Analysis of and Optimization for Write-dominated Hybrid Storage Nodes in Cloud

Shuyang Liu[1], Shucheng Wang[1], Qiang Cao[1], Ziyi Lu[1],
Hong Jiang[2], Jie Yao[1], Yuanyuan Dong[3] and Puyuan Yang[3]

[1]Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System,
Engineering Research Center of data storage systems and Technology, Ministry of Education of China,
Huazhong University of Science and Technology
[2]University of Texas at Arlington
[3]Alibaba
CorrespondingAuthor:caoqiang@hust.edu.cn

ABSTRACT

Cloud providers like the Alibaba cloud routinely and widely employ hybrid storage nodes composed of solid-state drives (SSDs) and hard disk drives (HDDs), reaping their respective benefits: performance from SSD and capacity from HDD. These hybrid storage nodes generally write incoming data to its SSDs and then flush them to their HDD counterparts, referred to as the *SSD Write Back* (**SWB**) mode, thereby ensuring low write latency. When comprehensively analyzing real production workloads from Pangu, a large-scale storage platform underlying the Alibaba cloud, we find that (1) there exist many write dominated storage nodes (WSNs); however, (2) under the SWB mode, the SSDs of these WSNs suffer from severely high write intensity and long tail latency. To address these unique observed problems of WSNs, we present *SSD Write Redirect* (**SWR**), a runtime IO scheduling mechanism for WSNs. **SWR** judiciously and selectively forwards some or all SSD-writes to HDDs, adapting to runtime conditions. By effectively offloading the right amount of write IOs from overburdened SSDs to underutilized HDDs in WSNs, SWR is able to adequately alleviate the aforementioned problems suffered by WSNs. This significantly improves overall system performance and SSD endurance. Our trace-driven evaluation of SWR, through replaying production workload traces collected from the Alibaba cloud in our cloud testbed, shows that SWR decreases the average and 99th-percentile latencies of SSD-writes by up to 13% and 47% respectively, notably improving system performance. Meanwhile the amount of data written to SSDs is reduced by up to 70%, significantly improving SSD lifetime.

CCS CONCEPTS

• **Information systems** → *Record and block layout*; *Query reformulation*; Physical data models; Point lookups;

KEYWORDS

Hybrid Storage, Write-dominated Workload, SSD Queue Management, SSD Write Redirect

ACM Reference Format:
Shuyang Liu[1], Shucheng Wang[1], Qiang Cao[1], Ziyi Lu[1],, Hong Jiang[2], Jie Yao[1], Yuanyuan Dong[3] and Puyuan Yang[3]. 2019. Analysis of and Optimization for Write-dominated Hybrid Storage Nodes in Cloud. In *ACM Symposium on Cloud Computing (SoCC '19), November 20–23, 2019, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3357223.3362705

1 INTRODUCTION

Large cloud providers have significant pressures to offer attractive features and services to customers, while controlling their datacenter costs. Most providers, such as Google[9], Amazon[18], Facebook[16] and Microsoft's online services[2], have built their respective global and warehouse-scale distributed storage systems to uniformly support a myriad of high-level online services. These cloud storage systems provide data abstractions, such as Files, Blobs, Tables and Queues, to enable the rapid development of new business lines. Pangu [6] is such a large-scale storage platform underlying Alibaba cloud with high-reliability, high-availability and high-performance, supporting most Alibaba cloud services including Table Store, MaxCompute, and AnalyticDB.

In order to achieve overall cost-effectiveness, cloud storage systems have wholeheartedly embraced storage heterogeneity within storage nodes[19, 29]. Solid-state drive (SSD) offers higher IOPS and lower IO latency[7] than hard disk drive (HDD) while the latter provides larger capacity at lower cost. Therefore, storage nodes generally are deployed with different types and numbers of SSDs and HDDs. To effectively reap the write performance advantage of SSDs, a hybrid storage node, first writes incoming data from frontend application servers into SSDs, and then flushes them into HDDs. This is referred to in this paper as the *SSD Write Back* (**SWB**) mode.

Observations from real production workloads of Pangu indicate that many storage nodes supporting latency-critical services experience a write-dominant IO pattern. We refer to these nodes as *write-dominant storage nodes* (WSNs) in this paper. Specifically, 77%-99% of IOs in these nodes are writes. This phenomenon stems from the fact that, to ensure high reliability and availability, frontend servers always flush new and updated data to the backend storage nodes as soon as possible, while reserving local fast-storage to cache hot data for user reads. In fact, backend storage nodes rarely serve reads from frontend servers because the upper latency-critical online services generally build their own application-aware cache to enhance service responsiveness.

By holistically analyzing traces of production workloads collected from Pangu WSNs, we come to the conclusion that executing the SWB mode in WSNs leads to severely harmful SSD overuse with high write intensity and long tail latency. These SSDs experience high write intensity (e.g., an average interarrival time of 62us) and large write volume (e.g., 3 TB per day per disk) while the IO capacity of the HDDs is severly under-utilized(e.g., 0.5 TB per day per disk). Worse still, SSDs with high IOPS suffer heavy-tail IO latency, due to the head-of-the-queue blocking by large writes (e.g., 1MB) and frequent garbage collections induced by high write intensity. As a result, optimizing writes is critically important, especially for WSNs where excessive writes harm both performance and reliability of SSDs.

In this paper, we propose *SSD write Redirect*, or **SWR** for brevity, a runtime IO scheduling mechanism for WSNs. The key idea behind SWR is to strategically offload writes intended for selected SSDs to HDDs while guaranteeing the IO latency requirement. In other words, SWR dynamically redirects some or all incoming SSD-writes to HDDs based on runtime conditions to exploit the underutilized IO handling capacity and sequential nature of HDDs within WSNs. So the IO burdens of SSDs can be significantly relieved.

The key runtime conditions for write redirection are write size and IO queue length. Specifically, SWR redirects SSD-writes of sizes exceeding a threshold to idle HDDs. The latencies of such large writes may suffer as a result of being executed in HDDs instead of their intended SSDs. However, many small SSD-writes behind large writes in the IO queues will no longer be blocked and can be completed quickly, thus decreasing the average and tail latencies. SWR also monitors the IO queue lengths of all SSDs and HDDs at runtime and redirects SSD-writes on heavily-loaded SSDs (i.e., with large queue length) to idle HDDs. This scheme helps reduce the write intensity and GC-induced delays by decreasing the GC frequency that is generally proportional to write intensity. The threshold values of write size and queue length which decide write redirection are initialized by the storage characteristics and workload behavior of each WSN, e.g., 99th-percentile size of SSD-writes as the threshold value. Once initialized, the threshold values can vary dynamically according to the current queue length of SSD, so that tail latency can be reduced while guaranteeing average latency.

This paper makes the following contributions:

- We analyze detailed IO characteristics of production workloads from Pangu and gain key insights into unique IO patterns in storage nodes of the cloud. We reveal that the excessive write IOs under the **SWB** mode can lead to severely harmful SSD overuse with high write intensity and long tail latency.
- We propose **SWR**, a runtime IO schedule mechanism to reduce excessive write and long tail problems of SSDs in WSNs by redirecting SSD-writes to HDDs while guaranteeing IO latency requirement.
- We design and implement a prototype SWR system and evaluate the effectiveness of the SWR scheduling mechanism. Evaluation driven by production workload traces collected from Pangu show that SWR reduces the total amount of data written to SSDs by up to 70%, alleviating wear-out of SSDs. Meanwhile, SWR effectively reduces both the average and tail latencies of SSD-writes by up to 13% and 47%, respectively.

The rest of our paper is organized as follows. Section 2 provides the necessary background for the SWR research, namely, the Pangu cloud storage system and WSN. Section 3 presents the Pangu production trace analysis to motivate the proposed SWR. Section 4 describes our methodology for the design and implementation of the SWR mechanism. We evaluate the effectiveness of SWR in reducing wear-out and improving write performance of SSDs in Section 5. Finally, Section 6 reviews prior works most relevant to SWR, while Section 7 concludes the paper with remarks on future work.

2 BACKGROUND

2.1 Pangu

Pangu is a hyper-scale and distributed storage system [2][18] for Alibaba Cloud. A storage cluster in Pangu deploys more than 10,000 servers, with a total capacity exceeding 1 EB. It provides cost-effective and unified storage services not only for Alibaba Cloud but also for other independent businesses of Alibaba Group and Ant Financial[6], such as object storage, cloud computing, etc. As a unified storage platform of Alibaba Group, Pangu must minimize the cost of total ownership and support multiple computing clusters while meeting QoS requirements.

