CLAP: The Unwritten Contract for Distributed File Systems on SSDs

Anshu Verma Arjun Balasubramanian

University of Wisconsin - Madison

Abstract

The distributed systems of today are designed using the guidance of the *CAP theorem*. Distributed systems need to be able to tolerate network partition and hence the CAP theorem boils down to a trade-off between *consistency* and *availability*. In this work, we show that the *CAP Theorem* is no longer self-sufficient to serve as the holy grail for distributed system design in wake of *modern storage media* such as SSDs. Due to their unique wearing characteristics, it is essential to be able to reason about SSD cluster lifetime while designing distributed systems. In this work, we propose a new ***CLAP Theorem*** to encapsulate *lifetime* within the CAP theorem. Through carefully crafted simulation experiments over a variety of *modern workloads*, we showcase how different strategies for consistency can have huge effects on the lifetime of the SSD cluster.

1 Introduction

The world is today moving away from Hard Disk Drives (HDDs) having mechanical moving parts and instead embracing NAND-based flash memory. This is evident from the rising popularity of Solid State Drives (SSDs). SSDs have already been widely deployed both in datacenter and personal computer settings due to their ability to provide high throughput and low latency. On similar lines, Non Volatile Memory (NVM) is expected to substitute or complement DRAM in the memory hierarchy. This has prompted several studies [17, 24] on the performance characteristics of these devices. However, one of the issues with flash memory is with its reliability. Flash memory wears out with writes and has high error rates [3]. Techniques like wear-leveling and data scrubbing are used to overcome these problems. These techniques however exhaust the limited P/E cycles of flash memory pages which leads to reduced lifetimes for these devices.

In a tangential world, the growing need for processing large amounts of data has led to the development of many big data analytic frameworks [1, 11, 14, 31, 32, 35, 37, 45, 46]. These applications rely on *distributed storage systems* [12, 36, 43, 48] to store their data. Additionally, distributed configuration stores like ZooKeeper [20] require storage media for holding configuration and states in a fault-tolerant manner. The distributed storage systems in turn interact with storage media such as HDDs and SSDs to actually store data.

Different distributed systems adopt different policies to store data or state. A myriad of options exist because different applications require different semantics for *data consistency*. Consequently, different systems provide options to trade-off on varying level of consistency with performance [8, 27, 42]; strong consistency guarantees would require synchronous replication of data leading to degraded performance. The foundation for this trade-off is embedded in the popular CAP theorem [13]. The CAP theorem has driven the design of many popular systems and is used as a guiding principle to develop distributed systems.

Traditional failure models for distributed systems characterize failure with the underlying assumption that disks can fail *independently* [38]. However, when distributed systems are deployed on top of flash-based media like SSDs, this assumption no longer holds true. Since SSDs gradually wear out with writes, the relative rate at which SSDs wear out in a distributed setting depends on the distribution of the workload across the SSDs; an SSD that takes on a larger proportion of the workload is likely to fail quicker than others in the cluster. Such a scenario makes the cluster doomed for faster failure. This is undesirable because SSDs would have to replaced at a faster rate leading to greater cost of maintenance. Consequently, in a distributed scenario, wear-leveling within an SSD is no longer sufficient; there is also a need to balance out the wearing among the SSDs in the cluster.

Hence, there is a need to deconstruct the impact of different replication policies with varying levels of consistency on the lifetime of the underlying SSD cluster. In this work, as a preliminary effort, we examine the impact of different distributed storage policies on the lifetime of the underlying cluster consisting of SSDs. We focus on two classes of distributed systems - one which adopts a stronger form of consistency like Google File System (GFS) [12] and one

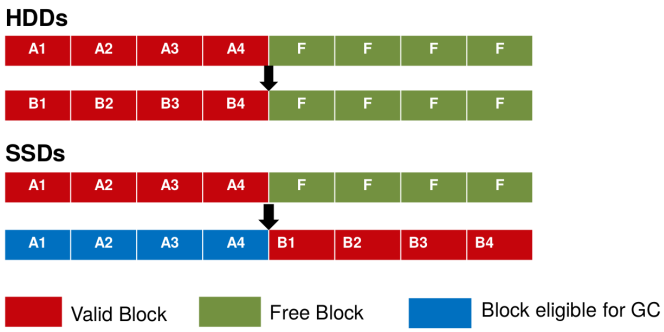


Figure 1: Difference in handling of writes in HDDs and SSDs

which adopts weaker forms of consistency like Ceph [48]. Through simulations of these schemes, we show that *stronger consistency policies promote better cluster lifetimes*. More importantly, it means that highly performance systems that employ weak consistency guarantees limit the lifetime of the SSD cluster on top of which they are deployed.

As a consequence, the CAP theorem is no longer sufficient to encapsulate the trade-offs of distributed system design in the wake of emerging storage media. To serve as a guiding light for system designers, we believe that the CAP theorem must be modified to include the trade-offs incurred in lifetime with varying levels of consistency. As a novel contribution, we propose a new ***CLAP theorem***, a soft addition of *Lifetime* to the CAP theorem for emerging storage media. In the context of this work, we do not attempt to prove the validity of the newly proposed *CLAP theorem*. We wish to propose the new *CLAP* theorem as a design panacea and leave proofs to future work. Hence, we introduce the *CLAP theorem* as an unwritten contract that distributed file systems must adhere to for emerging storage media like SSDs.

The rest of this paper is arranged as follows. Section 2 presents a background on SSDs, Distributed Systems, and CAP Theorem. Section 3 outlines the impact of different consistency models on the lifetime of an SSD cluster. Finally, Section 4 presents empirical results which leads to some lessons on distributed storage system design for better lifetimes.

2 Background

2.1 Solid State Devices

SSD Fundamentals

Flash chips aboard SSDs are commonly composed of blocks, which are typically in the order of hundreds of KBs or larger. A block consists of pages, which usually range from 2 KB to 16 KB in size. Three kinds of operations are supported by flash chips: read, erase, and program(or writes). Reads and writes are permitted at the granularity of a page, whereas an erasure is permitted only at the block level.

How are overwrites handled in an SSD?

Figure 1 shows a toy example of the difference between how writes are handled in HDDs and SSDs. In our setting, let us assume that each block has 4 pages. Initially, one block consists of data $A(A1, A2, A3, A4)$. Now, consider a scenario where data A needs to be overwritten with data $B(B1, B2, B3, B4)$. In HDDs, the updates can be performed in-place. However in SSDs, the contents would need to be written out to a new block and the old contents need to be marked as invalid. Invalid blocks are then reclaimed by a process known as *garbage collection*.

