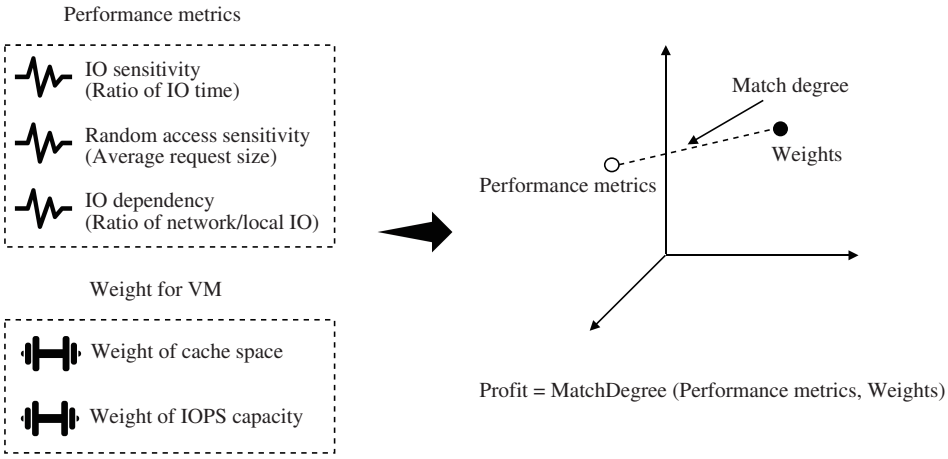

Figure 5 IO dependency based requirement model.

Second, we observe that the IO patterns may be different during the execution of one job. For example, the importance of VMs may be different for the map and reduce stage in one job, which leads to different hot spots in the application. The static SSD caching approach is far from enough to react to continuously changing workloads. The example is shown in Figure 4. We run the K-Means benchmark on a 5-VM Hadoop cluster. The x axis represents the time, while y axes represent the overall throughput and IOPS of the application during the execution of the job. We observe that IO patterns are different for different stages. In this scenario, the static SSD caching approach may be incapable. Thus, we need closed-loop adaptation to react to continuously changing workloads. Also, using a sliding window based monitoring may potentially reduce the waste of SSD cache.

3 IO dependency based requirement model

In this section, we first define the SSD cache allocation problem, and then show the NP completeness of it. Furthermore, we present IO dependency based requirement model, aiming to figure out the SSD cache requirement of each VM inside the application, which we regard as the importance of VMs, as shown in Figure 5. We characterize the importance of VMs from 3 metrics and use the match degree of given weights and performance metrics to figure out the best ratio of SSD cache space and IOPS capacity for each VM.

3.1 Problem definition

We aim to calculate the weights for each VM inside the application and allocate the SSD cache space and IOPS capacity based on the weight. Our goal is to create the connection between the low-level IO performance of VMs (such as the throughput and the IOPS) and the performance of applications (JCT of elastic Hadoop applications), which we regard as the importance of VMs.

There are multiple applications hosted in the cluster. Firstly, we regard the application A as the set of n VMs v, $A_i = \{v_1, v_2, \ldots, v_n\}$. Also, each VM is hosted on one specific hypervisor H. We have $H_i = \{v_1, v_2, \ldots, v_n\}$. Please note that VMs belong to one application may be hosted on different hypervisors, while multiple applications may be hosted on one hypervisor. Thus we need different sets A and H for them. We use the weight w to describe the importance of VMs, which shows the contribution to the JCT for each VM when running the specific workload. Thus, we have the weight vector of cache space and IOPS capacity $W = \{w_1, w_2, \ldots, w_n\}$, where w_i is the weight tuple $\{w_{\text{space}}, w_{\text{IOPS}}\}$ of the VM v_i.

Furthermore, when running different types of workloads, the requirement of SSD cache space and IOPS capacity for VMs may be totally different. Allocating the SSD cache space and IOPS capacity due to the importance and the IO pattern of the VM will help improve the performance and reduce the JCT. We regard the reduction of JCT of the VM when allocating the specific SSD cache space and IOPS capacity as the Profit. Thus, we aim to calculate the weight of SSD cache space and IOPS capacity for each VM to maximize the profit function. The detail of the profit function can be found in Subsection 3.3.

3.2 NP completeness of application-centric SSD cache allocation

Next, we show the NP completeness of application-centric SSD cache allocation problem by showing that the 0-1 knapsack problem is a special instance of the problem. Thus it is reasonable to use the GA based approach to calculate nearly optimal weights of VMs. To allocate per-VM cache space and IOPS capacity so as to minimize the JCT, we need to figure out appropriate ratio of cache space and IOPS capacity for each VM. To simplify the problem, we assume that the cost (usage of SSD cache space and IOPS capacity) and the value (the reduction of JCT) for each VM is constant when running the specific workload.

Then, we add restrictions to the problem 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 amount of the items is $|V| = n$. The cost for each item $C = \{c_1, c_2, \ldots, c_n\}$ is the ratio of cache space and IOPS capacity for each VM. The value for each item $W = \{w_1, w_2, \ldots, w_n\}$ is the reduction of JCT when the VM is selected to allocate specific ratio of cache space and IOPS capacity. The capacity of the knapsack is the total cache space and IOPS capacity of SSD. The objective function is to select VMs in the set V to allocate the cache space and IOPS capacity under the limitation of total capacity (the Σc_i) to get the maximized Σw_i. It is just like the objective function in the 0-1 knapsack problem, i.e., to select items under the restriction of cost C to maximize the total value of the items.

As the 0-1 knapsack problem is NP complete, we show that the application-centric SSD cache allocation problem is also NP complete.

3.3 Profit function

Unlike other types of application (such as the micro service based web application), the data dependency inside the Hadoop application has a strong impact on the JCT, so we need to firstly characterize it. For Hadoop applications, the IO time of a job can be divided into 2 parts.

