Summer Research Report on BCW: Buffer-Controlled Writes to HDDs for SSD-HDD Hybrid Storage Server

Nikhil Mehta (201801030) , Vatsal Gujarati (201801162)
Mentored by Prof. Jay Prakash Lalchandani
Dhirubhai Ambani Institute of Information and Communication Technology Gandhinagar, Gujarat, India

I. Introduction

Nowadays, storage clouds have deployed a hybrid storage server integrating HDDs and SSDs. Amazon, Alibaba, and Facebook have done so. This type of hybrid storage server uses an SSD-HDD tiered architecture to get the most out of both SSDs and HDDs' higher IO performance and bigger storage, resulting in great performance and cost-effectiveness. Incoming writes are first quickly steered to SSDs followed by flushing it to HDDs at a later time.

Observations have been made that SSDs are commonly overused, whereas HDDs are used at less than 10% of the time, allowing HDDs to be underutilised in terms of speed and capacity. But the overuse of SSDs also leads to fast wear out of them. It is because writes are known to be unfriendly to SSDs for two reasons. First, SSDs have a finite number of P/E (Program/Erase) cycles, which are proportional to the number of writes. Second, garbage collections cause SSDs to suffer from unpredictably severe performance degradations. Therefore, cloud providers must deploy additional and/or larger SSDs to ensure stable write performance of storage servers in write-heavy workloads, which will greatly increase the cost.

Experiments have shown that a sequence of continuous and sequential short HDD writes (e.g., 4KB) have low latency (e.g., 35µs) for about 60 ms, then a drastically escalated high latency (e.g., 12ms), then medium latency (e.g., 55µs) for about 40 ms. Fast, mid, and slow writes are the three states of write behaviours exhibited by the HDDs. Because the initial incoming writes are regarded complete and acknowledged (to the host) after their data has been written into the controller's built-in buffer, the first two sorts of writes can achieve µs-level responsiveness. When the write buffer is full, host writes are halted until the buffered data is flushed to the disc, resulting in slow writes. This motivates us to use HDDs to their fullest potential. However, implementing buffered writes in HDDs to take advantage of fast and mid writes is difficult due to the difficulties in anticipating when these write states will occur. The host is fully unaware of the internal buffer and other components of HDDs. The host can only determine the current write state based on its own latency, but it cannot predict future write states.Therefore, we build a prediction model for sequential and continuous write patterns that predicts the next HDD write state. Then we suggest using a Buffer-Controlled Write (BCW) method. According to the predictor and runtime IO monitoring, BCW can proactively and effectively control the buffer write behaviour. Besides, BCW also actively "skip" slow writes by filling padded data during HDD slow writes. Then after, we have a Mixed IO scheduler(MIOS) for SSD-HDD hybrid storage by leveraging the BCW approach. Depending on write states, runtime queue length, and disc status, MIOS adaptively routes incoming writes to SSDs or HDDs.

Main contributions of the research paper were as follows :

1) Because of the buffered write feature in HDDs, it was discovered through extensive experimental studies on HDD write behaviours that there exists a periodic staircase-shaped write latency pattern consisting of s level write latency (low and mid write states) followed by ms-level write latency (slow write state) upon continuous and sequential writes.
2) According to the prediction, the buffer-controlled write (BCW) technique proactively triggers continuous and sequential write patterns. Data padding is also used by BCW to deliberately avoid, or skip, sluggish writes for the host.
3) The SSD-HDD mixed IO scheduler (MIOS) improves the overall performance of SSD-HDD hybrid storage servers while lowering write traffic to SSDs.

II. SSD-HDD Hybrid Storage

SSD-HDD hybrid storage has emerged as an unavoidable alternative for cloud providers in order to meet exponentially expanding storage requirements while maintaining overall cost-effectiveness. The majority of providers, such as Google, anticipate increased storage capacity and improved performance at a reduced cost. They are progressively embracing storage heterogeneity, installing variable types and numbers of SSDs which provide low IO latency as the primary tier and HDDs which provide larger capacity at the secondary tier. The fast SSD tier typically serves as a write buffer, allowing

incoming write data to be stored quickly before being flushed to the slower but bigger HDD tier. As a result, the SSD tier absorbs the majority of foreground application write activity.