Role of Flash Translation Layer

The complexity of SSD internals is hidden by a software layer on the SSD called the *Flash Translation Layer (FTL)*. The FTL exposes a simple block interface to the upper layers. Because updates cannot be made in place, the FTL needs to hold a mapping between the logical block location from the client's perspective to the actual location of the block on the SSD.

In addition to this, the SSD also hosts a *Flash Memory Controller* which performs the below functions -

- **Garbage Collection.** The Flash Memory Controller runs a background garbage collection whenever the number of free blocks drops below a built-in threshold. It reclaims blocks by copying the valid pages in blocks into programmable blocks and then performing an erase on the block. This reclamation procedure ensures that blocks are ready to be programmed and can take up new writes.

- **Wear leveling.** Flash memory pages can endure a limited number of erase operations. The lifetime of the SSD would be shorten if certain pages wear out quickly in comparison to others. Hence, the Flash Memory Controller attempts to balance out the wear among pages in a process called wear-leveling.

Sources of Error in Flash Memory

As outlined in [23], there are 3 major sources of error in flash memory -

- **Wear.** Repeated erases wear out flash memory cells that store electrons and causes irreversible damage to them [4]. Each flash memory block has an endurance limit and this limit is not fixed across all blocks of the SSD due to variations in manufacturing.

- **Retention Loss.** Electrons stored in flash memory cells can leak over time, causing errors when data is read. The error rate increases as cells wear [4, 34]. Interestingly, these errors are transient; i.e. they get reset once the block is erased.

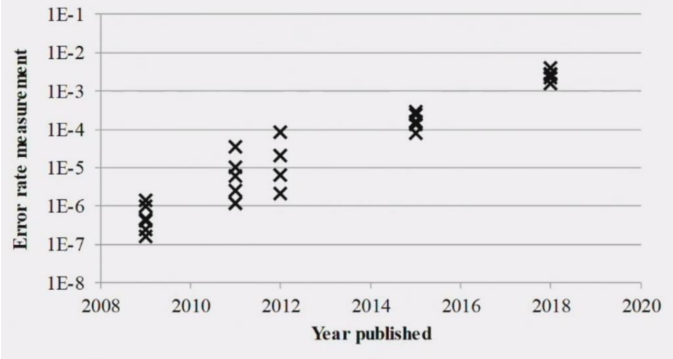


Figure 2: Progression of error rates in flash memory over the years. Data taken from [23]

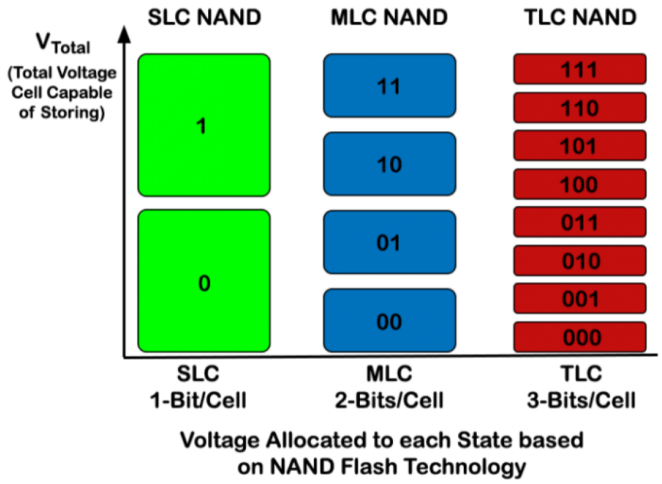


Figure 3: Differences between SLC, MLC, and TLC

- **Disturbance.** A read on a wordline in a block can weakly affect other wordlines in the block [4, 34] resulting in the *disturbance* of electrons in the flash cell. The amount of disturbance increases as cells wear out. Like retention loss, this kind of error is transient in nature as well.

Why is Flash Memory Error increasing over the years?

Figure 2 shows the progression of error rates associated with flash memory over the years. It is pretty counter-intuitive that errors are actually increasing over the years. The reason for increasing errors is in that hardware manufacturers are trying to reduce the size of flash cells by packing more bits into a single cell. As visible in Figure 3, a Single-Level Cell(SLC) flash consists of a single bit per cell, whereas MLCs and TLCs have two and three bits per cell respectively. Greater packing leads to more storage capacity per unit area but increases the chances of retention loss and disturbance errors.

Handling flash memory errors

SSD manufacturers provide additional storage capacity within each page to store Error Correction Codes (ECCs). Whenever a page is read, it is checked against the ECC for that page to check if the data was read correctly. If the data was not read correctly and if error correcting mechanisms fail, then the FTL attempts to retry reading the page.

To avoid data loss, the FTL performs an activity called *data scrubbing* to relocate pages that could be prone to read errors. Post this relocation, the pages become eligible for garbage collection. Once collected and erased, the page will no longer be prone to read errors since retention and disturbance errors are transient. In effect, the background data scrubbing reduces the chance for read errors but negatively impacts performance and accelerates wear.

2.2 Distributed File Systems

Different distributed file systems adopt different replication strategies that utilize varying levels of consistency.

Google File System (GFS) [12] is built on master-slave architecture with a single master holding the meta-data information in the memory. The metadata server stores access information, lock information, mapping of files to chunk id, chunk locations, and primary server for write propagation. When a client requests a read with filename and byte location, it first contacts the GFS master and obtains the chunk ID and a list of chunk locations. Similar to HDFS, the client then chooses the closest replica to do the read from. When a client issues a write request, it first contacts the GFS master to get the primary server and list of replicas. The client initiates the write to the primary and the primary determines the optimized topology to push down the writes to all replicas using chain replication [47]. In a certain sense, GFS allows for a *stronger* form of consistency by allowing reads from any of the replicas.

Ceph [48] improves scalability by assigning the responsibility of object allocation to CRUSH. Files are striped into objects and CRUSH uses a pseudo-random algorithm to determine the storage server. This approach helps to reduce the load from the meta-data server as well as ensuring the storage is distributed uniformly across all the storage devices. When a Ceph client opens a file, the request goes to Metadata server and it returns the inode number, file size, access information and striping strategy used to map objects into storage devices. It's imperative to note that the Ceph serves reads from primary and all the writes to the primary and replicas are written in a synchronous manner. In a certain sense, Ceph has a *weaker* form of consistency by allowing reads only on the primary replica.

2.3 CAP Theorem

The CAP Theorem states that in a distributed system, one can only have two out of the following three guarantees across a

write/read pair: Consistency, Availability, and Partition Tolerance; one of them must be sacrificed.

3 CLAP: The Unwritten Contract

Lifetime of Cluster

We incorporate a strict definition for the *lifetime* of a cluster. We define *lifetime* as the amount of time it takes for a *single page* in the cluster of SSDs to reach its endurance limit. Now, we outline the reason for this choice of definition for *lifetime*.