(1) Local IO operations related to the job. This part can be improved by allocating more SSD cache or IOPS capacity to specific VMs.

(2) Data transfer through network communication, which means one node requests for data stored on other nodes. This part can be improved by allocating more SSD cache space or IOPS capacity to the target VM.

Thus, we use the following metrics to characterize the requirement.

(1) IO Dependency. This indicates the ratio of IO through network communication inside the application. Optimizing the performance of a VM with high IO dependency will result in better reduction of JCT.

(2) Random access intensity. As SSD has advantages over HDD mostly for random access, allocating SSD cache for those with high rate of random access (small average request size) will make full use of the SSD performance. In this model, we regard the random access intensity as the reciprocal of the average size of IO operations.

(3) IO sensitivity of local operations. The ratio of local IO operations is also important for allocating the SSD cache space and IOPS capacity. In this model, we regard the IO sensitivity as the ratio of IO and non-IO operations.

For a specific VM, we use the ratio of the network throughput and the disk IO throughput as the dependency of other VM on it, as shown below. In this paper, we regard the throughput as the bytes of data transferred per second through network or disk I/O.

$$\text{Dependency}(v_i, A) = \sum_{j=1}^{n} \frac{\text{NetworkThroughput}(v_i, v_j)}{\text{IOThroughput}(v_i)}.$$

Thus, for a given weight of the VM, we use the following profit function to characterize the importance of each VM:

$$\text{Profit}(v, w) = \text{MatchDegree}(\text{match}_{\text{IO}}(v, w), \text{match}_{\text{random}}(v, w), \text{match}_{\text{dependency}}(v, w)).$$

Our goal is to find the w to match the 3 metrics, so as to maximize the $\text{Profit}(v, w)$. When using the model in the approach described in Section 4, we use the Euclidean distance as the match degree. We calculate the distance between the weight and the relative performance metrics as the profit. For SSD cache space allocation, we only use the match degree of IO sensitivity and IO dependency:

$$\text{MatchDegree}(\text{match}_{\text{IO}}(v, w), \text{match}_{\text{dependency}}(v, w)),$$

while for IOPS capacity allocation, we only use the match degree of random access intensity and IO dependency:

$$\text{MatchDegree}(\text{match}_{\text{random}}(v, w), \text{match}_{\text{dependency}}(v, w)).$$

4 Approach

In this section, we will introduce the GA based approach we used to calculate the nearly optimal weights, and the closed-loop adaptation inspired by MAPE-K [15] to react to continuously changing workloads.

4.1 Genetic algorithm based SSD cache allocation

Based on IO dependency based requirement model, we use the GA based approach to calculate the nearly optimal weights of cache space and IOPS capacity for each VM. As mentioned before, the elastic Hadoop application consists of multiple VMs. The importance of the master node (HDFS NameNode or YARN ResourceManager) is different from that of the slave nodes. Also, the IO dependency among the nodes inside the cluster has the strong impact on the application-level performance, i.e., the JCT. Thus the IO dependency based requirement model can be used with the GA. This approach can also be used for other types of application with the same architecture and characteristics.

4.1.1 *Structure of the genetic algorithm*

We introduce some critical parameters and the structure of the genetic algorithm. The chromosome indicates the weight tuple $\{w_{\text{space}}, w_{\text{IOPS}}\}$ (which is the ratio of cache space and IOPS capacity) for

each VM, along with the VM specific information used in the algorithm, such as the IO dependency, the random access intensity and the IO sensitivity. Thus, the genome indicates the allocation plan, including all VMs belong to all applications along with the weights of cache space and IOPS capacity. We define the selection operation as randomly selecting genomes by the fitness. We define the crossover operation as swapping random numbers of chromosome of two genomes. We define the mutation operation as changing the weight of randomly selected chromosomes of a genome inside a specific range. After performing the crossover or the mutation operation, the weights will be normalized again.

The structure of the GA we used is the same as general GAs. The initial population is generated with genomes of random values (the ratio of cache space and IOPS capacity). For each iteration, we calculate the fitness and sort the genome by fitness. Then the crossover and mutation operations are performed on genomes to generate the new population. If the count of the iteration exceeds the limitation, we output the best individual with the highest fitness as the result.

4.1.2 *Fitness calculation*

The fitness aims to help predict the JCT and is fundamental to the GA based approach. According to the model mentioned in the Section 3, we calculate the IO sensitivity, random access intensity and IO dependency for each VM. Then, the fitness is calculated by comparing the match degree of the weight with these metrics. The match degree is calculated by summing up the Euclidean distance between the weight and all 3 metrics. We finally adjust the match degree to a positive relative value to the JCT. The algorithm of fitness calculation is shown in Algorithm 1.

Algorithm 1 Fitness calculation

Require: Target genome; CPU time, IO usage (IO time, Bandwidth and IOPS), network throughput for each VM;
Ensure: Fitness for target genome.

Initialization;
for all chromosome c in target genome **do**
  VM c.v $\leftarrow$ VM bound to c;
  Application c.app $\leftarrow$ Application bound to c;
  Weight c.ws $\leftarrow$ Weight of cache space bound to c;
  Weight c.wi $\leftarrow$ Weight of IOPS capacity bound to c;
  IO sensitivity c.s $\leftarrow$ c.v.IOTime/c.v.CPUTime;
  Random access intensity c.r $\leftarrow$ c.v.TotalIOPS/c.v.TotalBandwidth;
  // Calculating the IO dependency
  IO dependency c.d $\leftarrow$ 0;
  for all network throughput t between c.v and VMs in c.app **do**
    c.d $\leftarrow$ c.d + t/c.v.TotalBandwidth;
  end for
end for
// Normalization
for all chromosome c in target genome **do**
  c.s $\leftarrow$ c.s/Total c.s;
  c.r $\leftarrow$ c.r/Total c.r;
  c.d $\leftarrow$ c.d/Total c.d;
end for
// Calculate the match degree
for all chromosome c in target genome **do**
  Match degree c.m $\leftarrow$ $(\text{c.ws} - \text{c.s})^2 + (\text{c.wi} - \text{c.r})^2 + (\text{c.ws} - \text{c.d})^2 + (\text{c.wi} - \text{c.d})^2$;
end for
Fitness $f \leftarrow 4 \times (\text{count}_{\text{VM}}) - \sum \text{c.m}$;
return f.

