Enabling Accurate Performance Isolation on Hybrid Storage Devices in Cloud Environment

Chuanwen Wang, Diansen Sun, Yunpeng Chai, and Fang Zhou
Key Laboratory of Data Engineering and Knowledge Engineering, Ministry of Education of China
School of Information, Renmin University of China, Beijing, China
{wangchuanwen, dssun, ypchai, xinxizhouf}@ruc.edu.cn

***Abstract*—Hybrid storage devices have been widely deployed in cloud platforms to provide cost-efficient storage services. So the performance isolation on hybrid storage devices is important and necessary to ensure the quality-of-service (QoS) of cloud tenants. However, existing performance isolation techniques (e.g., the widely used *cgroup*) cannot provide accurate performance isolation on hybrid devices, usually leading to an unexpected throughput shrink than the user-specified value. In this paper, we propose a new Self-Adaptive Accurate Throttling (SAAT) method to achieve accurate performance isolation on hybrid storage devices. SAAT can give an accurate estimation of the throughput shift of existing solutions periodically according to the latest status of hybrid storage devices, and then adjust the assigned throughput of *cgroup* to achieve the accurate performance isolation. The experimental results exhibit that SAAT outperform the original cgroup method for 3.42 times on the difference of average throughput (*DAT*) between the target and the measured one, i.e., reducing *DAT* from 6.56% to 1.94%; in the meanwhile, SAAT reduces the *Average Jitter* of practical throughput from 6.62% to 5.25%.**

***Index Terms*—Hybrid Storage; Performance Isolation; Multi-Tenant; Control Group; Cloud Storage**

I. Introduction

Hybrid storage devices can take both the capacity and the cost advantages of slow devices (e.g., disks) and the performance advantage of fast devices (e.g., SSDs), so they are widely deployed in cloud platform ((e.g., Amazon Web Services, Microsoft Azure, etc.) for cost-efficient storage services. When multiple users share a hybrid storage device, the performance isolation is an inevitable issue.

One of the most widely used performance isolation tools is Linux Control Groups (*cgroup*) [1]. By setting the upper bound of throughput, *cgroup* ensures the achieved throughput will not exceed the specified value. When tenants have enough requests, the practical I/O bandwidth of tenants is basically equal to the limitation of cgroup, i.e., cgroup can be used to accurately divide the I/O resources among tenants. Cgroup usually works very well when accessing single-media storage devices (e.g., disks or SSDs).

However, the performance isolation on hybrid devices is not accurate for cgroup - the measured throughput of tenants is unstable and obviously lower than the specified value (see the following Fig. 1 (a)). The reason mainly lies in that too many requests focus on the slow device of the hybrid device in some time periods when the hit rates become low and the slow device cannot supply enough bandwidth (see Section II for more). Thus, in order to achieve the user-specified bandwidth, we need to promote users' throughput limitation settings to appropriate values. Yet how to estimate the exact throughput shift to achieve the target throughput for tenants is a problem.

In this paper, we propose a new Self-Adaptive Accurate Throttling (SAAT) method to achieve accurate performance isolation on hybrid storage devices. Based on modeling the behavior patterns inside the hybrid device, SAAT can give an accurate estimation about the throughput shift of existing solutions, and update the estimation periodically according to the latest records to achieve the accurate performance isolation. We have made a series of experiments driven by enterprise I/O workloads to evaluate the effects of SAAT coupled with cgroup. The results exhibit that SAAT can reduce the difference between the specified throughput and the measured one to 1.91%, while the original cgroup leads to a 6.56% difference; SAAT also reduces the average jitter of practical bandwidth from 6.62% to 5.25%.