Let us say that the we have N SSDs in the cluster $SSD_1, SSD_2, ..SSD_N$. For sake of simplicity, let us say that each SSD_i for $i = 1, 2...N$ has a uniform number of pages - M. We refer to page m on SSD_i as $Page_{i,m}$.

For each SSD_i, we can take the page $Page_{i,m}$ for $m = 1, 2..M$ that has endured maximum wear as representative of that SSD. This is because the SSD will lose its reliability and degrade very fast when at least one page in it has reached its endurance limit. Let us say that the pages with maximum erase operations in the SSDs are $MaxErase_1, MaxErase_2..., MaxErase_N$ corresponding to $SSD_1, SSD_2, ..SSD_N$ respectively.

The distribution of $MaxErase_1, MaxErase_2..., MaxErase_N$ can help determine the lifetime of the cluster. If a particular $MaxErase_i$ dominates, then that SSD becomes a hotspot for failure and consequently for the failure of the entire cluster. Hence, we prefer distributions where the $MaxErase_i$ is nearly the same among all SSDs.

The impact on lifetime depends upon - (i) The relative placement of replicas of different chunks. (ii) The policy for handling read requests, i.e., which copy of the replica to read from. (iii) The distribution of write requests amongst chunks.

We now look at the impact of *strong* and *weak* consistency protocols on the lifetime of SSDs leading up to the ***CLAP Theorem***.

Strong Consistency Protocols

With strong consistency protocols, all of the replicas of data chunks are persisted durably during a write operation. Read requests can be routed to any copy of the replica. Hence, in general, *strong* consistency protocols ensure that read requests can be distributed uniformly among replicas leading to the SSDs holding them to endure *uniform wear*.

Weak Consistency Protocols

With weak consistency protocols, only a few replicas of chunks(sometimes even just a single replica) are persisted. Read requests are usually routed to only the primary replica. Hence, in general, *weak* consistency protocols ensure that read requests are skewed towards the primary replica, leading them to endure *non-uniform wear*.

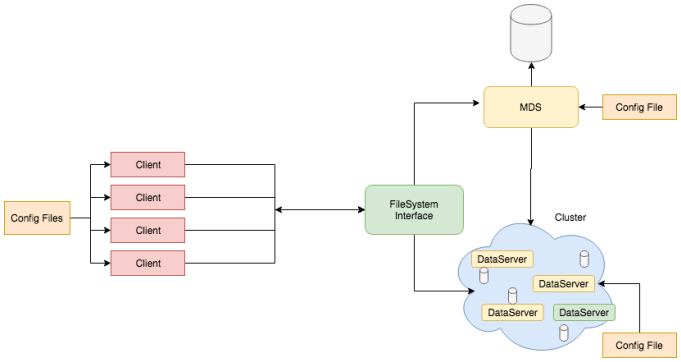


Figure 4: Design of Distributed SSD Simulator

Following from the above, we can conclude that the use of *strong consistency protocols* promotes *longer lifetimes*. This forms the core basis of the ***CLAP Theorem***.

4 Empirical Results

4.1 Methodology

We used a simulator to model a bunch of workloads atop two different distributed file systems - GFS [12] and Ceph [48]. In our implementation, GFS represents a distributed file system that supports a stronger form of consistency while Ceph represents a distributed file system that supports a weaker form of consistency.

Figure 4 specifies the high-level design of the simulator. At the highest layer, the user can specify the workload to be run. The workload interacts with the distributed file system APIs (Algorithm 1). The distributed file system can be configured to use either GFS or Ceph. Additional settings like the *number of replicas* and *chunk size* are also configurable. For the purpose of this prototype, we assume that all read and write requests will have a size that is a multiple of the chunk size. Additionally, we also assume that the chunk size is a multiple of the block size.

The distributed file system in turn interacts with the dataservers which are also configuration-based. The dataservers manage the allocation of data across SSD blocks and handle SSD internals such as *Garbage Collection*, *Wear-Leveling*, and *Data Scrubbing*. We parameterize the SSD as a set of configurations so that we can model SSDs from different manufacturers. Below is a list of configuration options available for each data server -

- **PAGE SIZE.** Defines size of SSD page in bytes.
- **PAGES PER BLOCK.** Defines the number of pages per block in the SSD.
- **TOTAL NUMBER OF PAGES.** Defines total number of pages available in the SSD. The storage capacity of the

Pseudocode 1 APIs supported by the simulator

```
1: ▷ FILENAMETOFD = Mapping from file name to FD
2: ▷ FDTOCHUNKID = Mapping from FD to list of chunk IDs
3: ▷ CHUNKIDTOSERVERS = Mapping from chunk ID to list of data servers
4: ▷ FDTOFILEATTRIBUTES = Mappping from FD to list of File Attributes
5: ▷ CLIENTIDFILETOOFFSET = Per client mapping from FD to offset
6:
7: procedure CREATE(Filename F, ClientID C)
8:     ▷ Insert F into FILENAMETOFD
9:     ▷ Create a new file attribute for F and insert into FDTOFILEATTRIBUTES
10:    ▷ Insert offset of 0 for file F and client C into CLIENTIDFILETOOFFSET
11:    ▷ return FD
12: end procedure
13:
14: procedure OPEN(Filename F, ClientID C)
15:    ▷ Get FD for F from FILENAMETOFD
16:    ▷ Insert offset of 0 for file F and client C into CLIENTIDFILETOOFFSET
17:    ▷ return FD
18: end procedure
19:
20: procedure READ(FileDescriptor FD, Buffer B, ReadCount RC, ClientID C)
21:    ▷ Get offset for client C and file descriptor FD from CLIENTIDFILETOOFF-
    SET
22:    ▷ STARTINGCHUNKNO = offset/ChunkSize
23:    ▷ CHUNKSTOREAD = RC/ChunkSize
24:    for all chunk ∈ list of chunks to read do
25:        ▷ Get list of dataservers from CHUNKIDTOSERVERS
26:        ▷ Decision of which dataserver to read from is implementation specific
27:    end for
28:    ▷ Update offset for FD and C in CLIENTIDFILETOOFFSET
29:    ▷ Return the contents read
30: end procedure
31: procedure WRITE(FileDescriptor FD, Buffer B, WriteCount WC, ClientID C)
32:    ▷ Get offset for client C and file descriptor FD from CLIENTIDFILETOOFF-
    SET
33:    ▷ STARTINGCHUNKNO = offset/ChunkSize
34:    ▷ CHUNKSTOWRITE = WC/ChunkSize
35:    for all chunk ∈ list of chunks to write do
36:        if chunk to be overwritten then
37:            ▷ DATASERVERS = get dataservers for chunk from CHUNKID-
    TOSERVERS
38:            ▷ Issue writes to servers to update all replicas.
39:        else if a new chunk needs to be created
40:            ▷ DATASERVERS = Identify dataservers to which new chunk should
    be written to. This is implementation specific
41:            ▷ Insert DATASERVERS for chunk into CHUNKIDTOSERVERS
42:        end if
43:        ▷ Update last modified attribute for FD in FDTOFILEATTRIBUTES
44:        ▷ Update offset for FD and C in CLIENTIDFILETOOFFSET
45:    end for
46:    ▷ return amount of data written
47: end procedure
48:
49: procedure SEEK(FileDescriptor FD, Offset O, ClientID C)
50:    ▷ Update offset for FD and C in CLIENTIDFILETOOFFSET to O
51: end procedure
52:
53: procedure DELETE(FileDescriptor FD)
54:    ▷ Remove mappings from all datastructures.
55:    ▷ Delete corresponding chunks from dataservers.
56: end procedure
57:
```