Pangu is a simple and scalable three-party architecture, as shown in Figure 1. It consists of Application Servers (AS), MetaServers (MS) and ChunkServers (CS). Application Servers are storage clients and provide online services for users. A subset of ChunkServers and Application Servers together are generally partitioned into Business Zones, where each Business Zone consists of a group of ApplicationServers and ChunkServers. This provides a specific online service (e.g., the Object Storage Service), avoiding multi-business collocation and potential hardware resource contention. ASs can access data in ChunkServers directly. Each file in Pangu is divided into a series of Chunks distributed across multiple

ChunkServers. Chunk is the smallest data management unit with itsa size of up to 64MB. Each Chunk is allocated a globally unique chunk ID. Each MetaServer has a global location table that maintains the mapping relationship of files to chunks and chunks to ChunkServers. MetaServers adopt the ParallelRaft protocol[3] to achieve a fully distributed metadata management with high load-balance, reliability and availability. To facilitate access to applications in the open source ecosystem, Pangu is compatible with HDFS. So it enables seamless accesses to massive data and interactions with other storage products of Alibaba Cloud.

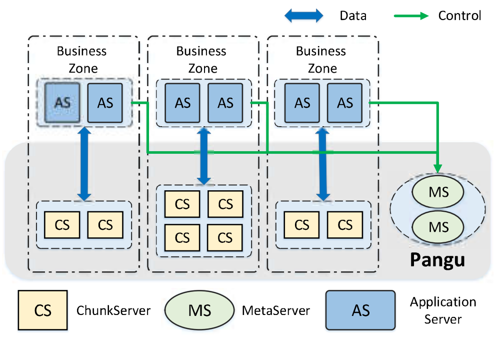


Figure 1: The system architecture of the Pangu cloud storage. The core base layer of Pangu consists of the Application Servers (clients), MetaServers and ChunkServers.

2.2 Hybrid Storage

Large-scale cloud storage systems like Pangu demand extremely high end-to-end performance at low cost. To meet this demand, the clouds increasingly embrace storage heterogeneity. They deploy variable types and numbers of SSDs, which offer higher IOPS and lower IO latency[7], and HDDs, which provide larger capacity at low cost, in each storage node.

Table 1: The characteristics of disk types.

Disk Type	SSD			HDD
Interface	PCIe NVMe	PCIe AHCI	SATA AHCI	SATA AHCI
Cost($/GB)	1.2-2.6	0.6-1.1	0.5-1.0	0.2-0.45
Ave. write lat.(us)	20-100	30-200	30-200	10k-30k
Ave. read lat.(us)	20-100	30-200	30-200	10k-30k
Max. throughput (GB/s)	3	0.52	0.52	0.2

The performance characteristics of mainstream SSDs and HDDs are listed in Table 1. The peak throughput of commodity SSDs reaches 3GB/s for the PCIe type and 520MB/s for the SATA type. The write IO latency of 4KB is as low as 100us-200us. Generally, the high performance of SSDs also means high cost. On the contrary, HDDs generally have a high capacity/cost ratio. Further, the actual performance of HDDs heavily depends on IO request size and access pattern. The write throughput of HDD varies with requested block sizes. We used *fio* to perform continuous and sequential writes to the disk (a 4TB ST4000DM005 HDD) with different write block size. When the block size is larger than 16KB, the disk reaches its maximum throughput at 180MB/s. Moreover,

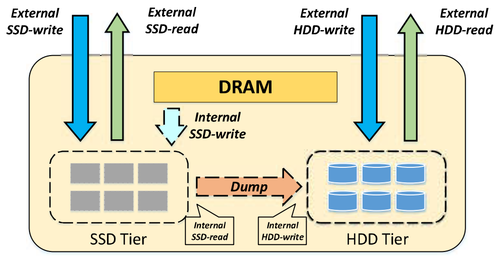


Figure 2: The architecture and IO flows of hybrid WSN comprising an SSD and an HDD Tier.

HDDs are notorious for their poor random IO performances due to mechanical delays by seeking and rotating because of a seeking latency of 9.5ms in average [27]. To reap the write performance of SSDs, hybrid storage nodes first write incoming data from frontend application servers into SSDs, then integrate and flush them into HDDs. This write pattern is referred to as the *SSD Write Back* (**SWB**) mode.

Compared to HDDs, SSDs have limited Program/Erase (P/E) cycles and asymmetric read/write delays [30]. To avoid fast and unpredictable wear-out, SSDs are limited in the amount of external data written daily defined as DWPD (Drive Writes Per Day) in practice. TBW (Terabyte Written) is used to approximately measure SSD's lifespan. Wear leveling mechanisms ensure that writes are evenly distributed over SSD cells. However, the total amount of data written in an SSD is determined by the product of its cell count and the maximal P/E cycles. A recent study [14] indicates that the limited write endurance is a critical design issue in SSD-based storage systems.

Additionally, out-of-place-write nature of SSDs necessitates garbage collection (GC), an SSD performance killer. It reclaims the invalid pages by moving valid pages to new blocks and then erases the old blocks[17]. GC can block incoming reads/writes, resulting in performance instability and long tail latencies. Therefore, recent studies [17, 31] indicate that SSDs do not always deliver their raw performance, often far from it.

2.3 Pangu ChunkServer

To take better tradeoff between performance and cost for different applications, Pangu generally deploys ChunkServers with different types and numbers of SSDs and HDDs.

Figure 2 shows typical IO flows in the hybrid ChunkServers in Pangu. IOs are categorized into **internal** and **external** for both read and write. The external writes and reads come from the frontend application servers through the datacenter network. They consist of two types: external SSD-writes/reads and external HDD-writes/reads. Both types of external writes are explicitly launched by Application Servers. ChunkServers perform the external SSD-writes to SSDs and then dump data into HDDs in the background. The external HDD-writes are directly written into HDDs without passing SSDs. ChunkServers periodically flush their internal critical data (e.g., local system metadata) from DRAM into SSDs in a log manner to ensure crash-recovery. The logging writes actually

Table 2: Trace characteristics. Traces are recorded from three different online services in Pangu: *A*, *B*, *C*. We select two *A* WSNs, two *C* WSNs, and one *B* WSN. The SSD ratio is the number of SSDs to all disks deployed in a WSN. The ratio is further divided into three levels as High (>33%), Mid (>10% and ⩽33%), and Low (⩽10%).

Node type	SSD ratio(%)	Time duration (hour)	Number of requests (millions)	Write request ratio(%)	Read request ratio(%)	Write data (GB)	Read data (GB)	Avg. IO intensity (us)	Avg. request size(KB)	Max. request size(KB)
A1	High	22	36.7	77.2	19.2	1534	24	2163	56	22336
A2	Mid	15	28.5	77.5	5	794	408	1933	73	21664
B	Low	10	66.9	97.2	0.9	9875	295	543	177	3092
C1	High	0.5	66	99.3	0.44	185	53	62.3	4.2	3841
C2	Mid	0.5	65.8	99.2	0.43	184	46	63.9	4.1	3827

invoke internal SSD-writes. The internal HDD-writes and internal SSD-reads are caused by the dump operations that migrate data from SSDs to HDDs.

2.4 Write-dominated Storage Nodes

Through production trace analysis detailed in the next section, many ChunkServers in Pangu experience a write-dominant workload behavior. We call these CSs WSNs (Write-dominant Storage Nodes). 77%-99% of requests are writes in WSNs, and the volume of data written in them is 2-3 orders of magnitude larger than that of data read from them.

This phenomenon can be explained by the fact that the frontend application servers absorb the vast majority of reads by caching hot and time-critical data. Typical frontend latency-sensitive online applications such as web applications, search engines or message services, generally have their application-aware cache layer [16], which deploys fast storage based on all SSD or large memory to serve user reads. Some hot online-service applications employ multiple cache layers, including Content Delivery Network (CDN)[10, 16, 25] to effectively serve most and burst reads. Consequently, only a tiny fraction of all read requests actually missed by the frontend application servers need to be responded by the backend ChunkServers in Pangu. Meanwhile, to ensure large available caches, such applications also attempt to immediately flush all writes into Pangu and reserve adequate local fast storage for hot data[3]. Therefore, Pangu offers a unified data persistence platform for production data, replica data, or the intermediate results in cloud computing. As a result, many ChunkServers in Pangu must serve extremely frequent and massive writes.