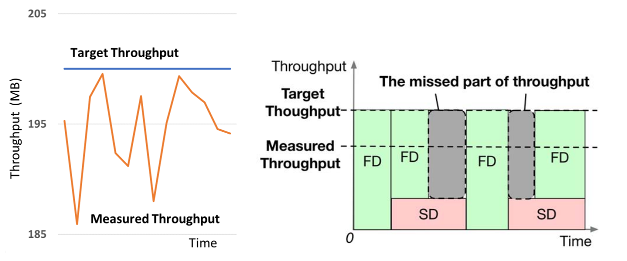
II. Background and Motivation

Hybrid storage refers to the combination of different types of storage devices, shielding internal details. A common approach is to combine different devices into one logical device at the OS level (e.g., Facebook's Flashcache [2]). In recent years, some studies [3]–[5] focused on applying SSD-based cache in hybrid storage to achieve high I/O speed. For the SSD or NVM devices which usually have the write endurance limitation [6], some other studies proposed some write-efficient cache replacement algorithms to balance the cache hit ratios and the cache device lifetime [7]–[10].

The most popular combination of hybrid storage is to set a Fast Device (FD) upon a Slow Device (SD) as a cache to take the advantage of both FD (i.e., fast) and SD (i.e., low-cost and capacity). We can set an SSD as the cache of an HDD, or even set the emerging Non-Volatile Memory (NVM) devices as the cache of an SSD for the performance boost. However, when we use *cgroup* to set an upper bound for a process group's throughput or IOPS, we observe that there is a gap between the target and the measured throughput when performing I/O requests constantly on hybrid devices, as shown in Fig. 1 (a).

The reason why the throughput throttling on hybrid devices is not accurate lies in that SD in a hybrid device undertakes most of the requests in some time period due to temporal low hit ratios and it cannot supply large enough bandwidth to satisfy the target throughput of tenants, as Fig. 1 (b) illustrates.

2576-6996/18/$31.00 ©2018 IEEE
DOI 10.1109/ICCD.2018.00090



(a) A fragment of the target and the measured throughputs. (b) This temporal low hit ratios lead to the throughput missing.

Fig. 1. The throughput shift problem of *cgroup* on hybrid storage devices' performance isolation and its reason.

In this case, although FD can supply higher throughput than the target one to promote the average bandwidth, the bandwidth of FD is strictly throttled by cgroup. If we can promote the throughput limitation to an appropriate value, the average throughput of the hybrid device can reach the specified value of users.

Therefore, a possible solution of accurate throughput throttling is to promote the assigned throughput throttling of cgroup on the hybrid devices properly to make the average total throughput of FD and SD reach the target specified by users. However, the internal behaviors in a hybrid device are complex and the status is constantly changing (e.g., cache hit rates, working time overlap of FD and SD, etc.), so it is a challenging problem to achieve accurate throughput throttling on hybrid devices.

III. Self-Adaptive Accurate Throttling

In order to achieve accurate performance isolation on hybrid storage devices, we propose a new method called Self-Adaptive Accurate Throttling (SAAT).

As Fig. 2 shows, *cgroup* is used to throttle the read and the write bandwidth for tenants. Assuming the target read and write throughput values are Th^T_{R1}, Th^T_{W1}, ..., Th^T_{Rn}, and Th^T_{Wn} for n tenants, the original cgroup adopts them as the assigned throughput throttling values. By the throttling of cgroup, the practically measured throughput values of these tenants (i.e., Th^M_{R1}, Th^M_{W1}, ..., Th^M_{Rn}, and Th^M_{Wn}) are usually smaller than the assigned values (i.e., Th^A_{R1}, Th^A_{W1}, ..., Th^A_{Rn}, and Th^A_{Wn}).