A. Write Behavior of Hybrid Storage

Many backend storage servers in cloud must accommodate write dominated workloads because write intensive workloads widely exist in many production environments. Pangu is a distributed large-scale storage platform and provides cost-effective and unified storage services for Alibaba Cloud. A comprehensive workload analysis of Pangu was that :

1) The majority of requests are writes. The amount of data written is 1-2 orders of magnitude bigger than the amount of data read from the servers. More than 77 percent and up to 95 percent of requests are writes. In fact, each SSD writes roughly 3 TB of data per day, which is close to the DWPD (Drive Writes Per Day) limit, which rigorously regulates the amount of SSD data produced daily for reliability.
2) The amount of data written to SSDs versus HDDs is vastly different. For example, average SSD IO utilisation is up to 28.5 percent, and HDD IO utilisation is less than 10%. Despite this, the majority of the HDD capacity is used to dump SSD data, with user requests being served rarely

B. Challenges

An easy method to reduce SSD pressure from write-heavy workloads is to increase the number of SSDs in hybrid nodes. This is, however, a costly option because it raises the total cost of ownership. When SSDs are overused, a solution is to utilise the highly underutilised HDD IO capacity in hybrid storage servers. The state-of-art solution SSD-Write-Redirect (SWR) redirects large SSD writes to idle HDDs. To some extent, this strategy can help with the SSD queue blockage problem. The IO delays incurred by requests routed to HDDs, on the other hand, have been demonstrated to be 3-12 times larger than those experienced by requests redirected to SSDs. For most modest writes that require µs-level latency, this is clearly unsatisfactory. The main problem is to get HDD IO delay down to the s-level, which is comparable to SSDs.

III. HDD Write Behaviors

A. Buffered Writes

Continuous and sequential writes were run on HDDs, which is the most friendly write pattern for HDDs, to better understand HDD write behaviours and estimate the prospect of reaching s-level write latency on HDDs. A series of continuous and sequential writes with the same IO size were written to an HDD as part of the "profiling" procedure to analyse precise HDD characteristics. The observations were as follows :

1) Series of continuous sequential write requests experience a similar sequence of three level write latency, i.e., low, mid, and high latencies, forming a staircase-shaped time series.
2) HDD write behavior is periodic. At the beginning (right after the buffer becomes empty) low latency writes last for a period, which is followed by a spike (high-latency writes) and then mid-latency writes. If the write pattern is continuous, high-latency writes and mid-latency writes will appear alternately.

The reason behind such observations is that Modern HDDs deploy a built-in DRAM (e.g., 256MB for the 10TB and 8TB HDDs). Based on observations, only a portion of the DRAM can be used to buffer incoming write IOs. The HDD's built-in DRAM's remaining capacity can be employed as a read-ahead cache. When a write is successfully buffered, HDD notifies the host that the request has been completed. When the buffered data exceeds a certain threshold, the HDD must cause the buffered data to be flushed into their original positions on the disc. Incoming writes must be prevented during this time until the buffer is freed up again. As data is flushed to the disc, the HDD buffer may become empty implicitly. However, we can use sync() to force flushing and manually empty the buffer.

B. HDD Buffered Write Model

To formally characterize the HDD write behavior, an HDD buffered-write model was made. Figure 1 illustrates the schematic diagram of the HDD buffered-write model in the time dimension (runtime). The x axis represents the time sequences of transitions among the three write levels, with each sequence being started by a "Sync" event.

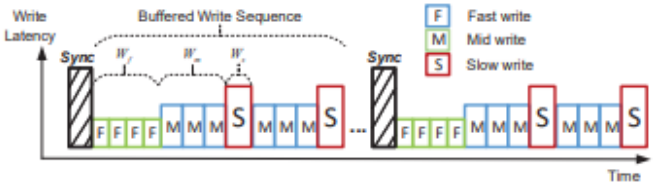

Fig. 1. The HDD Buffered-Write Model with two complete Buffered Write Sequences

A Buffered-Write Sequence is made up of the three types of HDD buffered writes discussed above, namely Fast (low-latency), Mid (mid-latency), and Slow (high-latency) writes, denoted by F, M, and S, respectively. F, M, and S are the states that a write request can be in according to the model. As described in Figure 2, these IO delays are denoted as Lf , Lm and Ls, respectively.