SSD can be computed as the product of PAGE SIZE and TOTAL NUMBER OF PAGES.

- **MAX ERASE COUNT.** Defines the maximum number of erase operations that every page within that SSD can endure. In reality, this number would vary from block to block due to manufacturing variations. However, for the purpose of this prototype, we assume that all pages have similar endurance.

- **GC THRESHOLD.** When the fraction of *free pages* within the SSD drops below the GC THRESHOLD, the SSD triggers *garbage collection*. Since we do not model performance, we check if GC needs to be triggered each time there is a change in block allocation. This works well enough to model lifetime for the sake of this prototype.

- **MAX READ RETRIES.** Define the max retries that the FTL will perform under read failures. If the number of retries reaches MAX READ RETRIES, then the read will result in a failure.

- **DATA SCRUBBING THRESHOLD.** When the fraction of number of retries used for serving a particular read request against MAX READ RETRIES exceeds the DATA SCRUBBING THRESHOLD, then the block read will be marked for migration.

We simulate *wear-leveling* by choosing the block that has incurred the least amount of writes while choosing new blocks to write to.

In an effort to promote research in the correlation between distributed file systems and SSD cluster lifetime, we have made our simulator available at https://github.com/Arjunbala/DistributedSystemsSSDs.

4.2 Experimental Setup

4.2.1 Cluster Configuration

In our base configuration, we set the number of replicas as 3 and the chunk size as 4MB. For data servers, we use a page size of 128KB, 4 pages per block, with a total disk capacity of 100MB. We set the GC threshold as 0.8, the data scrubbing threshold as 0.1, and the max read retries as 20. The max erase count is set to 1000. We intentionally scale down the disk capacity and the max erase count so as to get faster results. The trends we observe will hold true even when scaled up since the trend extrapolates.

4.2.2 Workload

We choose three workloads that result in different I/O patterns as below

- **Downpour Stochastic Gradient Descent.** Downpour Stochastic Gradient Descent (Downpour SGD) is a popular distributed machine learning training algorithm proposed in DistBelief [6]. Though the world of distributed ML training has evolved since DistBelief, the core idea of using Downpour SGD still pervades. Downpour SGD employs data parallelism - the training data is split into multiple disjoint parts among *multiple workers*. Each worker samples random training points from its split of data and performs gradient descent over them. The updates to parameters are then sent over the network to

a single master. The number of points that each worker samples is determined by a quantity known as *mini-batch size*. In terms of the I/O workload, Downpour SGD is a *read-heavy* workload that generates random I/O since each worker randomly picks training data points from its split of training data. In our setting, we assume that the entire training data is stored by the distributed file system and that individual workers fetch data by reading from the distributed file system.

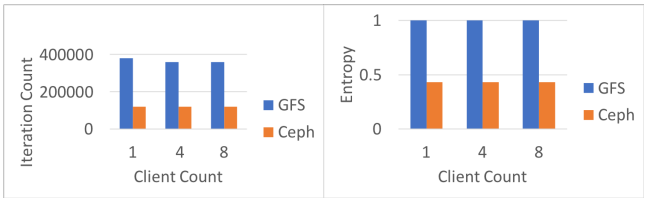
- **Ephemeral Data Workload.** Traditional compute-centric frameworks [7, 50] process batches of data in multiple stages often represented by a DAG abstraction. Each stage read input data, performs some compute, and writes some *intermediate data* to the disk. Post this, the next stage reads data from disk, once again performs some computation, and once again writes out *intermediate data* to the disk. Pocket [26] is a system that provides cheap, scalable storage for this intermediate data, which is also termed as *ephemeral data*. It is termed as *ephemeral data* because the data is no longer needed once downstream stages have consumed them. We model an *ephemeral data* workload by modeling it as a variant of iterative *MapReduce* [7] - First, the mapper reads data from disk, then writes out intermediate data to disk, which is then read by a *variable number of reducers*. The reducers once again write out data to disk which is then read by the mappers in the next iteration. The *ephemeral data* written out by mappers and reducers can be deleted when they have been consumed by their downstream stage. Thus, *emphemeral data* presents a workload which involves a high number of writes, reads, and deletions.

- **Hot and Cold Workload.** [40] presents an observation that users operate for a large fraction of their time on a small subset of files. We refer to the frequently accessed data as *hot data* and the less frequently accessed data as *cold data*. We define *skewness* as the percentage of total data that is *hot data*. For instance, a *skewness* of 10% implies that 10% of the data is accessed frequently while 90% of the data is accessed infrequently. We perform a mixture of reads and writes on both *hot* and *cold* data in our workload.

4.3 Metrics

We use two metrics to evaluate the lifetime of a distributed file system -

- **Iteration Count.** We run each of the workloads in iterations until it hits a point of *failure*. We define *failure* to have occured in one of the following scenarios occurs - (i) A read fails as the read retry for a page exceeds MAX READ RETRIES. (ii) The number of *erase operations* for a particular page on the SSD exceeds MAX ERASE COUNT for any SSD. In our failure model, we have a strict notion for *failure* as the event when a single page in any SSD crosses its endurance limit. In all of our experiments, we measure the *number of iterations* endured by the SSD cluster till *failure* occurs. A *higher* iteration count indicates better lifetime characteristics.