In this case, our proposed SAAT method is responsible for determining a set of appropriate assigned throughput values,

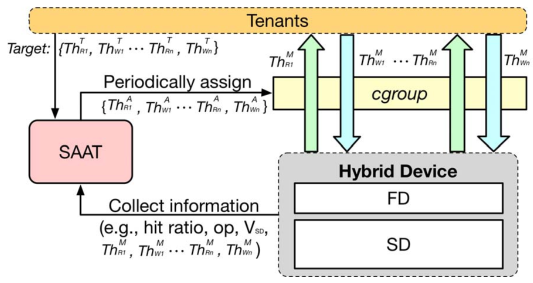

Tenants
Target: $\{Th^T_{R1}, Th^T_{W1} \cdots Th^T_{Rn}, Th^T_{Wn}\}$
Th^M_{R1}
$Th^M_{W1} \cdots Th^M_{Rn}$
Th^M_{Wn}
Periodically assign
$\{Th^A_{R1}, Th^A_{W1} \cdots Th^A_{Rn}, Th^A_{Wn}\}$
cgroup
SAAT
Hybrid Device
FD
SD
Collect information
(e.g., hit ratio, op, V_{SD}, $Th^M_{R1}, Th^M_{W1} \cdots Th^M_{Rn}, Th^M_{Wn}$)

Fig. 2. Accurate performance isolation of hybrid storage solution based on our proposed SAAT.

which are usually larger than the target throughput, for all the tenants to make the measured throughput close to the target as much as possible. By collecting the latest statistics and the difference between the actual throughput and the specified throughput, SAAT can give appropriate assigned throughput values to cgroup periodically. In the following parts, we will present how to determine the assigned throughput settings in the single-tenant case and the multiple-tenant case.

A. Single-Tenant Case

Single-tenant case means only one user is occupying the hybrid device. This is the basics of the multiple-tenant case. We assume the cache works in the write-back mode and there are adequate I/O requests, i.e., not less than the target and the assigned throughput, i.e., Th^T and Th^A. As Fig. 3 shows, all the write requests will be written into FD first, and those misses the cache will trigger a victim written back to SD. For the read requests, the difference lies in that the missed requests will not cause the working of FD. Therefore, the total bytes of coming requests req_f is equal to Th^A multiplied by FD's service time (i.e., t_f).

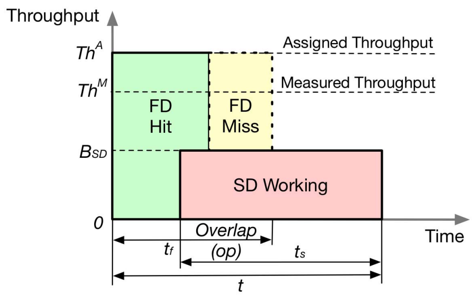


Fig. 3. Schematic diagram of requests processing in a hybrid device.

During the request processing period, FD and SD may work in parallel. Thus, we use t_s to represent the time that SD is working, and t_f for FD. Since FD and FD usually work in parallel, we use op to represent the proportion of the overlap time in t_f. The measured throughput Th^M is equal to the total bytes of coming requests req_f divides a given time period t, as shown in Eq. (1), for both read and write requests.

$$Th^M = \frac{Th^A \times t_f}{t_s + t_f - op \times t_f} \tag{1}$$

All cache misses during the period t trigger SD working. Because the backend operations are not throttled by cgroup and SD is often the bottleneck, SD usually processes I/O requests at its full bandwidth, i.e., B_{SD}. So we can get Eq. (2).

$$t_s \times B_{SD} = R_{miss} \times req_f \tag{2}$$

By eliminating t_f, t_s, and req_f in Eq. (1) $\sim$ Eq. (2), we get the relationship between Th^M, Th^A, R_{miss}, and B_{SD}, shown in Eq. (3).

$$\frac{1}{Th^A} = \frac{1}{1-op} \times (\frac{1}{Th^M} - \frac{R_{miss}}{B_{SD}}) \tag{3}$$

B. Multiple-Tenant Case

When multiple tenants share one device, for $tenant_i$, the relationship of the measured throughput Th_i^M, the assigned throttle Th_i^A, the FD service time t_{f_i}, the SD service time t_{s_i}, and the overlap rate *op* of the whole hybrid device also accords with Eq.(1). During the period, SD will process the missed requests of all tenants, so we can get Eq. (4) for the multi-tenant case based on Eq. (2).