4.2 Dynamic allocation of cache space and IOPS capacity

For elastic Hadoop applications, the importance of VMs are different when the workloads are continuously changing, or for different stages during the execution of one workload. Thus, we introduce the closed-loop

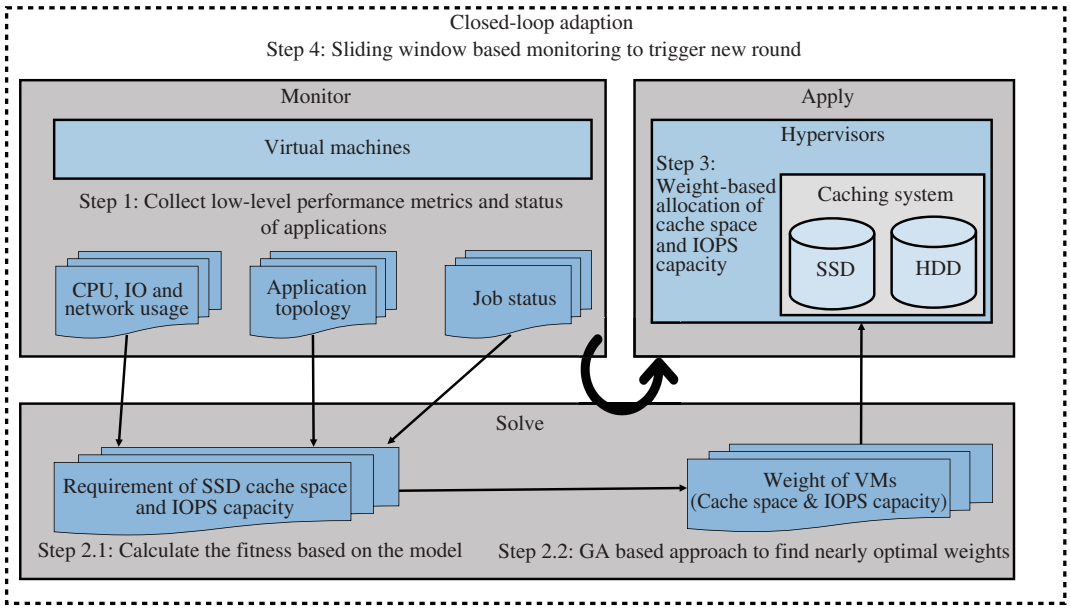

Figure 6 (Color online) Execution of the closed-loop.

adaptation to dynamically allocate the cache space and IOPS capacity. The closed-loop adaptation we used consists of 3 steps: monitoring, solving and applying.

4.2.1 *Three levels of monitoring*

We monitor status and the performance of elastic Hadoop applications from the hypervisor, the application and the VM level, to gather information for GA based approach.

For the hypervisor level, we deploy agent on each hypervisor to monitor the cache status, including the overall usage and statistics of the SSD, and the usage of cache space and IOPS capacity for each VM.

For the application level, we deploy agent on each VM to figure out the elastic Hadoop applications it belongs to, and build the topology for each application by monitoring the interactions of VMs via network communication. Furthermore, we detect the role of VMs by filtering the list of running processes. We also trace the status of the job execution to help find the important VMs.

For the VM level, we use the agent deployed on each VM to monitor the CPU time, the network throughput and the IO pattern, which will be used by GA based approach. The agent monitors the CPU time to get the IO and non-IO time, to help calculate the IO sensitivity. The agent monitors the inbound and outbound network throughput for each VM inside the elastic Hadoop application, to help calculate the IO dependency of VMs. The agent monitors the IO throughput and the average size of IO requests for each VM, to help calculate the random access intensity.

4.2.2 *Execution of the closed-loop*

The execution of the closed-loop is shown in Figure 6, which consists of 4 main steps. For one round of the closed-loop adaptation, AC-SSD firstly monitors the low-level performance of each VM, including the CPU usage, the network communication and the IO pattern. Also, AC-SSD characterizes the topology for each application and then traces the status of job execution (Step 1). Then, the information will be used by the GA based approach to calculate fitness based on IO dependency based requirement model (Step 2.1) and then get the nearly optimal weights of cache space and IOPS capacity (Step 2.2). Finally, AC-SSD allocates per-VM cache space and IOPS capacity (Step 3). Furthermore, the interval of the sliding window based monitoring and the execution of the closed-loop is one minute (Step 4). We observe that the interval is sufficient to achieve balance between the overhead and the accuracy of monitoring.

5 Implementation

In this section, we introduce the tool, AC-SSD, which provides the allocation of per-VM SSD cache space and IOPS capacity based on IO dependency based requirement model and GA. Furthermore, AC-SSD introduces closed-loop adaptation to react to continuously changing workloads. AC-SSD monitors the elastic Hadoop application to gather information from the VM, the hypervisor and the application level. The data will be used by the GA based approach to figure out the requirement of cache space and IOPS capacity for each VM and then calculate the nearly optimal weights. Finally the allocation plan will be applied by resizing the cache and controlling the IOPS capacity for each VM.

AC-SSD consists of 5 main components: the controller, the monitor, the solver, the executor and agents deployed on each hypervisor and VM, to support the closed-loop adaptation and the GA based approach.