(a) Iteration Count (b) Entropy

Figure 5: Downpour SGD - Trend by modifying client count (number of workers)

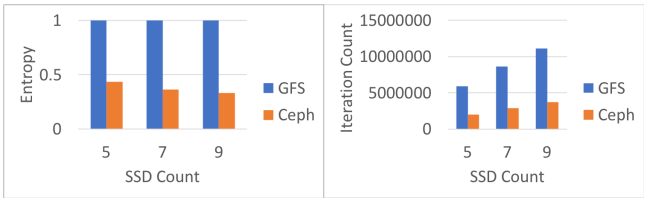
- **Entropy.** Another metric of interest is the *distribution of wear* across the cluster when failure happens. For this purpose, we use *Jain's Fairness Index* to compute the entropy of wears across the cluster. For each SSD, we find the page that has endured the maximum number of erase operations as $E_1, E_2..E_n$. We then compute the Jain's Fairness Index over the set of $E_1, E_2, ..E_n$. A value closer to 1 indicates a fair spread of wear across the cluster while lower values indicate an uneven spread of wear across the SSD cluster.

4.4 Lessons

4.4.1 Downpour SGD Workload

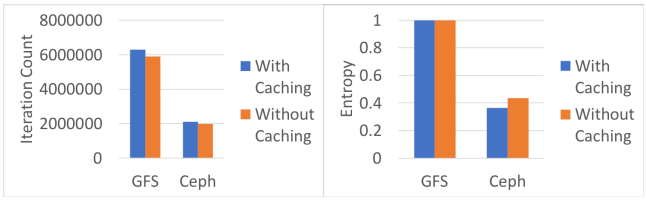
For this workload, we use training data of size 100MB loaded onto the distributed file system. Each training data sample has a size of 512KB. We vary the batch size and worker count in our experiments.

- **Lesson 1: *Distribute reads across replicas for better lifetime.*** Figure 5 presents the variation in lifetime and entropy in wear as the number of workers(clients) vary in the Downpour SGD workload. We observe that GFS outlasts Ceph irrespective of the client count. This can be attributed to the fact that GFS has a stronger consistency model. In GFS, a read can be done from any replica which in turn ensures a uniform distribution of reads across replicas. As a result, the wear caused by *data scrubbing* gets uniformly distributed across the cluster. In contrast, Ceph redirects all read requests to the primary replica. This results in a few SSDs incurring wear due to *data scrubbing*, while leaving out the others. As a result, we observe shorter lifetimes and worse entropy values.



(a) Iteration Count (b) Entropy

Figure 6: Downpour SGD - Trend by modifying the number of SSDs in the cluster.



(a) Iteration Count (b) Entropy

Figure 7: Downpour SGD - Trend by enabling/disabling caching

- **Lesson 2:** ***Distributing reads leads to better lifetimes as cluster scales.*** Figure 6 presents the variation in lifetime and entropy in wear as we vary the size of the cluster for the Downpour SGD workload. We observe that Ceph progressively gets worse in comparison to GFS as we increase the cluster size. This can be once again attributed to the weaker form of consistency in Ceph. In Ceph, even though the cluster size increases, the reads would continued to be directed only towards those SSDs that hold the replicas. Consequently, those SSDs would suffer greater wear in comparison to the rest, resulting in decreased lifetimes.

- **Lesson 3:** ***Caching does not solve all problems in read heavy workloads.*** Figure 7 captures the effect of caching on the lifetime and entropy of wear across a cluster of SSDs. In this experiment, we use a cache size that is 10% the size of the training data. We notice that caching does improve the lifetime of the cluster for both GFS and Ceph. The benefits would be better with larger cache sizes since the cache would be able to absorb a larger number of read requests. In spite of the absorbing a proportion of reads, we notice that Ceph still performs an order of magnitude worse than GFS. This outlines the fact that stronger consistency guarantees lead to better lifetimes even in the presence of significantly large caches.

- **Lesson 4:** ***Workload characteristics such as batch size do not matter much.*** Figure 8 shows the variance in lifetime and wear entropy as we vary the *mini-batch size* for the Downpour SGD workload. We notice that the number of iterations decreases as the mini-batch size increases. This is expected as larger mini-batches issue more reads in a single iteration. It is interesting to note that the lifetimes of both GFS and Ceph decrease by the same order of magnitude as the mini-batch size increases. This indicates that consistency policies do not affect wearing properties when workload characteristics such as mini-batch size are modified.

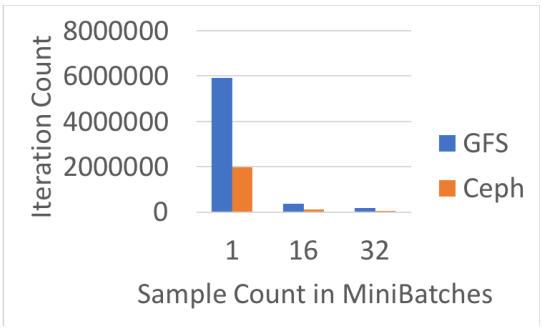
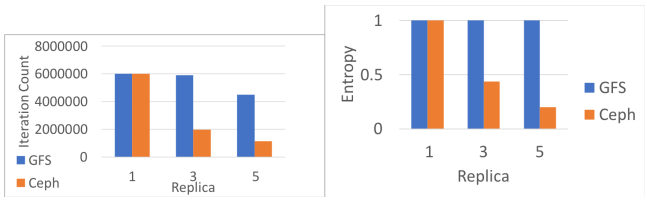


Figure 8: Downpour SGD - Trend of iteration count on varying mini-batch size



(a) Iteration Count (b) Entropy

Figure 9: Downpour SGD - Trend by varying the number of replicas in the cluster

- **Lesson 5:** ***Distributing reads ensures fairness of wear with a higher number of replicas.*** Figure 9 shows the variation in lifetime and entropy of wear as we vary the number of replicas employed by the distributed file system (GFS and Ceph). We notice that Ceph increases performs worse as we increase the number of replicas. For distributed storage systems like GFS that adopt stronger consistency policies, a larger number of replicas promotes a wider spread of reads, whereas the reads would continue to go to the same *primary* replica in Ceph. Hence, Ceph does not benefit in comparison to GFS with an increasing number of replicas.

4.4.2 Ephemeral Data Workload

In this workload, we assume that each mapper writes out 2KB of data and that each reducer again writes out 2KB of data. We vary the number of reducers in our experiments.