$$t_s \times B_{SD} = \sum_1^n R_{miss_i} \times Th_i^A \times t_f \tag{4}$$

By solving the N-ary equations, we can calculate the assigned throughput of each tenant, i.e., Th_i^A, shown as Eq. (5).

$$\frac{1}{Th_i^A} = (1 - op)^{-1}\left(\frac{1}{Th_i^M} - \frac{\sum_1^n R_{miss_i} Th_i^M}{B_{SD}}\right) \tag{5}$$

Eq. (5) reflects the relationships between the measured throughput, the assigned throttle, the cache miss rate, and overlap rate of FD and SD. According this equation, when the assigned throttle Th_i^A goes up, the measured throughput increases as well, which is in line with the actual situation. If the total number of missed requests $\sum_1^n R_{miss_i} Th_i^M$ increases, Eq. (5) suggests increasing Th_i^A to let more requests in, which also makes sense. When *op* goes up, Th_i^A is supposed to be smaller, because we should lease pressure when the device is busy.

C. Algorithm Design

The process of the SAAT algorithm is summarized in Alg. 1, in which the initial assigned Th^A is equal to target Th^T. When the requests arrive, we start a daemon to monitor the throughput of each tenant and to collect information including the miss rate and the measured throughput of current period. Then we use function f_1 to calculate *op* and use function f_2 to predict the next assigned throughput Th_{next}^A. f_1 and f_2 can be derived from Eq. (5).

Algorithm 1 SAAT Algorithm

```
$Th^A \leftarrow Th^T$
while $working$ do
    $(R_{miss}, Th_{cur}^A) \leftarrow CollectStatistics(Period)$
    $op \leftarrow f_1(Th^M, Th_{cur}^A, R_{miss}, B_{SD})$
    $Th_{next}^A \leftarrow f_2(op, Th^T, R_{miss}, B_{SD})$
end while
```

IV. Evaluation

In this section, we will evaluate the accuracy of the performance isolation on hybrid storage devices for SAAT and the original version of *cgroup*. The experiments were performed on a PC with 8-core Intel i7-6700 CPU @ 3.4GHz and 16GB DRAM, running 64-bit Linux kernel v4.4.62. We first built a hybrid device by Flashcache [2] consisting of a 100 MB persistent memory as the fast device (FD) and a 250GB solid-state drive as the slow device (SD). We modified the statistics module of Flashcache to collect the miss rate of each process. *cgroup* and SAAT were deployed to do the performance isolation on the hybrid storage device.

We adopted two kinds of metrics to evaluate the accuracy of performance isolation. The first metric is the Difference of Average Throughput, i.e, *DAT*, between the measured one Th^M and the target Th^T. The second one is *Average Jitter*, i.e., the average value of the difference between the measured throughput Th_P^M and the target throughput Th_P^M in each period P. The first metric reflects the overall accuracy in a long time period, while the second reflects how steady the measured throughput floating around the target.

The Microsoft Research (MSR) Cambridge traces [11] collected from typical enterprise data centers were replayed to represent different tenants in our experiments. As Table I shows, our evaluations include ten different kinds of applications.

TABLE I
THE REAL-WORD I/O WORKLOADS ADOPTED IN THE EVALUATIONS.

Trace	Application Type	Request Count	Read Ratio
usr	User home directories	12,873,274	72.14%
prn	Print server	17,635,766	19.79%
hm	Hardware monitoring	8,985,487	32.65%
rsrch	Research projects	3,254,278	11.22%
src	Source control	14,024,860	16.78%
stg	Web staging	6,098,667	31.76%
ts	Terminal server	4,216,457	25.89%
web	Web/SQL server	9,642,398	53.59%
mds	Media server	2,916,662	29.61%
wdev	Test web server	2,654,824	27.3%