Parameters	Description
$L_{f/m/s}$	The IO delays of write requests in the F/M/S write states
$W_{f/m/s}$	The cumulative amount of written data for the Fast/Mid/Slow Stages
$T_{f/m/s}$	The time duration of the Fast/Mid/Slow Stages
s_i	The IO size of write request i

Fig. 2. The list of descriptions about all the parameters in the HDD Buffered-Write Model

The F state indicates that an incoming write request wi of size si can be completely buffered in the HDD's built-in DRAM buffer. The M state indicates that the write buffer is nearly full. The S state indicates that the write buffer is full, blocking any incoming write requests. A Buffered Write Sequence is made up of a Fast stage and one or more Slow-and-Mid stage pairings. When there is enough buffer for the Fast stage, the sequence begins. It comes to an end when the present series of continuous writes is completed. The Fast, Mid, and Slow Stage last for Tf ,Tm, and Ts respectively, which are determined by the cumulative amount of written data Wf ,Wm, and Ws in the respective states.

Example:
In the 10TB HDD with 64KB write requests :

1) Lf=180s,Lm=280s,Ls=12ms,Tf=60ms,Tm=37ms, Ts=12ms.
2) Wf=16MB,Wm=8MB. Ws depends on the IO size si.

According to the HDD buffered-write model, the Fast and Mid writes of HDD have 100-s-level latency, which can approach the write latency of SSDs.

This inspires us to create a controlled buffer write technique for HDDs in hybrid storage systems to reduce writes on SSDs without losing overall performance.

IV. Design

The main two challenges which we need to encounter to exploit HDD buffered writes are:

1) The first problem is how to determine the write state that a write request will be Fast (F), Mid (M), or Slow (S), in order to properly schedule the write request. For the first problem, we build a Write-state Predictor to pre-determine the next write state based on the current write state and buffer state. Now, based on this we now propose BCW, a writing approach to proactively activate continuous and sequential write patterns that the predictor relies on, as well as effectively control the IO behavior according to the predictor and IO monitoring. Now to overcome the performance degradation caused due to the writes in the S state we propose a proactive padding-write approach to skip the S state by padding non-user data during the slow writes.
2) The second problem which is faced is how to steer an incoming write request to the HDD from an SSD without any performance degradation. For overcoming this issue we propose a Mixed Input output scheduler (MIOS), which would determine where to steer a write request.

A. Write - State Predictor

In HDD buffered write, each write request should be one of F,M,S state.The HDD write state is considered either in A (available) or U (unavailable). When the HDD write state is A then it means that the current Accumulative Data Written (ADW) in the F or the M state is less than Wf or Wm. Otherwise if the value of ADW would have been greater than Wf and Wm then the HDD would have been in the U state. The prediction of the next write state depends on the write buffer's free space and the write state of the current request.

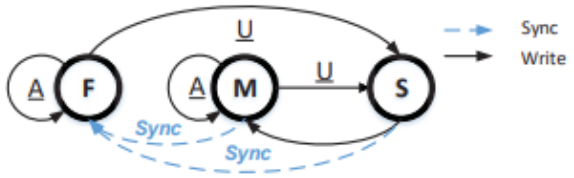

Fig. 3. State Prediction diagram

For any write request there are three write states F,M and S. A means that the value of ADW is less than Wf and Wm. Otherwise the state would have been U. Whenever the sync function is called then it takes the state back to F state.
The processes that are happening above can be described as :

- F / A : The current write state is F and the buffer is available. Next write state is most likely to be F.
- F/U : Although the current write state is F, the buffer is unavailable. Next write state is likely to change to S.
- M/A : The current write state is M and the buffer is available. Next write state is most likely to remain M.
- M/U : Although the current write state is M, the buffer is unavailable. Next write state should be S.
- S : The current write state is S. Next write state will be M with a high probability.
- The sync function shown above would force the next write state and buffer state back to F/A.

Based on the above shown diagram we can design a write state predictor which would give us the next write state based on the buffer space available and the current write state.