Note that in current networked storage systems like Pangu, NVMe SSDs offer a bandwidth of 3.6 GB/s for sequential writes, which nearly matches the bandwidth a 40GbE network offers. This means that network latency is similar to IO latency, and network may become a performance bottleneck in cloud storage systems [5]. Therefore, ultra-low (us level) IO latency in Pangu is not strongly desired.

3 TRACE ANALYSIS

3.1 Workload Traces

Studies on Cloud in general and its storage systems in particular to date lack real-world workload traces of production cloud storage servers from large cloud service providers. To gain meaningful insight into the IO characteristics of storage nodes in Cloud, we trace real workloads from online Pangu Chunkservers. There are three representative Business Zones: *A* (Cloud Computing), *B* (Cloud Storage) and *C* (Structured Storage). For each online service, we trace all operations from two nodes for *A*, two nodes for *C*, and one node for *B* as representative.

These workload traces collectively amount to 264 million request records with more than 13TB request data. Table 2 shows the high-level characteristics of these collected traces. We observe that these storage nodes experience significant dominated writes as writes account for more than 77%, 97%, and 99% of all requests in *A*, *B* and *C*. Obviously, all of these Chunkservers are WSNs.

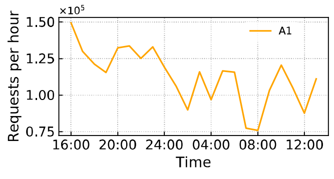
These WSNs have different configurations in number and interface type of their component SSDs and HDDs. Note that inactive HDDs and SSDs in WSNs are not tracked in the traces. The configuration difference is mainly a result of application requirements and gradual deployment plans. *B* node deploys the most HDDs for large capacity at low cost so that its SSD Ratio is the lowest in all nodes. The SSD Ratio of *A*1 node is higher than that of *A*2 node to ensure higher storage performance.

```
TimeStamp:          "2019-01-24 11:20:36.158678"
Operation:          "SSDAppend"
ChunkId:            81591493722114_3405_1
SATADiskId:         -1
SSDDiskId:          1
PhysicalLength:     16384
Offset:             56852480
Queueing delay:     76
IO delay:           213
......
```

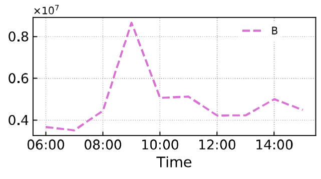
Figure 3: An example trace record.

Figure 3 shows the key part of a typical trace record with 9 important fields: the Timestamp field contains the request arrival time in the UTC timezone; Operation is the request command (e.g., SSDAppend and WriteChunk); ChunkID is the globally unique ID of each Chunk; DiskID shows the requested destination disk; Offset indicates the offset of the requested data in the Chunk; Length shows the amount of data written or read; QueueSize is the current queue length when requests complete; Waiting delay is the time of request waiting in the queue; IO delay indicates the processing time of requests in disks; The waiting delay and IO delay together constitute the *write latency* for a request. Some other auxiliary fields in a record are irrelevant in this work and are thus ignored.

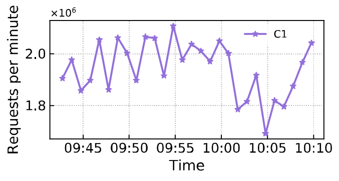
Pangu achieves good load balance across Chunkservers like Amazon S3[18] and Microsoft Azure[2]. This ensures that



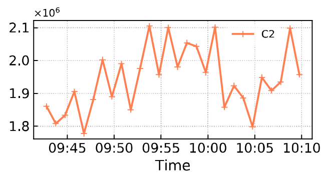
(a) A1



(b) B

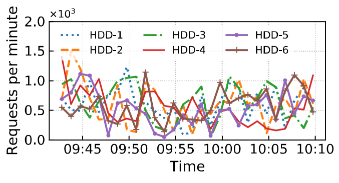


(c) C1

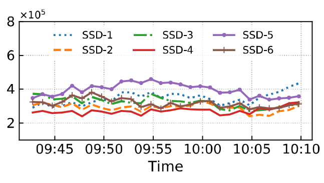


(d) C2

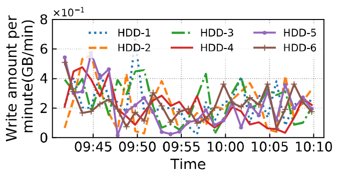
Figure 4: The request intensities, defined to be the number of requests per hour/minute, on the Chunkservers as a function of time under *A*, *B* and *C*.



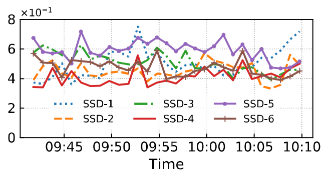
(a) External HDD-write(requests)



(b) External SSD-write(requests)



(c) External HDD-write(data)



(d) External SSD-write(data)

Figure 5: The number of requests and the amount of data written per minute in SSD and HDD of *C*1.

all Chunkservers in a Business Zone have similar workloads. Figure 4 shows that the request intensities on Chunkservers varies over time under A, B and C. These three scenarios exhibit significant workload variations. C workloads have a significantly high write intensity. To minimize intrusiveness and negative performance impact to Pangu's production services, we had to capture relatively short-range traces for the workload. These traces are still representative of the longer-time workload behaviors. However, even at the troughs, the request intensities are still quite high. Figure 5b and 5a demonstrate how the request intensities on six SSDs and six HDDs in $C1$ change over a 30-minute period of time. Figure 5d and 5c illustrate how the written data intensities vary on six SSDs and six HDDs of $C1$ over a 30-minute time window. The results show that both the request and written data intensities are roughly equal across either SSDs or HDDs

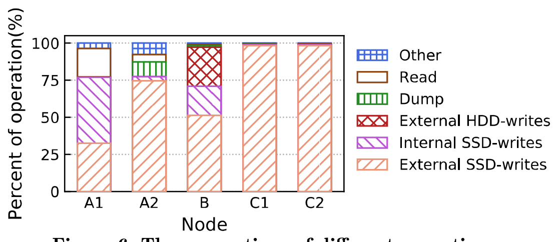


Figure 6: The proportions of different operations.

Table 3: The amount of data written to/read from one SSD or HDD in different types of WSN by proportionally scaling to 24 hours.

	A1	*A2*	*B*	*C1*	*C2*
SSD-writes(GB)	138	575	3071	820	1027
HDD-writes(GB)	0.1	3.7	555	384	355
SSD-reads(GB)	3.1	1.3	0.2	201	2.4
HDD-reads(GB)	0.3	12.3	0.2	33.6	7.2

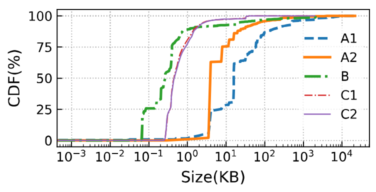
during this period. It means that WSNs achieve internal load balance reasonably well. Therefore, the workload of one SSD and one HDD can well represent their SSD-tier and HDD-tier respectively in the WSN.

3.2 SSD Dominant-Write Behavior

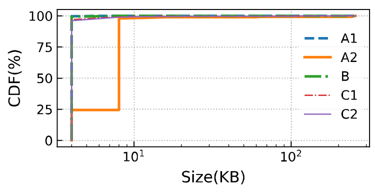
WSNs under **SWB** mode ensure quick data persistency for latency-critical online services. External-write data from frontend application servers is first written into SSDs, and then dumped to HDDs. Therefore, actual IO patterns in WSNs are heavily correlated to their corresponding applications based on Pangu. To better understand the write behaviors, we thoroughly investigate write operations, data written amount, request size and request interarrival time.

Figure 6 shows the proportions of all types of operations. First, the prevailing workload behavior in these five nodes is *dominant-write*. 97-99% of requests are writes in $C1$, $C2$ and B. The A set of nodes exhibit relatively lower write percentage (i.e., 77%). We believe that the difference mainly comes from their corresponding application requirements. We also observe that more than 99% of the writes are external SSD-writes in $C1$ and $C2$. *A1* exhibits 42% external SSD-writes but 57% internal SSD-writes. B demonstrates 53% external SSD-writes, 20% internal SSD-writes, but 27% external HDD-writes. The second frequent write operation is the internal SSD-writes. B prefers to use HDDs for cost-effective object storage. Therefore, the percentage of different type writes also depends on storage nodes. For these five WSNs, only A set of nodes have more than 5% reads, of which most are internal reads for the replica checksum. The external reads are as rare as less than 2%.