- **Lesson 6:** ***Workload characteristics do not matter for***

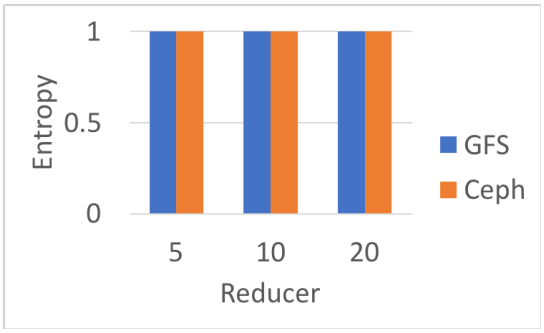
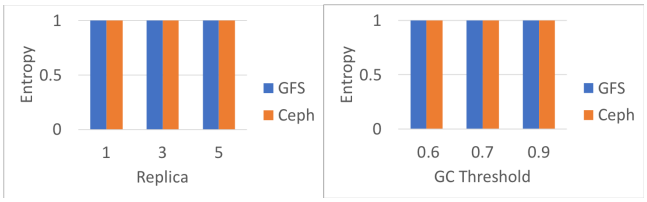


Figure 10: Ephemeral data - Trend of entropy on varying the number of reducers



(a) Iteration Count (b) Entropy

Figure 11: Ephemeral Data - Trend as distributed file system characteristics (Number of replicas) and SSD characteristics (GC Threshold) vary

ephemeral data. Figure 10 captures the trend as we vary the number of reducers for ephemeral data workloads. We observe these workload characteristics do not impact lifetime for ephemeral data workloads. This is because deletes dominate the cause of wear and the amount of data deletion is same for both Ceph and GFS.

- **Lesson 7:** ***File System and SSD characteristics do not affect ephemeral workloads.*** Figure 11 shows the effects on lifetime as we vary distributed system characteristics like the number of replicas and SSD characteristics like the GC THRESHOLD. In deletion heavy workloads like ephemeral data, the iteration count would not vary much with the number of replicas because all replicas would need to be destroyed irrespective of the consistency protocol. Similarly, when a lot of data is written and deleted, GC will invariably always kick in irrespective of the threshold, leading to similar wearing characteristics with varying GC THRESHOLDS.

4.4.3 Hot and Cold Workload

For this workload, we assume that each file has a size of 2MB. The proportion of hot and cold files is adjusted by the *skewness factor*.

- **Lesson 8:** ***Distributing the workload provides better lifetimes with a higher amount of data*** Figure 12 shows the variation of iteration count as we vary the file count. We observe that GFS gradually improves over Ceph as the number of files increase. This once again boils down to the fact that GFS distributes read among the replicas courtesy of its stronger consistency protocol. The effect as expected gets more magnified as the number of files (and consequently the amount of data stored) increases.

Figure 12: Hot and Cold Workload - Trend of iteration count as number of files changes

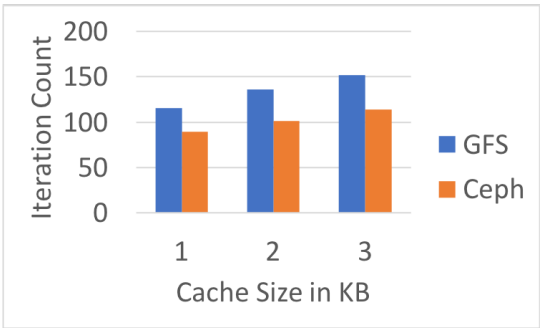


Figure 13: Hot and Cold Workload - Trend of iteration count as cache size increases

- **Lesson 9:** ***Increasing cache size does not proportionally increase lifetime*** Figure 13 shows the impact of varying cache sizes on the iteration count and SSD cluster lifetime. We observe that inspite of additional caches does decrease the order of magnitude by which GFS outperforms Ceph. However, the increase is not linear in nature. Thus, increasing cache size does not linearly decrease the difference in lifetimes between stronger and weaker consistency protocols.

- **Lesson 10:** ***Skewed workloads affect the system lifetime*** Figure 14 shows the impact on lifetime as we vary the *skewness*. We observe that the factor by which GFS outperforms Ceph increases as we the percentage of hot data increases. This can be attributed that as more data gets touched(reads/writes), the effects of splitting the workload among the different replicas in protocols having stronger consistency semantics helps.

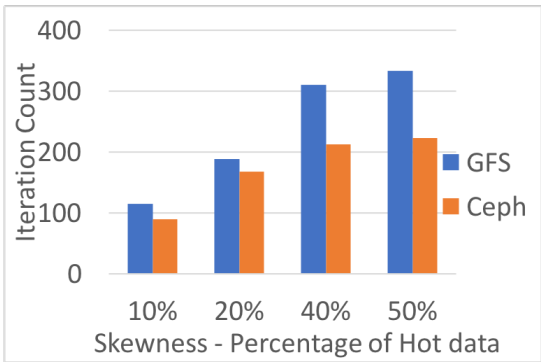


Figure 14: Hot and Cold Workload - Trend of iteration count as the skewness *of hot data is varied*

5 Related Work

5.1 SSD Performance and Reliability

Several researchers have looked at the properties of SSDs and performed studies to understand workloads and design choices suitable for single SSDs. However, no prior work has looked at the performance and lifetime of a cluster of SSDs.

He et. al [17] performed a detailed analysis of applications atop modern file systems and FTLs and formalized an *unwritten contract* that clients of SSDs must follow to achieve high performance. They present five rules that are critical for SSDs - (i) *Request Scale* : SSD clients must issue large requests or many requests to fully utilize the parallel bandwidth offered by SSDs. (ii) *Locality* : To avoid SSD cache translation misses, clients must issue requests to the SSD with locality (iii) *Aligned Sequentiality* : To reduce the cost of converting page-level mappings to block-level mappings in hybrid-mapping FTLs, clients must write with sequentiality within a block. (iv) *Group by Death Time* : Clients must group writes by death time to reduce cost of garbage collection [33] (v) *Uniform Data Lifetime* : Clients must create data with similar lifetimes to reduce overheads of wear leveling [3].

Kim et. al [23] looked at the design trade-off for reliability within an SSD. They outline three major sources of flash memory errors as - (i) *Wear*: Repeated P/E cycles gradually wear out flash memory cells [4, 15]. (ii) *Retention Loss*: Electrons in flash memory leak over time and the errors caused by retention increases with the amount of wear. It is also important to note that retention errors disappear once the block has been erased [34]. (iii) *Disturbance* : Studies have shown reading a wordline in a block weakly affects other wordlines in a block and gradually causes errors with re-reads [4, 15, 34]. Like with retention, this error too disappears once the block has been erased. They further discuss the trade-offs that arise in terms of reliability and performance due to operations like error correction, intra-SSD redundancy [25], and data scrubbing [16, 29].

5.2 Distributed Storage Reliability

Several studies have been carried out to characterize the reliability of disks and disk failures [9, 18, 21, 30, 38, 39, 41] and have come up with schemes like redundancy and erasure codes [2, 5, 19, 28] to prevent data loss due to disk failures. However, no work has comprehensively looked at or modeled wear across a cluster of SSDs.