(1) The controller triggers the execution of whole closed-loop adaptation. It communicates with the monitor, solver, executor and agents via the RESTful API. It also starts the new round of closed-loop adaptation for the interval of one minute.

(2) The monitor gathers performance data of CPU, disk IO and network communications from the agents deployed on VMs to help figure out the requirement of cache space and IOPS capacity. It also parses the process list of each VM to build the topology of the elastic Hadoop application and detect the role of each VM, and then traces the status of running jobs. Additionally, it gathers data from the agent deployed on each hypervisor to get the resource supply and the limitation of SSD cache space and IOPS capacity. The gathered performance data will then be used by solver to calculate the nearly optimal results.

(3) The solver calculates weights of cache space and IOPS capacity for each VM inside each elastic Hadoop application by using the GA based approach. Finally, an allocation plan will be generated for the executor to apply.

(4) The executor sends messages to the agent on each hypervisor to apply the plan by allocating per-VM SSD cache and IOPS capacity.

(5) The agent is deployed on each VM and hypervisor. For VMs, the agent aims to gather performance data, parse the process list, and figure out the role of VMs inside the elastic Hadoop application. For hypervisors, the agent resizes the per-VM SSD cache and controls the IOPS capacity for VMs hosted on them.

We implement AC-SSD based on Xen [16], a widely used hypervisor. The controller, monitor, solver, executor and agents are implemented in Java. We also modify the default block driver of QEMU and add an LRU based SSD cache. We use separate file stored on SSD to maintain the per-VM cache. In order to allocate IOPS capacity, we use Linux cgroup to change the weight of IOPS for different VMs. We use the sysstat package and the Linux proc filesystem to monitor the performance of CPU, disk IO and network communication. Furthermore, we use the library libvirt to manage the lifecycle of VMs and safely suspend the VM before allocating the cache space and IOPS capacity.

6 Evaluation

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

(1) Is AC-SSD better than the shared cache approach for elastic Hadoop applications with the JCT metric?

(2) Can AC-SSD reallocate the cache space and IOPS capacity according to the dynamic workloads?

(3) Is AC-SSD effective for different data scales of workloads?

Table 1 Experiment setup[a)]

Cluster		Hypervisor	
Cluster #	Node #	Hypervisor #	Node #
1	**1**, 3, 6, 11, 16	1	**1**, 2, 3, 4, 5
2	2, 4, **7**, 9, 12, 14, 15, 17, 19, 20	2	6, **7**, 8, 9, **10**
3	5, 8, **10**, 13, 18	3	11, 12, 13, 14, 15
		4	16, 17, 18, 19, 20

a) The node with bold text is the master node of the cluster.

6.1 Experiment design

6.1.1 *Experiment setup*

Our experimental environment includes four Inspur blade servers, each has 8 cores and 16 GB memory with a 240 GB SSD and a 300 GB SAS HDD. Besides, there is a shared storage based on the HDDs. All servers run on CentOS 7.2 guest OS under Xen hypervisor, and we deploy 3 VM-based clusters with three tenants, as show in Table 1.

The clusters 1, 2 and 3 consist of 5 VMs, 10 VMs and 5 VMs, respectively. The master node of cluster 1 does not interfere with the others, and the master node of clusters 2 and 3 are placed on the same hypervisor. In this scenario, tenants usually build VM-based Hadoop clusters with fairness strategy. Thus, we place the three clusters fairly on four hypervisors to simulate the strategy, which means each application consists of VMs hosted on all hypervisors. We deployed Hadoop 2.7.2 for each cluster, while the replication of HDFS is set to 2.

For the GA based approach, we set the mutation rate to 20%, crossover rate to 50%, and the range of mutation to 20%. Moreover, we set the population size to 100 and the max generation to 3000. Thus the algorithm can end in less than 1 s.

6.1.2 *Benchmarks*

We aim to simulate the scenarios in the real world and choose benchmarks from three widely used open source projects: Apache Hadoop, Apache HBase, and Apache Mahout, as shown below:

(1) The IO sensitive benchmark. We select the TestDFSIO benchmark from HDFS. When the tenants are running cross-cluster data synchronization tasks, the IO pattern is just like the IO sensitive benchmarks.

(2) Parallel algorithms. We select the Sort, Terasort and Wordcount benchmarks from Hadoop. These workloads are similar to the data processing tasks with big input and small output.

(3) SQL. We select the Aggregation, Join and Scan benchmarks from HBase. The pattern is similar to the execution of query on a big database.

(4) Data mining and websearch. We select the Bayes, K-Means and Pagerank benchmarks from Mahout. The pattern is similar to the data analysis tasks, such as the recommendation or the mining of user behavior.

The SQL benchmarks (Aggregation, Join and Scan) and the Pagerank benchmark are from HiBench [17]. We use the small data scale of HiBench for evaluating the effect of SSD cache allocation and the closed-loop adaptation. We observe that the working sets for benchmarks with small data scale are mostly over 200 MB for each VM, which is reasonable to be used in the experiment.

6.2 The effect of GA based SSD cache allocation

In this subsection, we aim to evaluate AC-SSD and compare the application-level IO performance and JCT to the shared SSD cache, i.e., the VMs deployed on the same hypervisor share the entire SSD cache space. We use three clusters as mentioned in Subsection 6.1.1, and run workloads simultaneously on them. We set the total SSD cache space to 2560 MB for all VMs. Thus, the average cache space for one VM is 128 MB, and the size of the shared cache for each hypervisor is 640 MB. We do not allocate the IOPS capacity when using the shared cache.