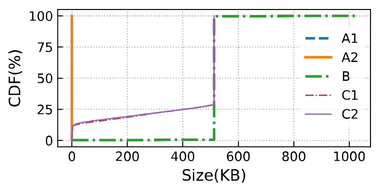
The most important factor impacting the SSD lifespan is the amount of data written [4]. We calculate the total amount of data, including both internal and external writes, that is written into one SSD of each type WSN by proportionally scaling to 24 hours. The results are tabulated in Table 3. The B node writes up to 1.25 terabytes to one SSD within 10 hours. On the contrary, the amount of reads generally is less than 4GB in all WSNs except $C1$.



(a) External SSD-writes.

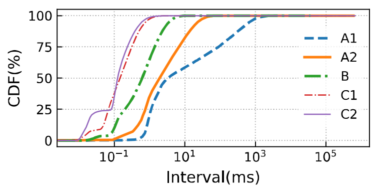


(b) Internal SSD-writes.

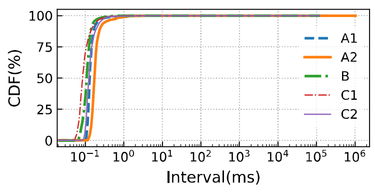


(c) External HDD-writes.

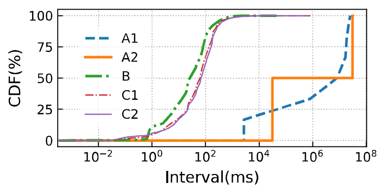
Figure 7: The request size CDF for different type writes.



(a) External SSD-writes.



(b) Internal SSD-writes.



(c) External HDD-writes.

Figure 8: The request interarrival times CDF for different types of writes.

There are two main observations. First, the key insight from the analysis is the prevailing problem of SSD overuse in WSNs, which is detrimental to the lifespan and performance of SSD. For example, assuming that an SSD in the *B* node has a capacity of 500GB with an average lifespan of 300TB[8] TBW (Terabyte Written), it will wear out within 4 months under the write intensities of production workloads underlying analysis. If the amount of data written to an SSD each day is limited by DWPD, more SSDs or storage nodes have to be added to meet the write volume requirement. This will increase the overall cost proportionally. Second, HDDs are underutilized in most of time, and we will analyze it in Section 3.4.

Next, we investigate the distributions of request size and interarrival time. Figure 7 shows CDFs of request size under three types of write operations. External SSD-writes have a wide range distribution of request sizes in three scenarios. *B* has a relatively small write sizes where 90% are less than 1KB, while *A*1 has large sizes with 75% being larger than 10KB. Internal SSD-writes have almost fixed data sizes, of which 74% are 8KB in *A*2 and 99% are 4KB in other WSNs. Similarly, the external HDD-writes have discrete and fixed sizes, e.g., 512KB in almost all WSNs except *A*. Such fixed write sizes could be caused by the threshold-based write-trigger scheduling polices used by both the application servers and Chunkservers.

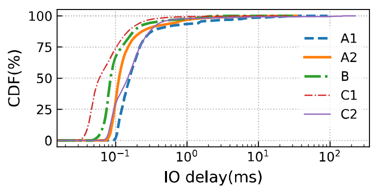
Figure 8 shows CDFs of request interarrival times under three write types. We find that internal SSD-writes are very intensive, with 90% of the interarrival times being less than 0.25 ms in all WSNs. Moreover, external writes have a wide range distribution of interarrival times for different node types. External SSD-writes in node *C* have the highest intensity. 99% of the interarrival times are less than 1 ms, while that in node *A* exhibit relatively low write intensities that 90% of them are more than 15 ms. On the contrary, most writes to HDD have second-level interarrival times, meaning that HDDs are very lightly loaded and remain idle most of the time.

3.3 Write Performance

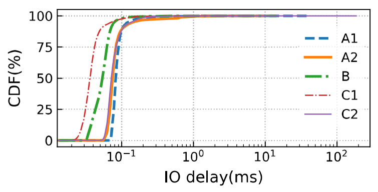
We now comprehensively analyze the write performance of WSNs. We first analyze write latency. Figure 9 shows the CDF of waiting delays and IO delays for different write types under the five WSNs.

Figure 9a, Figure 9b and Figure 9c show the IO delays for external SSD-writes, HDD-writes, and internal SSD-writes. We observe that 90% of the external SSD-writes are less than 800us while 90% of internal SSD-writes are less than 100us. This is because the former have generally larger IO size than the latter. Figure 7a and Figure 7b show that over 75% of external SSD-writes are of sizes one order of magnitude larger than their internal SSD-writes in *A* set of nodes, which directly results in the long IO delay of external SSD-writes. Meanwhile, *B* and *C* have similar IO delay distribution.

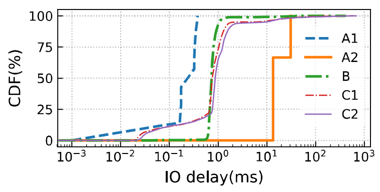
Figure 9d, Figure 9e, and Figure 9f show the waiting delays for external SSD-writes, internal SSD-writes, and external HDD-writes. The waiting delays of internal SSD-writes are far shorter than their own external writes. For internal SSD-writes in all nodes, 95% of waiting delays for the former are less than 3us, with the maximum being lower than 2.5ms. Whereas, 90% of waiting delays for the latter are almost 70us, far longer than that of internal SSD-writes. Because when two IO flows with different intensities execute concurrently, the length of the queues can more adversely affect the flow with a lower IO intensity (i.e. external SSD-writes) than a higher IO intensity (i.e. internal SSD-writes) [26]. Therefore, the external SSD-writes are severely affected by the write



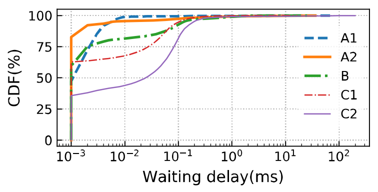
(a) CDF of IO delays of external SSD-writes



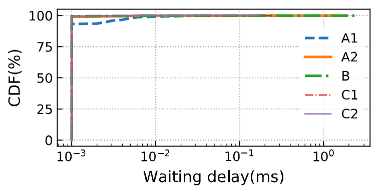
(b) CDF of IO delays of internal SSD-writes



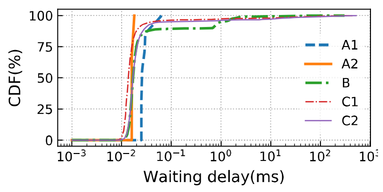
(c) CDF of IO delays of external HDD-writes



(d) CDF of waiting delays of external SSD-writes



(e) CDF of waiting delays of internal SSD-writes



(f) CDF of waiting delays of external HDD-writes

Figure 9: The distributions of IO and waiting delays for different types of writes.

Table 4: The average and maximum write latencies(in milliseconds) for three types of operations in WSNs.

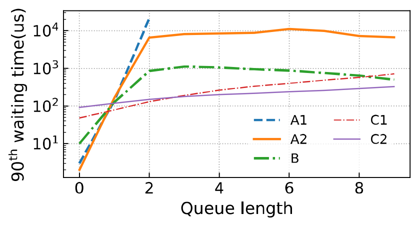
Node type	External SSD-write	Internal SSD-write	External HDD-write
***A*1**	0.8/113.0	0.09/39.5	0.3/0.4
***A*2**	0.3/94.1	0.1/9.2	19.2/31.0
B	0.1/31.8	0.05/14.4	1.3/415.7
***C*1**	0.1/25.3	0.04/6.7	4.3/613.3
***C*2**	1.0/302.0	0.09/184.1	6.9/774.8

intensities. Besides, the waiting delays are one order of magnitude larger than their IO delays, suggesting that are severely overloaded.

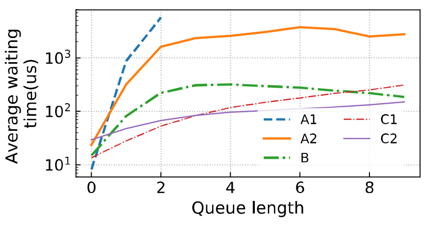
As a supplement to Figure 9, Table 4 provides the maximum and average write latencies of three write operations. It is found that the peak write latency is 2-3 orders of magnitude longer than the average ones in all five WSNs. For example, the gap between maximum and average latencies for external SSD-writes are 318x and 313x in B and $A2$ respectively. This is compelling evidence that writes to SSD suffer from a severe long-tail latency problem.