Other work has looked at how to use measures of reliability to configure redundancy settings. The most recent work in this area is by *Kadekodi et. al* [22], who analyze large-scale storage systems having a heterogenous mix of storage devices with significantly different failures rates. They make the observation that redundancy settings are statically configured and explore opportunities to reduce the amount of redundancy during periods where disks have high reliability. Using the information that annualized failure rates follow a *bathtub curve* [10, 49], they design a system *HeART* that identifies periods of infancy, useful lifetime, and wearout in order to configure redundancy settings in an online fashion.

5.3 Tiering File Systems

A popular solution to effective utilize and maximize the benefits and lifetime of SSDs has been to use them in combination with other persistent media.

Griffin [44] is a hybrid storage device with HDD as a write cache for a SSD based storage systems. The main motivations behind this approach are : *(i)* HDDs can match the sequential write bandwidth of SSD. *(ii)* General purpose workloads contain a fraction of batch overwrite. By ensuring those writes on HDD in a log structured format, one can reduce the write amplification on SSD.

Ziggurat [51] looks at how NVMM can complement SSDs to create storage systems with near-NVMM performance and large capacity. The paper presents a synchronicity predictor to decide which storage tier to redirect write requests to. The paper also looks at the distinction between hot data and cold data and intelligently places only cold data in SSDs in order to reduce write amplification.

6 Conclusion

In this work, we have explored the impact of different consistency protocol strategies on the lifetime of an SSD cluster. Through experiments and logical arguments, we have shown that *stronger* consistency protocols can promote *longevity* in SSD clusters. This forms the basis for the newly proposed *CLAP theorem* that can cleanly encapsulate the design trade-offs for distributed system design on top of modern storage media.

References

[1] ARMBRUST, M., XIN, R. S., LIAN, C., HUAI, Y., LIU, D., BRADLEY, J. K., MENG, X., KAFTAN, T., FRANKLIN, M. J., GHODSI, A., AND ZAHARIA, M. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, ACM, pp. 1383–1394.

[2] BLÖMER, J., KALFANE, M., KARP, R., KARPINSKI, M., LUBY, M., AND ZUCKERMAN, D. An xor-based erasure-resilient coding scheme, 1995.

[3] BOBOILA, S., AND DESNOYERS, P. Write endurance in flash drives: Measurements and analysis. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2010), FAST'10, USENIX Association, pp. 9–9.

[4] CAI, Y., GHOSE, S., HARATSCH, E. F., LUO, Y., AND MUTLU, O. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proceedings of the IEEE 105*, 9 (Sep. 2017), 1666–1704.

[5] CORBETT, P., ENGLISH, B., GOEL, A., GRCANAC, T., KLEIMAN, S., LEONG, J., AND SANKAR, S. Row-diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2004), FAST'04, USENIX Association, pp. 1–1.

[6] DEAN, J., CORRADO, G. S., MONGA, R., CHEN, K., DEVIN, M., LE, Q. V., MAO, M. Z., RANZATO, M., SENIOR, A., TUCKER, P., YANG, K., AND NG, A. Y. Large scale distributed deep networks. In *NIPS* (2012).

[7] DEAN, J., AND GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation* (San Francisco, CA, 2004), pp. 137–150.

[8] DECANDIA, G., HASTORUN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., AND VOGELS, W. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles* (New York, NY, USA, 2007), SOSP '07, ACM, pp. 205–220.

[9] ELERATH, J. Hard-disk drives: The good, the bad, and the ugly. *Commun. ACM 52*, 6 (June 2009), 38–45.

[10] ELERATH, J. G. Afr: problems of definition, calculation and measurement in a commercial environment. In *Annual Reliability and Maintainability Symposium. 2000 Proceedings. International Symposium on Product Quality and Integrity (Cat. No.00CH37055)* (Jan 2000), pp. 71–76.

[11] ESPEHOLT, L., SOYER, H., MUNOS, R., SIMONYAN, K., MNIH, V., WARD, T., DORON, Y., FIROIU, V., HARLEY, T., DUNNING, I., LEGG, S., AND KAVUKCUOGLU, K. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning* (Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018), J. Dy and A. Krause, Eds., vol. 80 of *Proceedings of Machine Learning Research*, PMLR, pp. 1407–1416.

[12] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2003), SOSP '03, ACM, pp. 29–43.

[13] GILBERT, S., AND LYNCH, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News 33*, 2 (June 2002), 51–59.

[14] GONZALEZ, J. E., XIN, R. S., DAVE, A., CRANKSHAW, D., FRANKLIN, M. J., AND STOICA, I. Graphx: Graph processing in a distributed dataflow framework. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2014), OSDI'14, USENIX Association, pp. 599–613.

[15] GRUPP, L. M., CAULFIELD, A. M., COBURN, J., SWANSON, S., YAAKOBI, E., SIEGEL, P. H., AND WOLF, J. K. Characterizing flash memory: Anomalies, observations, and applications. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Dec 2009), pp. 24–33.

[16] HA, K., JEONG, J., AND KIM, J. An integrated approach for managing read disturbs in high-density nand flash memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35*, 7 (July 2016), 1079–1091.

[17] HE, J., KANNAN, S., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. The unwritten contract of solid state drives. In *Proceedings of the Twelfth European Conference on Computer Systems* (New York, NY, USA, 2017), EuroSys '17, ACM, pp. 127–144.

[18] HEIEN, E., LAPINE, D., KONDO, D., KRAMER, B., GAINARU, A., AND CAPPELLO, F. Modeling and tolerating heterogeneous failures in large parallel systems. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov 2011), pp. 1–11.

[19] HUANG, C., AND XU, L. Star : An efficient coding scheme for correcting triple storage node failures. *IEEE Transactions on Computers 57*, 7 (July 2008), 889–901.

[20] HUNT, P., KONAR, M., JUNQUEIRA, F. P., AND REED, B. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2010), USENIXATC'10, USENIX Association, pp. 11–11.

[21] JIANG, W., HU, C., ZHOU, Y., AND KANEVSKY, A. Are disks the dominant contributor for storage failures?: A comprehensive study of storage subsystem failure characteristics. *Trans. Storage 4*, 3 (Nov. 2008), 7:1–7:25.

[22] KADEKODI, S., RASHMI, K. V., AND GANGER, G. R. Cluster storage systems gotta have heart: improving storage efficiency by exploiting disk-reliability heterogeneity. In *17th USENIX Conference on File and Storage Technologies (FAST 19)* (Boston, MA, 2019), USENIX Association, pp. 345–358.

[23] KIM, B. S., CHOI, J., AND MIN, S. L. Design tradeoffs for SSD reliability. In *17th USENIX Conference on File and Storage Technologies (FAST 19)* (Boston, MA, 2019), USENIX Association, pp. 281–294.