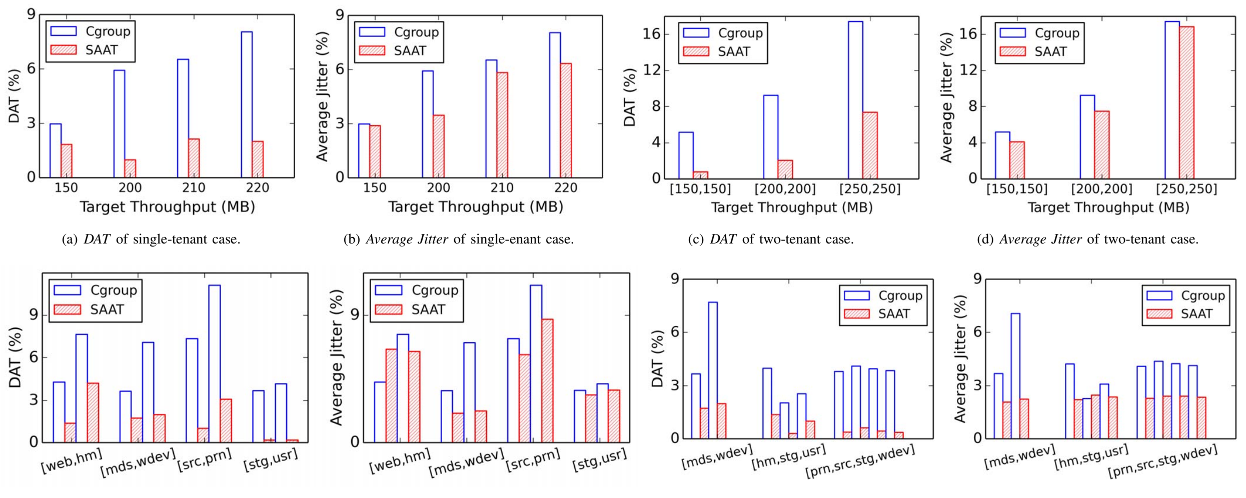
A. Overall Results

In order to compare *DAT* and *Average Jitter* under different target throughput settings, we use the ratio of them to the value of target throughput. For example, if we set the target throughput to 200 MB/s and the measured difference is 10 MB/s, the *DAT* or *Average Jitter* is considered as 5%.

We performed 14 groups of experiments covering the cases of varied trace combinations, different target throughput values, and different number of tenants. Each group was run for five times and the average results were adopted in our experiments. On average of the 14 groups, SAAT can reduce *DAT* from 6.56% to 1.94% (i.e., 3.38 times), and lower *Average Jitter* from 6.62% to 5.25%.

In the single-tenant case, we adopted the target throughput settings ranging from 150 MB/s to 220 MB/s on five traces and calculated the average results of the five traces. Fig. 4 (a) shows that the SAAT algorithm can reduce *DAT* from 5.87% to 1.7%, achieving 3.39 times smaller throughput shift. Furthermore, *DAT* of the original cgroup case keeps increasing when the target throughput increases (i.e., from 2.96% to 8.05%). On the contrary, SAAT can maintain a stable and smaller *DAT* in these cases (i.e., 0.98% $\sim$ 2.13%). For *Average Jitter*, SAAT is 1.26 times better than the original cgroup method (i.e., 5.87% vs. 4.63%), as shown in Fig. 4 (b). Note that the reduction of *Average Jitter* is lower than *DAT* because the former requires accurate throughput throttling in each period, much harder than the latter.

In the multiple-tenant case, we set two tenants to the same target throughput values ranging from 150 MB/s to 250



(a) *DAT* of single-tenant case. (b) *Average Jitter* of single-enant case. (c) *DAT* of two-tenant case. (d) *Average Jitter* of two-tenant case.

(e) *DAT* for different trace combination. (f) *Average Jitter* for different trace combination. (g) *DAT* for 2, 3, or 4 tenants. (h) *Average Jitter* for 2, 3, or 4 tenants.

Fig. 4. **Comparisons between SAAT and the original cgroup**: SAAT outperforms cgroup on both metrics, i.e. *DAT* and *Average Jitter*.