In what follows, we endeavor to find the root cause of the observed long-tail write latencies suffered by SSDs in WSNs. Figure 10a and 10b plot the 90^{th}-percentile and average latencies as a function of the queue length. The 90^{th}-percentile of request latency is lower than 100us without queuing. However, it becomes 2-3 orders of magnitude longer when the queue length reaches 2. Consequently, an outstanding request could cause long waiting delay for its subsequent requests.

To further understand the reason behind queue blockage, Figure 11a shows the request-size distribution of the queue-head requests which cause queue blockage. We observe that the distributions have a wide range, both large and small writes can cause blockage. First, for $A1$ and $A2$, at least 20% of the blocked writes are more than 512KB. This percentage is about two times larger than their original percentage in all writes as less than 10% shown in Figure 7a. Second, small



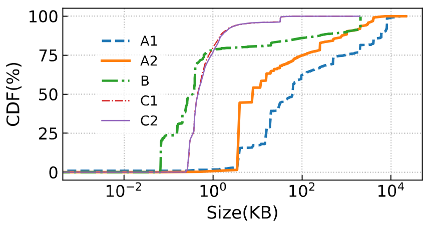
(a) The 90^{th}-percentile waiting delay.



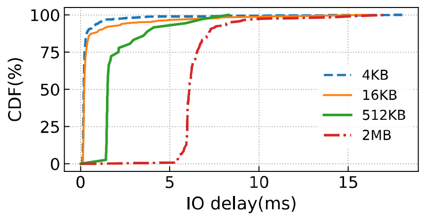
(b) The average waiting delay.

Figure 10: The 90^{th}-percentile and average waiting delays of all types of SSD-writes as a function of queue length from 0 to 10.

requests are also likely culprits for SSD queue blockage. 80% of the blocked requests in B, $C1$ and $C2$ and 45% in A are smaller than 4KB. Figure 11b further shows that 4KB, 16KB, 512KB, and 2MB blocked SSD-writes have long tails of IO delay. Garbage collections(GC) inside SSD can temporarily freeze the SSD queue until their completion, leading to the long-tail for all kinds of writes[17, 31]. A single GC blocks the SSD controller, as a result of which all outstanding IOs cannot be served[31]. The higher the write intensities (i.e., larger number of small requests in the queue) are, the more frequently garbage collections will be triggered.



(a) **CDF of size of queue-head requests causing queue blockage.**



(b) **CDF of IO delay of queue-head requests causing queue blockage.**

Figure 11: The size and IO delay distribution of the queue-head requests that cause queue blockage.

Consequently, a long IO-delay request (i.e., due to large IO or GC) can lead to queue blocking, which further lengthens waiting delays for its subsequent requests.

3.4 Low Utilization of HDD

Table 3 shows that there is a great gap between the amount of data written to SSD and that to HDD in all WSNs. For instance, in $A1$, the amount of data written by external SSD-write is 1380x larger than HDD-write, while such gaps in $A2$, B, $C2$ and $C1$ are 155x, 5.5x, 2.9x and 2.1x, respectively. This also implies very low HDD utilization, defined to be the percentage of time an HDD is working within a given time period. Specifically, Figure 12 plots the average utilizations of HDDs in all five WSNs over a 10-hour period. We observe that HDDs are generally in very low utilization states. For example, the HDD utilization in $A1$ is far less than 0.1% on average, while none in other WSNs is more than 14.3%. The SWB mode causes SSDs to be heavily loaded (overused) and become bottlenecks, while the HDDs are grossly underutilized. This is because SWB prevents HDDs from directly serving requests due to their high IO latency.

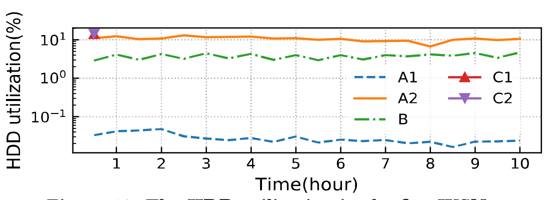


Figure 12: The HDD utilization in the five WSNs.

However, we found that the HDD performance is not as unacceptable as one might expect, raising the question of whether HDDs should directly serve some appropriate requests. Figure 9a and 9c reveal that the IO delay experienced by about 85% of HDD-writes is only 10 times longer than that experienced by SSD-writes in the $C1$, $C2$ and B nodes. The SSD and HDD performances in IO delay in *A1* are almost the same. Meanwhile, Figure 10 illustrates that the tail latency of write requests can be orders of magnitude longer than the average in a long-queue state. In other words, the actual write delay of SSDs with very heavy workload can be close to that of HDDs with very light load. This suggests that we should exploit the underutilized HDDs to serve some of the SSD-writes without significant performance degradation when SSDs are heavily loaded.

3.5 Dump

In the **SWB** mode, the dump operation is an indispensable internal operation that migrates data from SSDs to HDDs. When dump is executed, both SSD-reads and HDD-writes are performed, which will occupy the request queues of HDDs. To understand the impact of the dump operation on the service quality of WSNs, Figure 13 depicts the CDF of the disk queue length caused by dump. We observe an interesting fact that in all WSN nodes, more than 95% of the dump operations do not cause any queue blocking since the queue length is always less than 1 in these cases. This is because Pangu uses an efficient idle-time dump strategy, in which the dump operation is triggered only when there is few requests. In this way, even if the dump operation moves terabytes of data and consumes IO bandwidth for both SSDs and HDDs, it has negligible effect on user requests.

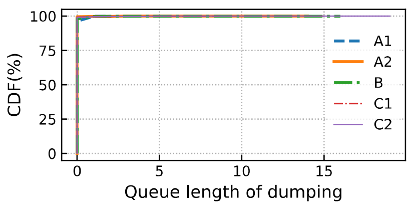


Figure 13: The queue length CDF of dumping.

3.6 Summary

We find that:(1) In WSNs, SSDs overuse is very prevalent, which is detrimental to both lifespan and performance of SSDs. Meanwhile, HDDs are underutilized in most of time. (2) Writes to SSD suffer from a severe long-tail latency problem, which is caused by large IO or GC. (3) The dump operation has negligible effect on user requests.

4 DESIGN OF SSD-REDIRECT

In this section, we propose *SSD Write Redirect*, or **SWR**, a runtime IO scheduling mechanism for WSNs. The goal of SWR is to relieve the SSD write pressure by leveraging HDDs while ensuring QoS. The architecture of SWR is shown in Figure 14. The SWR algorithm is described as Algorithm

1. SWR monitors all request queues of SSDs and HDDs in WSNs at runtime. When external and internal SSD-writes arrive, the scheduler determines whether an SSD-write should be redirected. If yes, the scheduler determines a suitable HDD as the destination for the redirected write. If all HDDs have waiting requests, the redirection is paused and the request is still written back to its original SSD. This strategy is based on the observation that SSD-writes suffer from dramatically long tail latency upon workload bursts or heavy GCs. The queue length can be 8-10 times longer than its average in this case.

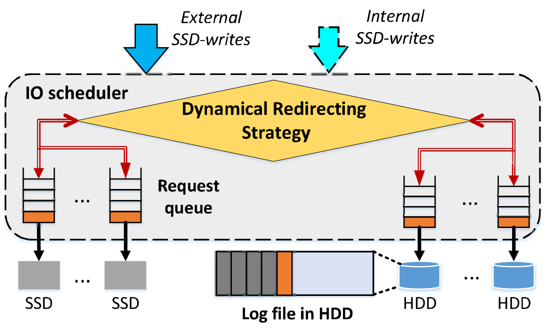


Figure 14: Architecture of SWR. It monitors all queues of SSDs and HDDs. The SSD-writes meeting the conditions are redirected to appropriate destination HDDs.