Algorithm 1 The algorithm of Write-state Predictor

Input: Current write request size: *size*; The last write state: *state*; Current accumulative amount of data written: *ADW*;
The amounts of data written in the *F* state and *M* state: W_F and W_M
Output: Write-state prediction for the next request (*F*, *M* or *S*)

```
function Predictor()
if state == F then
    if (ADW + size) < W_f then : return F
    else: return S
    end if
else if state == M then
    if (ADW + size) < W_m then : return M
    else: return S
    end if
else: return M
end if
```

Fig. 4. The Algorithm of Write State Predictor

B. Buffer-Controlled Writes

BCW is an HDD writing approach which ensures that users would only write in the F and M state and would avoid writing in the S state. It has been seen that many a time the buffered writing becomes uncontrollable and for overcoming that issue we propose a write approach which is controllable.

When BCW is turned on, a sync() operation is made to run to force synchronisation and actively empty the buffer. If the HDD is projected to be in the F or M state, BCW sends sequential user writes to it. Otherwise, non-user data is padded to the HDD. If there are many user requests in the queue then first HDD writes the then serially and after it is completed then it adds it to ADW and then would update the write state accordingly.

HDD works best when the data is continuously written in it and for ensuring this during the light and the idle workload periods when the number of requests are very less, the HDD write request queue would be empty from time to time and for making the data look continuous the BCW function pads non user data at that time. If the state is F or M then it pads PF and in the S state it pads PS amount of data. The value of PS is greater than that of PF.A small PF can minimize the waiting time of user requests and a larger PS helps trigger the slow write quickly. In the current condition of F or M state, BCW calculates ADW on a continual basis. When ADW approaches Wf or Wm, the HDD buffered write is nearing the end of the Fast or Mid stage. After multiple writes, the S write state may appear. BCW notifies the scheduler at this point and initiates the Slow write with PS. During this time, all incoming user requests must be routed to other storage devices, such as SSDs, to avoid long latency for user writes. According to the write-state predictor, after an S write is performed, the next write will be M. The ADW is then reset, and BCW accepts user requests once more.

Algorithm 2 The algorithm of Buffer-Controlled Write

Input: The max loop of Buffered Write Sequence: $Loop_{max}$
Request R_i size: $size_i$; Current written amount: ADW;
The state of last write: $state$;
Active padded writes and their size: PS,PF and $size_{PS}$,$size_{PF}$

```
1: sync()
2: while loop < Loop_max do
3:     if request R_i in the HDD write queue then
4:         write R_i to HDD, update ADW and state
5:     else
6:         if Predictor() == S then
7:             flag_HDD = False    // Stop receiving
8:             while state == S do
9:                 write PS to HDD, update ADW and state
10:            end while
11:            flag_HDD = True    // Start receiving
12:            reset ADW; loop++
13:        end if
14:        if Predictor() == M then
15:            write PF to HDD; update ADW and state
16:        end if
17:    end if
18: end while
```

Fig. 5. The Algorithm of Buffer-Controlled Write

The sequential and continuous write pattern created by BCW is relatively stable in most circumstances. This trend, however, can be interrupted, as evidenced by HDD reads. Slow writes can also be caused by user requests or PF writes in advance. BCW constantly runs PS until an S write state is recognised to reclaim control of buffered writes. The write-state predictor will be re-calibrated as a result. BCW wastes IO and storage on the HDD to conduct PS writes, which would be the cost of this method. Furthermore, we can use sync() to clear the buffered write state in BCW. A sync(), on the other hand, can take several hundred milliseconds, during which time the HDD is unable to receive any writes. Fortunately, we discovered that BCW interrupted cases are uncommon in the experiment.
The algorithm for BCW with reference to the above discussion is present in Figure 5.

C. Mixed IO Scheduler

Architecture : At runtime, MIOS watches all SSD and HDD request queues, wisely starts the BCW process, and determines if a user write should be directed to a particular HDD or SSD. During the configuration procedure, MIOS creates a device file on each HDD. The BCW writes are stored in the device file in an append-only format.