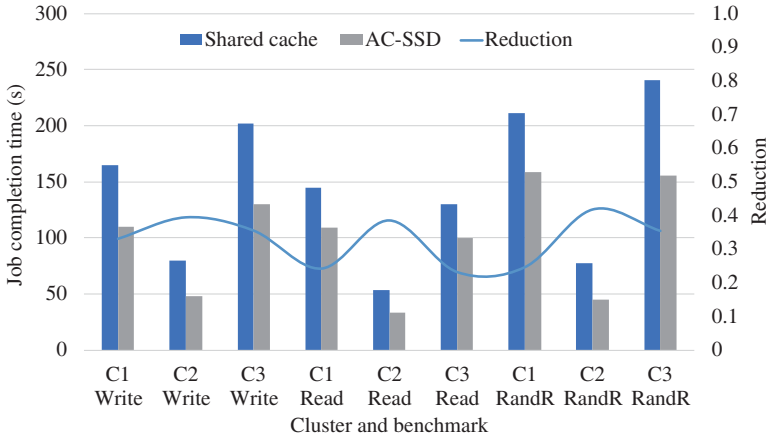

Figure 7 (Color online) Job completion time of TestDFSIO for 3 clusters under 3 modes.

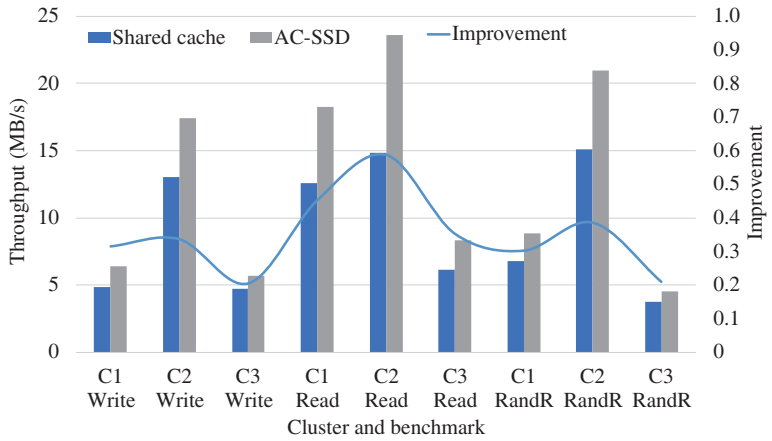

Figure 8 (Color online) Throughput of TestDFSIO for 3 clusters under 3 modes.

We firstly run the TestDFSIO benchmark to confirm that AC-SSD can better reduce the application-level throughput and JCT comparing to the shared cache. We perform the write, read and random read tests on 10 files. The size of each file is 256 MB. Figure 7 compares the reduction of JCT to the shared cache for all three clusters. The results show that the JCT of IO sensitive workloads is reduced by up to 39% and 31% in average. Thus the cluster can run 45% more IO sensitive workloads comparing to the shared cache. Figure 8 compares the application-level IO performance. By using AC-SSD, the throughput is improved by up to 57%, and 35% in average. As AC-SSD continuously detects the important VMs and reallocates cache space and IOPS capacity, it works better for larger clusters and for the workloads with high ratio of random IO operations, especially for reads.

Secondly, we use several real world applications to evaluate AC-SSD. We run the parallel algorithms, the SQL benchmarks and the data mining benchmarks on all three clusters and compare AC-SSD with the shared cache. Figure 9 shows the reduction of JCT. We observe that when facing the data intensive workloads, such as the three SQL benchmarks, the JCT is reduced by up to 19%. However, for those CPU sensitive workloads or workloads with low network communications, such as the Wordcount or Sort, the JCT is similar to the one using the shared cache. Furthermore, for those workloads with high ratio of network communication and IO throughput, such as the Bayes and K-Means benchmarks, the JCT is reduced by 27% when using AC-SSD. In average, by using AC-SSD, the JCT is reduced by 14.3% for cluster 1, 17.8% for cluster 2, and 16% for cluster 3. It shows the effect of GA based approach and the closed-loop adaptation as AC-SSD considers the relationships among VMs and reacts to the change of IO patterns quickly. Similarly to the evaluation on IO sensitive workloads, the reduction of JCT for larger clusters is a little better, as AC-SSD characterizes the importance of VMs dynamically and reallocates