Algorithm 1 SWR algorithm

```
Input: Request $i$ operation: $op_i$, Request $i$ size: $size_i$;
    SSD queue length at time $t$: $l_{SSD}(t)$, HDD queue length
    at time $t$ : $l_{HDD}(t)$;
    Initial size value $S_{max}$, Step value $p$, SSD queue length
    threshold $L$;
Output: $flag_i$(True: to SSD queue, False: to HDD queue);
 1: Set the size threshold value of $S$ to $S_{max}$
 2: for the head request $i$ in the global write queue do
 3:   if $op_i$ is HDD-write then
 4:     $flag_i$ is False
 5:   else
 6:     if $l_{SSD}(t)$ exceeds $L$ then
 7:       calculate $S$ according to $S_{max}$ and $p$
 8:     end if
 9:     if $l_{HDD}(t)$ is 0 and $size_i$ is higher than $S$ then
10:       $flag_i$ is True
11:     else
12:       $flag_i$ is False
13:     end if
14:   end if
15:   Send the request $i$ to the corresponding queue accord-
      ing to $flag_i$
16: end for
```

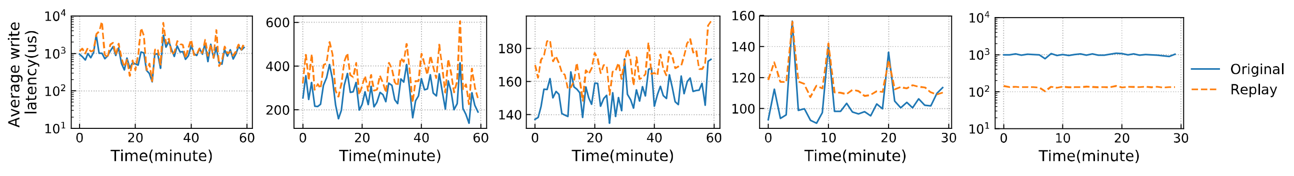
SWR simply redirects an SSD-write to an idle HDD when the request size s is larger than a size threshold S. Furthermore, SWR dynamically adjusts the S value according to current queue length $l_{SSD}(t)$ of SSDs. When the $l_{SSD}(t)$ of an SSD exceeds a threshold L, this SSD is considered overloaded and needs to reduce its write pressure by redirecting SSD-writes to idle HDDs.This will not sacrifice the average latency but slightly or moderately increase the latencies of a few of the redirected large-writes. Besides, the redirected writes are written to the destination HDD in a log manner to minimize write latency, because HDDs are friendly to large and sequential writes. In other words, both SSDs and HDDs work cooperatively to handle the write bursts.

From our earlier analysis, each WSN deploys different storage configuration and experiences diverse workloads. Therefore, SWR is a WSN-dependent and dynamical IO scheduler. The initial size-threshold is set to a maximum (upper limit) value S_{max}. S_{max} mainly affects the amount of redirected data and is set to a percentile (e.g. 99^{th}) of SSD-writes sizes in each WSN. The queue length threshold L is set to value such that when SSD queue length exceeds L, the write latency of blocked SSD-writes with S_{max} size is higher than the average IO delay of sequential HDD-writes. This value is determined from the measurements done in practice. It ensures that redirected SSD-writes do not lengthen their latencies significantly. L can practically reflect the impact of workload behaviors and hardware configurations. L can also be adjusted according to performance profiling of the real systems.

To better handle workload bursts, when the queue length is long, more incoming writes with a lower size-threshold S are preferred to be redirected to idle HDDs. To do this, when the $l_{SSD}(t)$ of an SSD is always higher than L, SWR gradually decreases the size threshold S with a fixed step value p. p deals with burst-intensive requests and tries to redirect more requests to HDDs. The step value p is set to be proportional to S_{max}, e.g., $1/4$. The threshold reduction ceases as soon as the $l_{SSD}(t)$ value returns back to L. Finally, in order to make full use of SSD, once the $l_{SSD}(t)$ becomes 0, it will be reset to the initial value.

Logging HDD-writes. To take full advantage of HDD sequential write performance, SWR executes redirected HDD-writes to a log file in an append-only approach. The log file is a device file that only stores the data field of HDD-writes in an append-only manner. We further use DIRECT_IO [21] to accelerate the data persistence process. Generally speaking, this append-only manner enhances write performance by sacrificing reads. Fortunately, for WSNs, external HDD-reads are very rare and HDDs are extremely lowly utilized. Besides, we also periodically write the logged Chunk data to their corresponding Chunks in HDDs, and then clear the log file.

Log Metadata.To manage all Chunks in the log file, we design a metadata structure that records and tracks the Chunks at request-level granularity. We choose a hash table as the key data structure because it is query friendly. To manage data at the granularity of Chunks, the hash table uses the *Chunk ID* field of a request as the hash key, and performs a single linked list to store the metadata of requests. Each list node represents an update request to this Chunk and records four fields: the length of the request data (*Data*



(a) A1 (b) A2 (c) B (d) C1 (e) C2

Figure 15: The average latencies of writes per minute in the original and replaying cases changes over time.

Length), the location of data that is persisted in the log file (*Physical offset*), the position of data in the Chunk (*Logical offset*) and the request arrival time (*Timestamp*) to keep the sequential order of requests.

5 EVALUATION

5.1 Experiment Setup

We run experiments on a server with two Intel Xeon E5-2696 v4 processors (2.20 GHz, 22 CPUs) and 64 GB of DDR3 DRAM. From trace analysis in Section 3, we observe that SSDs among five WSNs exhibit significant performance diversity. The peak throughput of SSD-writes in B, $C1$ and $C2$ reach 1GB/s, and 0.5GB/s in $A1$ and $A2$. To reflect such performance variations, we choose two types of experimental SSDs with different performance levels, i.e., a 256GB Intel 600p NVMe-SSD with 0.6 GB/s peak writes and a 256GB Samsung 960 EVO NVMe-SSD with 1.1GB/s peak write. HDDs are 4TB Seagate ST4000DM005 HDD with 180 MB/s peak write.

Considering that all SSDs in a WSN experience nearly identical workload behaviors as discussed in Section 3, we choose one SSD trace and one HDD trace as presentative from each type of WSNs. We replay one-hour traces on our server, and record the actual performance results as the baseline in the SWB mode. Figure 15 shows the performance comparison (i.e., average write latency per minute) between the replay results and their corresponding original trace results. Although there exist nearly constant performance gaps due to different hardware deployment between the original and test servers, the performance-varying trend tracks both cases very closely for each workload. Therefore, we believe that the replay approach is viable and fair to the evaluation.

SWR IO scheduler has three key parameters: the maximum block size threshold S_{max}, the SSD queue length threshold L and the reduction step value p, which are explained in Section 4.

First, we select the 99th-percentile block size of SSD-writes (i.e., external and internal SSD-writes) as the initial block size threshold S_{max}. For instance, S_{max} is 324KB for the B node and 32KB for $C1$ node. This is because the redirected writes should be tiny in number and large in request size. Consequently, only a very small number of writes may see their latencies increased moderately so that the HDD will very unlikely be overloaded. Large writes also efficiently leverage HDD's high sequential write throughput. Besides,

Table 5: The amount of redirected writes data and requests with SWR.

	A1	*A2*	*B*	*C1*	*C2*
SSD data written with SWB(GB)	7.3	24.1	126	11.1	14
SSD data written with SWR(GB)	2.5	13.4	37.8	6.2	7.7
Redirected requests(%)	1.8	1.0	0.7	2.0	2.0

we found that the large IO requests blocking the queue typically account for only 1.1% of all requests. As a result, the 99th-percentile size is a sensible choice. Second, the selection of the queue length threshold L has been explained in the previous section. The value of L is set to 6 for $A1$, 5 for $A2$, 30 for B, 40 for $C1$ and 57 for $C2$. Third, when the current SSD queue length continuously exceeds L, SWR will decrease the block size threshold S by the reduction step value p in proportion to S , namely, $p = \{0, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1\}$.

5.2 SSD-write Reduction

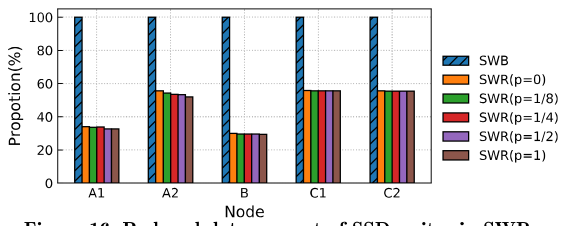


Figure 16: Reduced data amount of SSD-writes in SWR.

The design goal of SWR is to reduce the amount of data written to SSD, significantly mitigating wear-out of SSDs and decreasing SSD deployment requirement for cloud storage. We measure the amount of SSD-writes data in each node with SWB (baseline) and SWR respectively for comparison purpose. Figure 16 shows that SWR effectively reduces the amount data written to SSD, by 70% in B and about 45% in the other four nodes, compared to SWB. In addition, we observe that p has no effect on the write reduction. This is because only the bursty cases which are very rare can trigger the adjustment of S.