Scheduling Strategy : The parameters that are important for writing in the SSD are its queue length at any time l(t) and its threshold length L. If the value of l(t) is greater than the value of L then the MIOS function will steer the write requests to the HDD if the write state of the HDD is F or M. We simply set the threshold L to the lowest l if the request with queue length l has a latency greater than the IO delay of the HDD in the M state. The logic is that when the SSD queue length is more than L, the SSD writes will have latencies comparable to those of an HDD in the F or M write state with BCW. The value of L can be found experimentally and adjusted according to the workload conditions.

Triggering BCW is also optional when the queue length of SSD is less than L. MIOS_E or MIOS_D indicate if BCW is enabled or disabled in this situation. In other words, the MIOS_E technique supports BCW redirection when the SSD queue length is less than L. MIOS_D on the other hand disables redirection to the HDD when the SSD queue length is less than L. We should also keep in mind that write latency of HDD in M state is way more than that of SSD. Typically, a hybrid storage node will have many SSDs and HDDs. All drives are divided into separate SSD/HDD pairs, each including an SSD and one or more HDDs. An independent MIOS scheduler instance manages each SSD/HDD combo.

Finally, MIOS necessitates total control over HDDs. It indicates that other IO operations cannot affect the HDDs in BCW. When a read request arrives when an HDD is running BCW, MIOS suspends BCW and serves the read. At this moment, it will attempt to reroute all writes to other idle discs. BCW can be turned off in read-dominated workloads to avoid interfering with reads.

Based on the above shown feature for the MIOS function, an algorithm for it is shown below:

Algorithm 3 The algorithm of Mixed IO Scheduler

Input: SSD queue length at time t: $l(t)$;
Queue length threshold: L; HDD available flag: $flag_{HDD}$;
Schedule Strategy: *MIOS_D* or *MIOS_E*

```
if (flag_HDD == True) then
    if l(t) > L && Predictor() == F or M then
        Send to HDD queue
    else if MIOS_E && Predictor() == F then
        Send to HDD queue
    else: Send to SSD queue
    end if
else: Send to SSD queue
end if
```

Fig. 6. The Algorithm of Mixed IO Scheduler

V. Conclusion

Some hybrid storage servers cater to write-heavy workloads, resulting in SSD usage and long-tail delay while HDDs go unused. However, experimental research shows that with adequate buffered writes, HDDs can achieve s-level write IO latency. As a result of request redirection, HDDs were used to offload write demands from overloaded SSDs. By selecting fast writes for user requests and padding non-user data for slow writes, the Buffer-Controlled Write(BCW) method was utilised to proactively regulate buffered writes. Then, based on runtime monitoring of request queues, a mixed IO scheduler was employed to autonomously direct incoming data to SSDs or HDDs.

References

[1] Wang, Shucheng, et al. "BCW: Buffer-Controlled Writes to HDDs for SSD-HDD Hybrid Storage Server." 18th USENIX Conference on File and Storage Technologies (FAST 20). 2020.

[2] Gao, C., Shi, L., Zhao, M., Xue, C. J., Wu, K., Sha, E. H. M. (2014, June). Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives. In 2014 30th Symposium on Mass Storage Systems and Technologies (MSST) (pp. 1-11). IEEE.

[3] Deng, F., Cao, Q., Wang, S., Liu, S., Yao, J., Dong, Y., Yang, P. (2020, August). SeRW: Adaptively Separating Read and Write upon SSDs of Hybrid Storage Server in Clouds. In 49th International Conference on Parallel Processing-ICPP (pp. 1-11).

[4] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. Extending ssd lifetimes with disk-based write caches. In FAST, volume 10, pages 101–114, 2010.

[5] Jiang, C., Han, G., Lin, J., Jia, G., Shi, W., Wan, J. (2019). Characteristics of co-allocated online services and batch jobs in internet data centers: a case study from Alibaba cloud. IEEE Access, 7, 22495-22508.

[6] Pan Yang, Ni Xue, Yuqi Zhang, Yangxu Zhou, Li Sun, Wenwen Chen, Zhonggang Chen, Wei Xia, Junke Li, and Kihyoun Kwon. Reducing garbage collection overhead in SSD based on workload prediction. In 11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19), 2019.