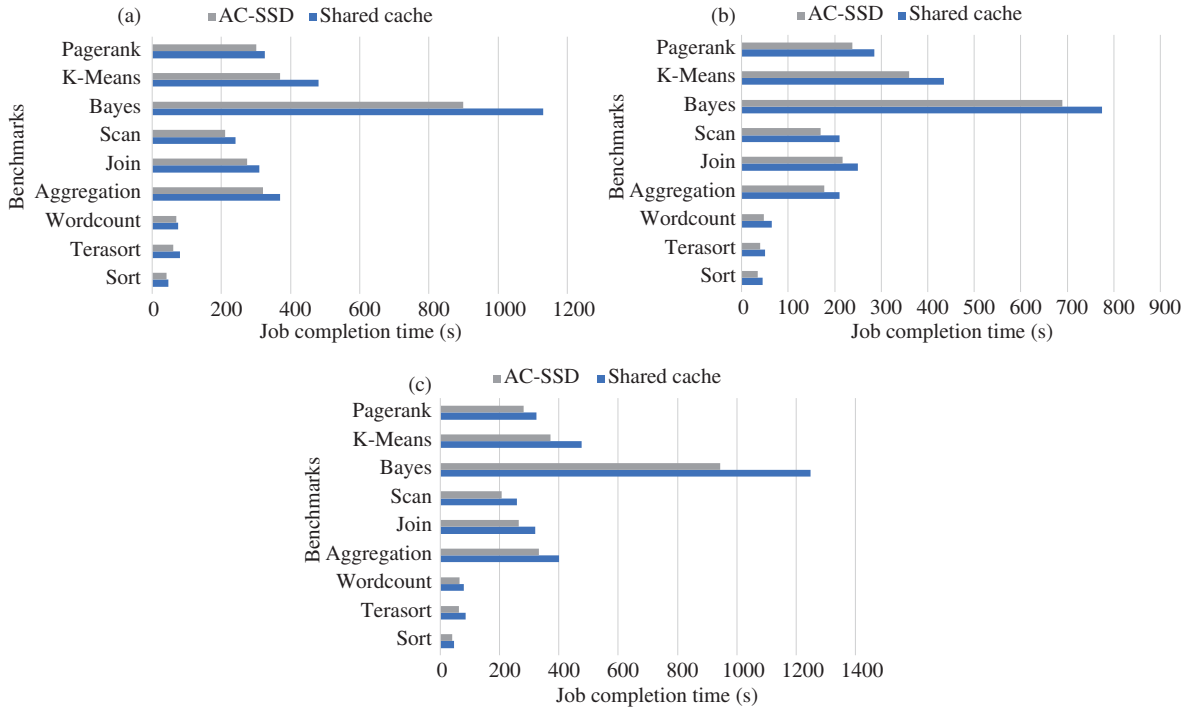

Figure 9 (Color online) Job completion time for 3 clusters running different workloads. (a) Cluster 1; (b) cluster 2; (c) cluster 3.

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	K-Means	Pagerank	K-Means
9	Wordcount	K-Means	Terasort

the cache space and IOPS capacity quickly. The results of clusters 1 and 3 are different because there may be resource contention of IOPS between the master nodes of clusters 2 and 3 as they are placed on the same hypervisor. However, AC-SSD allocates more the cache space and IOPS capacity to the master nodes of clusters 2 and 3, which lead to the reduction of JCT comparing to the shared cache.

Additionally, the overhead of the sliding window based monitoring bringing by the agents is around 3%, which is considerable for the cloud platform.

6.3 Using AC-SSD in the dynamic environment

We change the type of workloads to simulate the scenario that multiple tenants in the cloud platform run workloads as they need. We aim to evaluate AC-SSD and confirm that AC-SSD can react to continuously changing workloads. We change the type of workloads running on each cluster by the order listed in Table 2. Each workload will execute for 10 min, or at least once, and we record the average JCT by using AC-SSD and shared cache. We still use the same three clusters as mentioned in Subsection 6.2, consisting of 20 VMs in total.

Figure 10 compares the JCT of 3 clusters running continuously changing workloads. When AC-

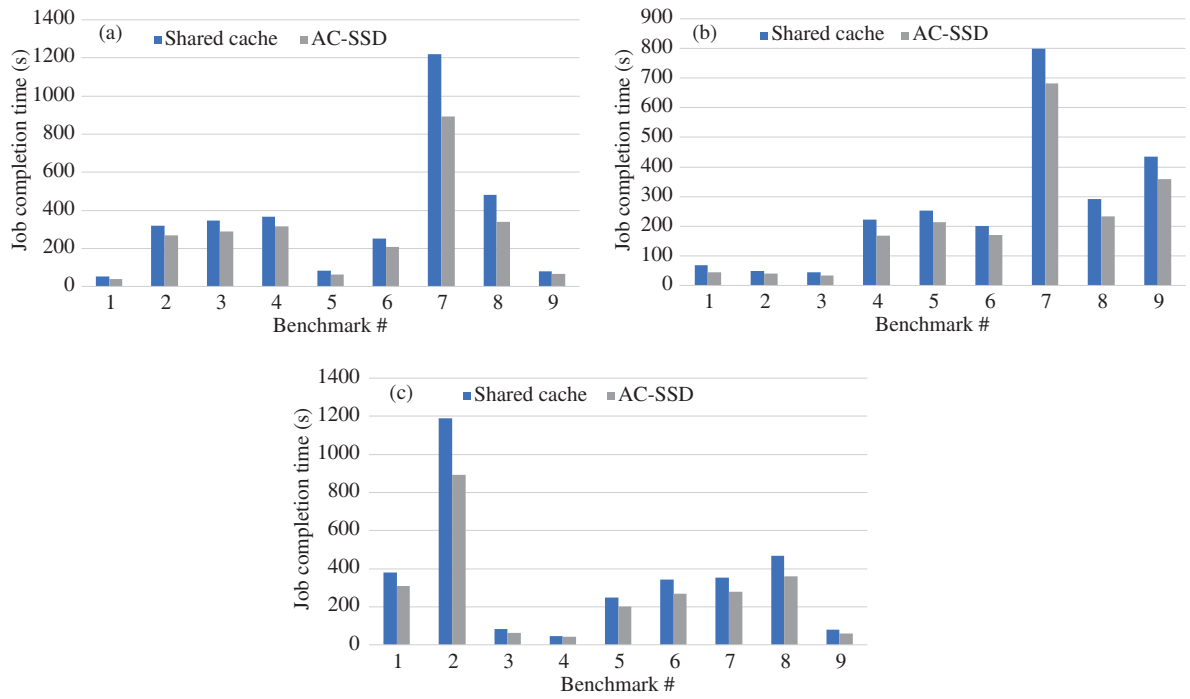

Figure 10 (Color online) Reduction of JCT in the dynamic environment. (a) Cluster 1; (b) cluster 2; (c) cluster 3.

SSD detects the change of workloads, it will quickly reallocate the cache space and IOPS capacity. Thus, for those clusters running CPU sensitive workloads, AC-SSD react quicker than the shared cache, which shows the up to 29% reduction of JCT. For example, when the workload of cluster changes from Terasort to Scan (from group 5 to 6), the reduction of JCT changes significantly. Thus, AC-SSD improves the application-level performance and the utility of SSD by resizing the cache space and IOPS capacity once the change of workloads is detected. In average, by using AC-SSD, the JCT is reduced by 19% for cluster 1, 22% for cluster 2, and 18% for cluster 3. The improvement of the performance is better than the result when facing the static workloads, and is better for big clusters.