Table 5 compares the amounts of data written to SSD with **SWB** and **SWR** and reports the percentage of writes redirected by SWR. By redirecting less than 2% write requests from SSDs to HDDs, SWR is able to reduce 44%-70% of the data written to SSD. It also means that compared with

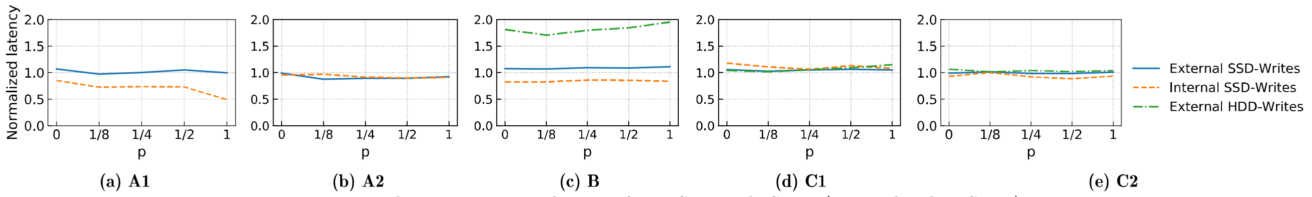


Figure 17: The average write latency for WSNs with *SWR* (Normalized to SWB).

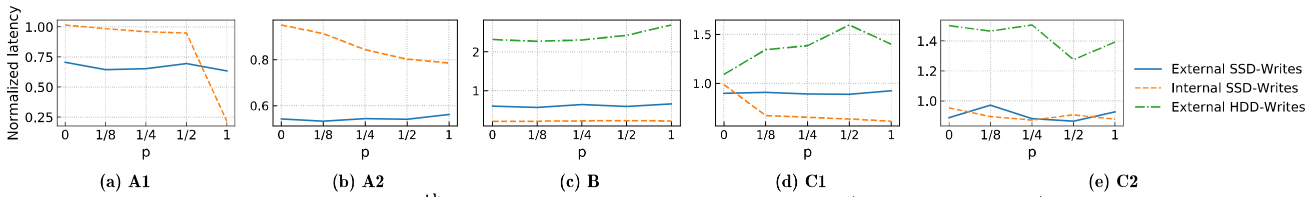


Figure 18: The 99^{th}-percentile write latency for WSNs of SWR (Normalized to SWB).

the SWB mode, SWR may indirectly increases the SSD lifetime by up to 70%. Because an SSD's lifespan is directly proportional to the amount of data written to it.

5.3 Write Performance

We measure the average and 99^{th}-percentile write latency with **SWB** and **SWR** in five WSNs. We take the SWB mode as the baseline and present the SWR performance normalized to that of SWB. We measure the average and 99^{th}-percentile tail of request write latency for each type of write operations to show the effect of SWR on the write performance. SWR is shown to reduce the average and 99^{th}-percentile latencies of all SSD-writes by up to 13% and 47% respectively.

Figure 17 shows the average write latency for each operation (except external HDD-writes in A set because they have less than 3 records in one hour) with SWR. We find that in almost all nodes, the performance differences of external SSD-writes between SWR and SWB are application-dependent and relatively small (i.e., between -10% and +13%). It indicates that SWR does not significantly affect the average latencies of WSNs.

Figure 17a, 17b and 17c show that, in $A1$, $A2$ and B, the average write latencies of internal SSD-writes with SWR are reduced by up to 52%, 11% and 19%. Because SWR reduces waiting delays for internal SSD-writes by redirecting external large IO writes. In the other two WSNs with SWR, the average write latencies of internal SSD-write are nearly the same as SWB.

Figure 17c shows that SWR can almost double the write latency of HDD-writes in B, from 5ms to 9.3ms. This is because the HDDs in B are much busier than in other WSNs. Figure 6 shows that the external HDD-writes in B account for much higher percentage of all external writes than other nodes. Therefore, those SSD-writes redirected to HDDs by SWR will compete with the original external HDD-writes, thus increasing the write latency. The HDD write competition reported here is actually grossly overestimated in this experiment where redirected SSD-writes are written to exactly one HDD instead of tens of HDDs in the B node.

Figure 18 plots the 99^{th}-percentile write latency of three write operations with SWR. Compared with the average latency, SWR has shown a significant and consistent positive effect on the tail latency performance. In Figure 18a through 18e, the 99^{th}-percentile write latencies of external SSD-writes with SWR are reduced by up to 37% for $A1$, 47% for $A2$, 44% for B, 12% for $C1$, and 14% for $C2$. Similarly, the tail latencies of internal SSD-writes are decreased obviously.

However, the 99^{th}-percentile write latency differences of external HDD-writes between SWR and SWB are bigger than that of the average latency. The tail latency with SWR is nearly three times that with SWB in B, from 66 ms to 160 ms. This result suggests that the HDD competition between external HDD-writes and redirected SSD-writes greatly aggravates the tail latency performance of HDD. However, the HDD-writes competition can be significantly alleviated by forwarding most of these writes to the remaining tens of HDDs in node B. To verify this approach, we add another HDD to relieve the IO burden of the single HDD. As seen in Figure 19, both the average and 99^{th}-percentile write latency decrease dramatically.

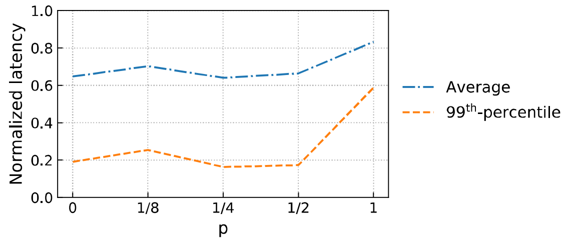


Figure 19: The average and 99^{th}-percentile write latency of External HDD-Writes of SWR scheduling upon two HDDs in node B (Normalized to SWB).

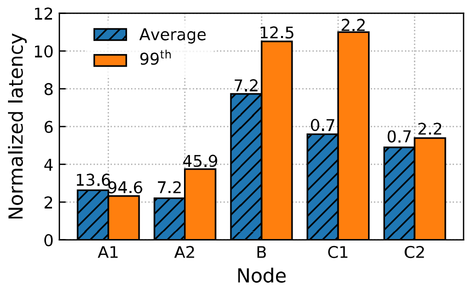


Figure 20: Redirected write latency with SWR normalized to SWB. The actual latencies (ms) are noted on top of the relevant bars.

Additionally, we also find that, the change of p has a significant impact on the 99th-percentile tail latency, especially for internal SSD-writes. Based on our earlier analysis, when the blocked requests are internal SSD-writes which are small and intensive, the block size threshold S decreases faster as p increases.This raises the possibility of redirecting more blocked internal SSD-writes. Therefore, a high value of p is actually beneficial for intensive and small requests, resulting in improved tail latency of internal SSD-writes. In contrast, p does not dramatically affect the performance of external SSD-writes, since it has a wide range of size distribution.

To better understand the latency of redirected writes (e.g., external and internal SSD-writes) with SWR, Figure 20 shows the write latency increases of the redirected writes in average and the 99th-percentile tail latencies. The relative latencies in these two cases are increased by up to 8 and 11 times respectively, compared to their original SSD-writes under SWB, due to the relatively long write-latency of HDDs. In the worst case, the average latency of such a few writes (0.7%) in B can increase from 0.94 ms with SWB to 7.29 ms with SWR, of which both the value and percentage are lower than the SLA objectives (e.g., 50ms at the average [13]) in clouds.

In summary, **SWR** provides significant benefits in reductions of both data written to SSDs and tail-latency. These benefits come at the expense of a tiny percentage of writes (up to 2% as shown in Table 5) whose latencies are increased. However, such a cost is arguably quite acceptable and necessary because it trades for vastly increased SSD performance and lifespans without violating the SLA. Additionally, the redirect conditions can also be adjusted, e.g., a larger size threshold and a longer queue length, to effectively reduce the number of redirected writes but with decreased benefit on reducing data written to SSDs. The cloud administrators or users can determine the tradeoff.

6 RELATED WORK

To our knowledge, no prior work presents the detailed write behavior analysis on hybrid storage nodes in production clouds. Our work makes up this research field. Nevertheless, our work is related to the following efforts. Next, we briefly discuss related work on hybrid storage and tail latency in storage systems.

Hybrid storage. Previous SSD-HDD hybrid designs mainly focus on using SSD as a cache layer [2, 10, 11, 32]. Griffin [23] employs HDDs as a write cache for SSDs to extend SSD lifetimes. Ursa [12] further uses SSDs for primary storage and HDDs as backup. **SWR** considers SSDs as write buffer while directly and selectively writing data into HDDs according to runtime disk queues.