[24] KIM, H.-J., LEE, Y.-S., AND KIM, J.-S. Nvmedirect: A user-space i/o framework for application-specific optimization on nvme ssds. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)* (Denver, CO, 2016), USENIX Association.

[25] KIM, J., LEE, E., CHOI, J., LEE, D., AND NOH, S. H. Chip-level raid with flexible stripe size and parity placement for enhanced ssd reliability. *IEEE Transactions on Computers 65*, 4 (April 2016), 1116–1130.

[26] KLIMOVIC, A., WANG, Y., STUEDI, P., TRIVEDI, A., PFEFFERLE, J., AND KOZYRAKIS, C. Pocket: Elastic ephemeral storage for serverless analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (Carlsbad, CA, 2018), USENIX Association, pp. 427–444.

[27] LU, L., PILLAI, T. S., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Wiskey: Separating keys from values in ssd-conscious storage. In *14th USENIX Conference on File and Storage Technologies (FAST 16)* (Santa Clara, CA, 2016), USENIX Association, pp. 133–148.

[28] LUO, J., SHRESTHA, M., XU, L., AND PLANK, J. S. Efficient encoding schedules for xor-based erasure codes. *IEEE Transactions on Computers 63*, 9 (Sep. 2014), 2259–2272.

[29] LUO, Y., CAI, Y., GHOSE, S., CHOI, J., AND MUTLU, O. Warm: Improving nand flash memory lifetime with write-hotness aware retention management. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)* (May 2015), pp. 1–14.

[30] MA, A., DOUGLIS, F., LU, G., SAWYER, D., CHANDRA, S., AND HSU, W. Raidshield: Characterizing, monitoring, and proactively protecting against disk failures. In *13th USENIX Conference on File and Storage Technologies (FAST 15)* (Santa Clara, CA, 2015), USENIX Association, pp. 241–256.

[31] MALEWICZ, G., AUSTERN, M. H., BIK, A. J., DEHNERT, J. C., HORN, I., LEISER, N., AND CZAJKOWSKI, G. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 international conference on Management of data* (New York, NY, USA, 2010), pp. 135–146.

[32] MENG, X., BRADLEY, J., YAVUZ, B., SPARKS, E., VENKATARAMAN, S., LIU, D., FREEMAN, J., TSAI, D., AMDE, M., OWEN, S., XIN, D., XIN, R., FRANKLIN, M. J., ZADEH, R., ZAHARIA, M., AND TALWALKAR, A. Mllib: Machine learning in apache spark. *J. Mach. Learn. Res. 17*, 1 (Jan. 2016), 1235–1241.

[33] MEZA, J., WU, Q., KUMAR, S., AND MUTLU, O. A large-scale study of flash memory failures in the field. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA, 2015), SIGMETRICS '15, ACM, pp. 177–190.

[34] MIELKE, N. R., FRICKEY, R. E., KALASTIRSKY, I., QUAN, M., USTINOV, D., AND VASUDEVAN, V. J. Reliability of solid-state drives based on nand flash memory. *Proceedings of the IEEE 105*, 9 (Sep. 2017), 1725–1750.

[35] MURRAY, D. G., MCSHERRY, F., ISAACS, R., ISARD, M., BARHAM, P., AND ABADI, M. Naiad: A timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 439–455.

[36] NIGHTINGALE, E. B., ELSON, J., FAN, J., HOFMANN, O., HOWELL, J., AND SUZUE, Y. Flat datacenter storage. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)* (Hollywood, CA, 2012), USENIX, pp. 1–15.

[37] NOGHABI, S. A., PARAMASIVAM, K., PAN, Y., RAMESH, N., BRINGHURST, J., GUPTA, I., AND CAMPBELL, R. H. Samza: Stateful scalable stream processing at linkedin. *Proc. VLDB Endow. 10*, 12 (Aug. 2017), 1634–1645.

[38] PATTERSON, D. A., GIBSON, G., AND KATZ, R. H. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 1988), SIGMOD '88, ACM, pp. 109–116.

[39] PINHEIRO, E., WEBER, W.-D., AND BARROSO, L. A. Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2007), FAST '07, USENIX Association, pp. 2–2.

[40] ROSELLI, D. S., LORCH, J. R., ANDERSON, T. E., ET AL. A comparison of file system workloads. In *USENIX annual technical conference, general track* (2000), pp. 41–54.

[41] SCHROEDER, B., AND GIBSON, G. A. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2007), FAST '07, USENIX Association.

[42] SHUE, D., FREEDMAN, M. J., AND SHAIKH, A. 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)* (Hollywood, CA, 2012), USENIX, pp. 349–362.

[43] SHVACHKO, K., KUANG, H., RADIA, S., AND CHANSLER, R. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (Washington, DC, USA, 2010), MSST '10, IEEE Computer Society, pp. 1–10.

[44] SOUNDARARAJAN, G., PRABHAKARAN, V., BALAKRISHNAN, M., AND WOBBER, T. Extending ssd lifetimes with disk-based write caches. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2010), FAST'10, USENIX Association, pp. 8–8.

[45] THUSOO, A., SARMA, J. S., JAIN, N., SHAO, Z., CHAKKA, P., ANTHONY, S., LIU, H., WYCKOFF, P., AND MURTHY, R. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow. 2*, 2 (Aug. 2009), 1626–1629.

[46] TOSHNIWAL, A., TANEJA, S., SHUKLA, A., RAMASAMY, K., PATEL, J. M., KULKARNI, S., JACKSON, J., GADE, K., FU, M., DONHAM, J., BHAGAT, N., MITTAL, S., AND RYABOY, D. Storm@twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2014), SIGMOD '14, ACM, pp. 147–156.

[47] VAN RENESSE, R., AND SCHNEIDER, F. B. Chain replication for supporting high throughput and availability. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6* (Berkeley, CA, USA, 2004), OSDI'04, USENIX Association, pp. 7–7.

[48] WEIL, S. A., BRANDT, S. A., MILLER, E. L., LONG, D. D. E., AND MALTZAHN, C. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2006), OSDI '06, USENIX Association, pp. 307–320.

[49] YANG, J., AND FENG-BIN SUN. A comprehensive review of hard-disk drive reliability. In *Annual Reliability and Maintainability. Symposium. 1999 Proceedings (Cat. No.99CH36283)* (Jan 1999), pp. 403–409.

[50] ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULY, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)* (San Jose, CA, 2012), USENIX, pp. 15–28.

[51] ZHENG, S., HOSEINZADEH, M., AND SWANSON, S. Ziggurat: A tiered file system for non-volatile main memories and disks. In *17th USENIX Conference on File and Storage Technologies (FAST 19)* (Boston, MA, 2019), USENIX Association, pp. 207–219.