Additionally, AC-SSD will allocate more cache space and IOPS capacity than the shared cache if the target cluster is facing the heavy IO sensitive workloads and network interactions (such as the K-Means and the Bayes benchmarks). However, for the shared cache, the cache needs to be warm up to give the preferential treatment to the IO sensitive workloads. Besides, AC-SSD reacts quickly than the shared cache when the type of workloads changes between CPU sensitive and IO sensitive.

Thus, AC-SSD can react to continuously changing workloads, and shows better performance when the cluster is big or the type of workloads changes frequently between IO sensitive and CPU sensitive.

6.4 The effectiveness for different data scales

We run micro benchmarks of real world algorithms and change the data scale to find out if AC-SSD works well for different data scales. We evaluate AC-SSD in 1.5x and 2x data scale than we use in Subsections 6.2 and 6.3. We run the benchmarks on a 5-node Hadoop cluster and still compare the JCT to the shared SSD cache.

Figure 11 shows the JCT of benchmarks under the small and the large data scale. The results show that by using AC-SSD, the JCT is reduced by 14.6% in average, and up to 19.8% for the 1.5x data scale. While for the 2.0x data scale, AC-SSD reduces the JCT by 12.5% in average, up to 18.4%. Comparing to the results in Subsections 6.2 and 6.3, the performance improvement is reduced, as for the big data scale, more IO operations miss the cache, and the average IO latency increases. However, the results show that AC-SSD still works efficiently for different data scales.

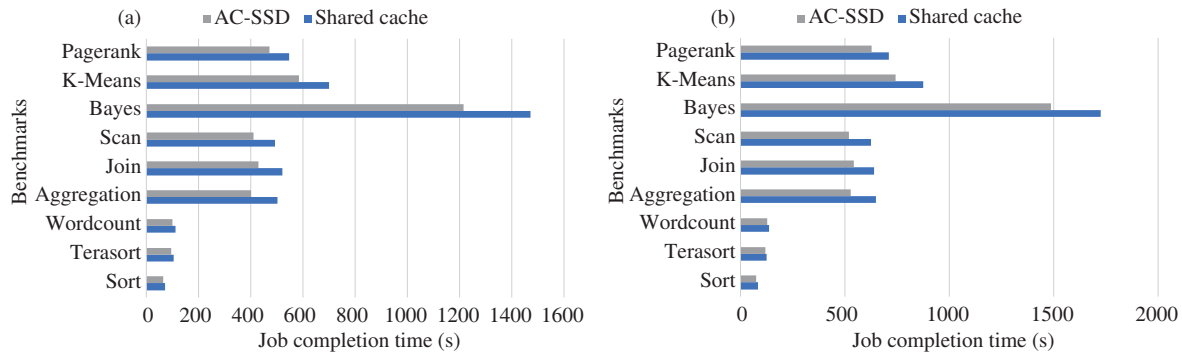

Figure 11 (Color online) Reduction of job completion for different data scale. (a) Small; (b) large.

6.5 Lessons learned

We observe that AC-SSD can reduce the JCT significantly for elastic Hadoop applications comparing to the shared cache. It performs better for the workloads with high ratio of random IO operations. Furthermore, it is capable to be used in multi-tenants cloud platforms as it allocates both the cache space and IOPS capacity.

We use small clusters and SSD based caching system to evaluate AC-SSD. However, AC-SSD can easily be used with big clusters consisting of hundreds of VMs as the genetic algorithm can end in a short time. Additionally, AC-SSD can also be applied to other media with similar characteristics as SSD, such as NVMe SSD or Intel Optane[5)].

Moreover, the GA based approach can also be applied to other types of applications consisting of multiple VMs, as it takes the relationships among VMs into consideration. The GA may be useful for those applications with the same architecture as the elastic Hadoop application, i.e., consisting of one master node and several slave nodes, with IO dependency among them. For other types of applications with different architecture, such as the micro service based transactional web applications (with heavy IO on database and low IO dependency among nodes) and the decentralization applications, the GA with the IO dependency based requirement model may not be the best choice. The application-level performance need to be characterize using different metrics, and the priori rules can be used to simplify the calculation.

Furthermore, there are some limitations for AC-SSD. Firstly, the contention of IOPS capacity cannot be ignored if VMs hosted on the same hypervisor are facing heavy random IO workloads. For example, if master nodes of multiple Hadoop applications are placed on the same hypervisor, there may be strong impact on JCT. However, this problem cannot be solved by allocating the IOPS capacity. The replacement of VMs is needed to avoid the contention. Secondly, we perform the monitoring from the VM, the application and the hypervisor level only by using non-intrusive approach. We do not instrument the Hadoop application to deeply trace the execution of jobs, which means AC-SSD cannot change the allocation plan accurately due to the behavior of jobs. Thus, there may be performance degradation when the type of workloads changes rapidly inside one sliding window of monitoring.

7 Related work

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

We use the closed-loop adaptation and GA based algorithm before in TA-SSD [18], aiming to allocate per-VM SSD cache space to reduce the response time of transactional web applications. However, TA-SSD cannot detect the requirement of IOPS capacity of VMs. Also, the metrics it uses highly depend on the behavior of the transactional applications, which may not be capable for the elastic Hadoop application.

5) Intel Optane Technology. http://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html.

CloudCache [8] proposes a new cache demand model based on the reused working set, to satisfy the actual cache demand of the workload so as to balance the IO performance and the wear out of the SSD. Compared to CloudCache, the requirement model used by AC-SSD focuses on the metrics from the high level, rather than the working set. Thus AC-SSD may lead to better performance when facing the applications consisting of multiple VMs.

Capo [19] applies different cache policies for different directories on the file system of virtual desktops to optimize the durability, reduce the load on shared storage, and improve the performance. Furthermore, it uses local disk, not the SSD, as the persistent cache. Unlike Capo, AC-SSD focuses on the application-level performances and allocates the SSD cache space and IOPS capacity for each VM considering the characteristics of SSD. Thus, it is capable to be used with elastic Hadoop applications.