MB/s. We calculated the average *DAT* or *Average Jitter* of two tenants for each test. The results indicate that SAAT constantly outperforms cgroup when facing multiple tetants in all throughput settings. As Fig. 4 (c) shows, SAAT reduces *DAT* from 10.61 % to 3.85 % and reduces *Average Jitter* from 10.62 % to 9.48 % on average.

B. *Impacts of Traces*

In order to evaluate SAAT under various workloads, we performed four groups of experiments using four different combinations and eight traces when two tenants were deployed. We set the target throughput as 200 MB/s in all cases. As Fig. 4 (e) plots, SAAT outperforms cgroup in each group of trace combination on *DAT*. For *Average Jitter*, Fig. 4 (f) exhibits that SAAT is 1.24 times better than cgroup on average.

C. *Scalability Evaluation*

We measured the performance of SAAT when the tenant number is 2, 3, or 4, respectively. The results show that SAAT always achieves much less throughput shift as the count of tenant increases. SAAT improves *DAT* of each tenant, shown as Fig. 4 (g), for 3.78 times compared with cgroup on average. When four tenants share the same hybrid device, SAAT can reduce *DAT* from 3.91% to 0.452%, which is 8 times of improvement. Fig. 4 (h) indicates that *Average Jitter* is reduced from 4.245% to 2.293% on average.

V. Conclusion

In this paper, we give a deep analysis about the accuracy of the performance isolation of hybrid storage devices. Aiming at solving the throughput shift problem of the current mainstream performance isolation solution (i.e., Linux *cgroup*), we propose a new Self-Adaptive Accurate Throttling (SAAT) method to dynamically calculate and set appropriate values of the assigned throughput for cgroup I/O throttling based on the performance model of hybrid storage devices. Experiments driven by practical hybrid devices and enterprise I/O workloads exhibit that SAAT can effectively bridge the gap between the expected throughput of tenants and the measured one.

Acknowledgment

This work is supported by the National Key Research and Development Program of China (No. 2018YFB1004401), National Natural Science Foundation of China (No. 61732014 and 61472427), Beijing Natural Science Foundation (No. 4172031), and the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China (No. 16XNLQ02).

Yunpeng Chai is the corresponding author of this paper.

References

[1] Tejun Heo. Control Group v2. https://www.kernel.org/doc/Documentation /cgroup-v2.txt. 2015.
[2] Facebook. Flashcache: A general purpose, write-back block cache for Linux. https://github.com/facebookarchive/flashcache. 2017.
[3] Saxena M, Swift M M, Zhang Y. FlashTier: a lightweight, consistent and durable storage cache[C]//Proceedings of the 7th ACM european conference on Computer Systems. ACM, 2012: 267-280.
[4] Yang Q, Ren J. I-CASH: Intelligently coupled array of SSD and HDD[C]//High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on. IEEE, 2011: 278-289.
[5] Kgil T, Roberts D, Mudge T. Improving NAND flash based disk caches[C]//Computer Architecture, 2008. ISCA'08. 35th International Symposium on. IEEE, 2008: 327-338.
[6] Boboila S, Desnoyers P. Write Endurance in Flash Drives: Measurements and Analysis[C]//FAST. 2010: 115-128.
[7] Gregg, B., L2arc, https://blogs.oracle.com/brendan/entry/test, 2008.
[8] Pritchett T, Thottethodi M. SieveStore: a highly-selective, ensemble-level disk cache for cost-performance[C]//ACM SIGARCH Computer Architecture News. ACM, 2010, 38(3): 163-174.
[9] Huang S, Wei Q, Feng D, et al. Improving flash-based disk cache with lazy adaptive replacement[J]. ACM Transactions on Storage (TOS), 2016, 12(2): 8.
[10] Chai Y, Du Z, Qin X, et al. Wec: Improving durability of ssd cache drives by caching write-efficient data[J]. IEEE Transactions on computers, 2015, 64(11): 3304-3316.
[11] Narayanan D, Donnelly A, Rowstron A. Write off-loading: Practical power management for enterprise storage[J]. ACM Transactions on Storage (TOS), 2008, 4(3): 10.