Tail latency in storage systems. Several works improve read tail latencies [20, 24] by cache and data replication. RobinHood[1] repurposes the existing cache layer in the multitier system to directly address request tail latency by dynamically partitioning the cache. Earlier works [22, 28, 33] ensure latency SLOs using centralized components and/or complex modeling. [15] manages tail latency of datacenter-scale file systems. **SWR** focuses on write tail latency within WSNs and designs disk queue scheduling at runtime.

7 CONCLUSION

Hybrid storage nodes play a critical role in providing high performance and low cost for cloud providers. However, the behaviors of these nodes are not fully understood in real production clouds. In this paper, we analyzed real production traces from Alibaba Pangu, and found that some hybrid storage nodes have write-dominated workload behaviors. We revealed that current request serve mode in such nodes leads to SSD overuse and long-tail latency. Based on our findings, we proposed an IO scheduling mechanism and performed a trace-driven evaluation, which selectively forward large SSD write requests to HDDs and dynamically optimize for small and intensive burst requests.

8 ACKNOWLEDGMENTS

We would like to thank our shepherd, Virajith Jalaparti, and the anonymous reviewers for their valuable feedback and suggestion. This work is supported in part by NSFC No. 61872156, Creative Research Group Project of NSFC No.61821003, Fundamental Research Funds for the Central Universities No. 2018KFYXKJC037, the US NSF under Grant No.CCF-1704504 and No.CCF-1629625, National key research and development program of China (Grant No. 2018YFA0701804) and Alibaba Group through Alibaba Innovative Research (AIR) Program.

REFERENCES

[1] Daniel S. Berger, Benjamin Berg, Timothy Zhu, Siddhartha Sen, and Mor Harchol-Balter. 2018. RobinHood: Tail Latency Aware Caching – Dynamic Reallocation from Cache-Rich to Cache-Poor. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*.

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

[3] Wei Cao, Zhenjun Liu, Peng Wang, Sen Chen, Caifeng Zhu, Song Zheng, Yuhui Wang, and Guoqing Ma. 2018. PolarFS: an ultra-low latency and failure resilient distributed file system for shared storage cloud database. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1849–1862.

[4] Yunpeng Chai, Zhihui Du, Xiao Qin, and David A Bader. 2015. WEC: Improving durability of SSD cache drives by caching write-efficient data. *IEEE Transactions on computers* 64, 11 (2015),

3304–3316.

[5] Chanwoo Chung, Jinhyung Koo, Junsu Im, Sungjin Lee, et al. 2019. LightStore: Software-defined Network-attached Key-value Drives. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 939–953.

[6] Alibaba Clouder. 2018. Pangu – The High Performance Distributed File System by Alibaba Cloud. https://www.alibabacloud.com/blog/pangu_the_high_performance_distributed_file_system_by_alibaba_cloud_594059.

[7] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS operating systems review*, Vol. 41. ACM, 205–220.

[8] Samsung Electronics. 2018. Samsung V-NAND SSD Data Sheet. 5–6. https://s3.ap-northeast-2.amazonaws.com/global.semi.static/Samsung_SSD_860_EVO_Data_Sheet_Rev1.pdf.

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. (2003).

[10] Qi Huang, Ken Birman, Robbert van Renesse, Wyatt Lloyd, Sanjeev Kumar, and Harry C Li. 2013. An analysis of Facebook photo caching. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 167–181.

[11] Cheng Li, Philip Shilane, Fred Douglis, Hyong Shim, Stephen Smaldone, and Grant Wallace. 2014. c: A Capacity-Optimized {SSD} Cache for Primary Storage. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*. 501–512.

[12] Huiba Li, Yiming Zhang, Dongsheng Li, Zhiming Zhang, Shengyun Liu, Peng Huang, Zheng Qin, Kai Chen, and Yongqiang Xiong. 2019. Ursa: Hybrid Block Storage for Cloud-Scale Virtual Disks. In *Proceedings of the Fourteenth EuroSys Conference 2019*. ACM, 15.

[13] Ning Li, Hong Jiang, Dan Feng, and Zhan Shi. 2016. PSLO: enforcing the X th percentile latency and throughput SLOs for consolidated VM storage. In *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 28.

[14] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. 2012. SFS: random write considered harmful in solid state drives.. In *FAST*, Vol. 12. 1–16.

[15] Pulkit A Misra, María F Borge, Íñigo Goiri, Alvin R Lebeck, Willy Zwaenepoel, and Ricardo Bianchini. 2019. Managing Tail Latency in Datacenter-Scale File Systems Under Production Constraints. In *Proceedings of the Fourteenth EuroSys Conference 2019*. ACM, 17.

[16] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, et al. 2014. f4: Facebook's Warm {BLOB} Storage System. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*. 383–398.

[17] J. Ou, J. Shu, Y. Lu, L. Yi, and W. Wang. 2014. EDM: An Endurance-Aware Data Migration Scheme for Load Balancing in SSD Storage Clusters. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 787–796. https://doi.org/10.1109/IPDPS.2014.86

[18] Mayur R Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. 2008. Amazon S3 for science grids: a viable solution?. In *Proceedings of the 2008 international workshop on Data-aware distributed computing*. ACM, 55–64.

[19] Raghu Ramakrishnan, Baskar Sridharan, John R Douceur, Pavan Kasturi, Balaji Krishnamachari-Sampath, Karthick Krishnamoorthy, Peng Li, Mitica Manu, Spiro Michaylov, Rogério Ramos, et al. 2017. Azure data lake store: a hyperscale distributed file service for big data analytics. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 51–63.

[20] Waleed Reda, Marco Canini, Lalith Suresh, Dejan Kostić, and Sean Braithwaite. 2017. Rein: Taming tail latency in key-value stores via multiget scheduling. In *Proceedings of the Twelfth European Conference on Computer Systems*. ACM, 95–110.

[21] Alessandro Rubini and Jonathan Corbet. 2001. *Linux device drivers*. " O'Reilly Media, Inc.".

[22] David Shue, Michael J Freedman, and Anees Shaikh. 2012. Performance isolation and fairness for multi-tenant cloud storage. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*. 349–362.

[23] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. 2010. Extending SSD Lifetimes with Disk-Based Write Caches.. In *FAST*, Vol. 10. 101–114.

[24] Lalith Suresh, Marco Canini, Stefan Schmid, and Anja Feldmann. 2015. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*. 513–527.

[25] Linpeng Tang, Qi Huang, Wyatt Lloyd, Sanjeev Kumar, and Kai Li. 2015. RIPQ: Advanced Photo Caching on Flash for Facebook. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. USENIX Association, 373–386.

[26] Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan Gómez-Luna, and Onur Mutlu. 2018. FLIN: Enabling fairness and enhancing performance in modern NVMe solid state drives. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 397–410.

[27] Seagate Technology. 2018. BARRACUDA COMPUTE Product Manual. 8–8. https://www.seagate.com/www-content/product-content/barracuda-fam/barracuda-new/en-us/docs/100804187g.pdf.

[28] Andrew Wang, Shivaram Venkataraman, Sara Alspaugh, Randy Katz, and Ion Stoica. 2012. Cake: enabling high-level SLOs on shared storage systems. In *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 14.

[29] Hui Wang and Peter Varman. 2014. Balancing Fairness and Efficiency in Tiered Storage Systems with Bottleneck-Aware Allocation. In *Proceedings of the 12th {USENIX} Conference on File and Storage Technologies ({FAST} 14)*. 229–242.

[30] Yeong-Jae Woo and Jin-Soo Kim. 2013. Diversifying wear index for MLC NAND flash memory to extend the lifetime of SSDs. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*. IEEE Press, 6.

[31] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A Chien, and Haryadi S Gunawi. 2017. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs. *ACM Transactions on Storage (TOS)* 13, 3 (2017), 22.

[32] Ke Zhou, Si Sun, Hua Wang, Ping Huang, Xubin He, Rui Lan, Wenyan Li, Wenjie Liu, and Tianming Yang. 2018. Demystifying Cache Policies for Photo Stores at Scale: A Tencent Case Study. 284–294. https://doi.org/10.1145/3205289.3205299

[33] Timothy Zhu, Alexey Tumanov, Michael A Kozuch, Mor Harchol-Balter, and Gregory R Ganger. 2014. Prioritymeister: Tail latency qos for shared networked storage. In *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 1–14.