S-CAVE [20] considers the effectively used cache space (rECS) to figure out the cache demand of one VM. Furthermore, it allocates the cache space due to the estimated rECS and the heuristics based on it. Unlike S-CAVE, AC-SSD uses the application-centric metrics to characterize the requirement of cache space and IOPS capacity for VMs inside elastic Hadoop applications.

vCacheShare [21] uses a cache utility model concerning the cache hit rate, reused intensity, disk latency and read ratio. Unlike vCacheShare, AC-SSD focuses on the metrics related to the application-level performance, rather than using the low-level performance metrics of individual VMs.

Centaur [10] uses curve to guide the adjustment of partitioned SSD cache. The miss rate curves (MRC) and the latency curves are used to meet different QoS targets. However, Centaur mostly focuses on the low-level IO performance of VMs and the status of SSD cache, not the application-level performance metrics. Unlike Centaur, AC-SSD considers the relationship among VMs and the temporal characteristics so that it is capable for meeting more complex requirements.

8 Conclusion

We present IO dependency based requirement model to characterize the importance of VMs inside the elastic Hadoop application. Based on the model, we present AC-SSD, a novel tool to allocate per-VM SSD cache space and IOPS capacity in order to reduce the JCT of the elastic Hadoop application. We use the GA based approach to calculate the nearly optimal allocation plan and introduce the closed-loop adaptation to react to continuously changing workloads. The evaluation shows that AC-SSD can significantly improve the performance of elastic Hadoop applications comparing to the shared cache.

Acknowledgements This work was supported by National Key Research and Development Program of China (Grant No. 2016YFB1000103), National Natural Science Foundation of China (Grant No. 61572480), Tianjin Massive Data Processing Technology Laboratory, and Youth Innovation Promotion Association, Chinese Academy of Sciences (Grant No. 2015088).

References

1 Shvachko K, Hairong K, Radia S, et al. The hadoop distributed file system. In: Proceedings of the 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, 2010
2 Wei H F, De Biasi M, Huang Y, et al. Verifying pipelined-RAM consistency over read/write traces of data replicas. IEEE Trans Parallel Distrib Syst, 2016, 27: 1511–1523
3 Wei H F, Huang Y, Lu J. Probabilistically-atomic 2-atomicity: enabling almost strong consistency in distributed storage systems. IEEE Trans Comput, 2017, 66: 502–514
4 Kim J, Lee D, Noh S H. Towards slo complying SSDs through ops isolation. In: Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST), Santa Clara, 2015. 183–189
5 Lu L, Pillai T S, Arpaci-Dusseau A C, et al. WiscKey: separating keys from values in SSD-conscious storage. In: Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST), Santa Clara, 2016. 133–148
6 Hansen J G, Jul E. Lithium: virtual machine storage for the cloud. In: Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC), Indianapolis, 2010. 15–26
7 Ye L, Lu G, Kumar S, et al. Energy-efficient storage in virtual machine environments. In: Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE), Pittsburgh, 2010. 75–84

8 Arteaga D, Cabrera J, Xu J, et al. CloudCache: on-demand flash cache management for cloud computing. In: Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST), Berkeley, 2016. 355–369

9 Byan S, Lentini J, Madan A, et al. Mercury: host-side flash caching for the data center. In: Proceedings of the 28th Symposium on Mass Storage Systems and Technologies (MSST), San Diego, 2012

10 Koller R, Mashtizadeh A J, Rangaswami R. Centaur: host-side SSD caching for storage performance control. In: Proceedings of IEEE International Conference on Autonomic Computing (ICAC), Grenoble, 2015. 51–60

11 Oh Y, Lee E, Hyun C, et al. Enabling cost-effective flash based caching with an array of commodity SSDs. In: Proceedings of the 16th Annual Middleware Conference, Vancouver, 2015. 63–74

12 Tang Z, Wang W, Huang Y, et al. Application-centric SSD cache allocation for hadoop applications. In: Proceedings of the 9th Asia-Pacific Symposium on Internetware, Shanghai, 2017

13 Vavilapalli V K, Murthy A C, Douglas C, et al. Apache hadoop yarn: yet another resource negotiator. In: Proceedings of the 4th Annual Symposium on Cloud Computing (SoCC), Santa Clara, 2013

14 George L. HBase — The Definitive Guide: Random Access to Your Planet-Size Data. Sebastopol: O'Reilly Media, 2011

15 Agarwal G, Shah R, Walrand J, et al. An architectural blueprint for autonomic computing. IBM White Paper. 2013. http://users.cs.fiu.edu/~sadjadi/Teaching/Autonomic%20Grid%20Computing/CIS-6612-Summer-2006/AC-Blueprint-WhitePaper-V7.pdf

16 Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP), New York, 2003. 164–177

17 Huang S S, Huang J, Dai J Q, et al. The HiBench benchmark suite: characterization of the mapreduce-based data analysis. In: Proceedings of the 26th International Conference on Data Engineering Workshops (ICDEW), Long Beach, 2010. 41–51

18 Tang Z, Wu H, Sun L, et al. Transaction-aware SSD cache allocation for the virtualization environment. In: Proceedings of the 12th International Symposium on Service-Oriented System Engineering Workshops (SOSEW), Bamberg, 2018. 174–179

19 Shamma M, Meyer D T, Wires J, et al. Capo: recapitulating storage for virtual desktops. In: Proceedings of the 9th USENIX Conference on File and Stroage Technologies (FAST), San Jose, 2011

20 Luo T, Ma S Y, Lee R B, et al. 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), Edinburgh, 2013. 103–112

21 Meng F, Zhou L, Ma X S, et al. 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), Philadelphia, 2014. 133–144