SEA: SSD Staged Energy Efficient Object Storage System Architecture

Sai Vishwas Padigi, Sameer Kulkarni, Phalachandra HL, Dinkar Sitaram

***Abstract:** The advent of social media, smart mobile devices and the Internet of Things (IoT) has led to the generation of unstructured data at an astronomical rate, thereby creating an ever-increasing demand for object storage. These object storage systems consume a lot of energy, resulting in increased heat dissipation, greater cooling requirements (which in turn consumes more energy), higher operational costs, and excessive carbon footprint. Although there has been some progress in building energy-efficient disk systems, works on energy-efficient object storage systems are still in the nascent stage.*

In this paper, we propose SEA: An SSD Staged Energy Efficient Object Storage System Architecture, wherein we introduce a staging layer comprising Solid State Drives (SSDs) on top of the existing object storage system consisting primarily of Hard Disk Drives (HDDs). SSDs not only consume lesser power as compared to HDDs but are also much faster. Leveraging SSDs for staging reduces the number and frequency of requests hitting the object storage system underneath, allowing us to selectively spin down a substantial number of disks without violating any Service Level Agreements driven by Quality of Service requirements while reducing the total disk energy consumption. Given the high-performance characteristics of SSDs, this SSD staging layer significantly enhances the performance of the object storage system as a whole. As a case study, we have modeled this architecture for OpenStack Swift. Our simulation results using a Dropbox-like workload show that, even after factoring in the additional energy consumed by the SSD staging layer, our model was able to reduce the total disk energy consumption by up to 15.235 % and improve performance by up to 29.06 %.

***Keywords:** Energy efficiency, object storage, performance enhancement, SSD staging*

I. INTRODUCTION

Increased proliferation of social media, smart mobile devices, and the Internet of Things (IoT) has led to the generation and the need for storing enormous amounts of unstructured data [1]. This large unstructured data typically gets stored as objects in object storage systems [2].

Revised Manuscript Received on May 30, 2020.
***** Correspondence Author
Sai Vishwas Padigi, Computer Science department, PES University, Bangalore, India. Email: saivishwasp@gmail.com
Sameer Kulkarni, Computer Science department, PES University, Bangalore, India. Email: contact@sameerkulkarni.me
Phalachandra HL*, Computer Science department, PES University, Bangalore, India. Email: phalachandra@pes.edu
Dinkar Sitaram, Computer Science department, PES University, Bangalore, India. Email: dinkars@pes.edu

Objects in object storage systems are abstractions of the logical collection of data bytes. Unlike blocks, objects could be of variable sizes and are capable of storing entire data structures, files, or multimedia [3].

These object storage systems are capable of providing reliable, scalable storage for enormous amounts of data at relatively low costs due to their ability to distribute the objects across a large number of storage servers [4].

These object storage systems are hosted on the cloud or in large on-premise data centers and consume a lot of energy. The high amount of energy consumed by a large number of these storage devices causes high heat dissipation, necessitating greater cooling requirements, which in turn leads to higher energy consumption in the data centers [5].

The high energy consumption also results in higher energy costs of data centers, which is increasing by 25% annually [6]. The high energy consumption also leads to another pressing issue of a large carbon footprint being left behind. In 2005, data centers alone were responsible for 2% of the carbon dioxide emissions worldwide, and it was estimated to increase by 6% every year [7]. Given that storage devices account for almost 27% of the total energy consumed by a data center [8], it is of utmost importance that we build energy-efficient storage systems.

Most legacy data centers typically have Hard Disk Drive (HDD) based storage devices [9], which contribute significantly to the total energy consumption of the data center. HDDs do not exhibit energy proportional characteristics, wherein the energy consumed is to be proportional to the I/O operations/sec. Hence, HDDs consume around 85% of energy even when idle [7]. On the other hand, Solid State Drives (SSDs), which are a more recent technology, exhibit these energy proportional characteristics and hence consume significantly lesser power as compared to HDDs.

As against HDDs, the absence of a mechanical arm and rotating platters enables the SSDs to support much higher read and write data transfer rates [7]. Hence, SSDs are getting increasingly popular due to these high-performance and low energy consumption characteristics as discussed above.

However, it is still not feasible to replace all of the HDDs by SSDs due to factors related to cost. The difference between the cost per gigabyte ($/GB) of SSDs and that of HDDs has been shrinking, but it is still large enough to prevent data center operators from completely switching over to SSDs [7].

We also see the adoption rate of SSDs consistently increasing due to their high-performance characteristics, with studies indicating that SSDs, which accounted for 8% of the non-tape storage in 2012, would constitute 47% of the installed drive base in 2020 [9].

Although it may not be feasible to switch over to SSDs completely, it is still practical and feasible for most data centers to have a small percentage of the total disks to be SSDs to achieve a significant improvement in performance.

To improve the energy efficiency and the performance of the object storage system, we present SEA: An SSD Staged Energy Efficient Object Storage System Architecture, which efficiently exploits the high-performance and low power consumption characteristics of these small percentage of SSDs rather than treating them as just a replacement to HDDs. SEA leverages SSDs for staging the objects which are most likely to be requested for retrieval soon, thereby drastically reducing the number of requests needed to be served by the object storage system comprising HDDs.

The reduction in the amount of traffic to the object storage system layer comprising the HDDs allows for the spinning down of a significant set of HDDs in the object storage system, without impacting the performance. In this architecture, the efficiency of the staging layer is the dominant factor in determining the extent of performance enhancement. Similarly, the number of HDDs in the object storage layer that could be spun down has a significant influence on the amount of energy that can be conserved. SEA aims to achieve significant energy savings and improve the performance extensively.

In this paper, as a case study, we have modeled the architecture for OpenStack Swift [4], which is open-source software designed to manage the storage of large amounts of data cost-effectively on a long-term basis across clusters of standard server hardware. However, this architecture could be leveraged by any of the other object storage systems including Amazon S3 [10] and Google Cloud Storage [11].

II. A CASE STUDY WITH OPENSTACK SWIFT

A. Basic Swift Terminology

In Swift, each physical hardware/machine running Swift processes is referred to as a storage node. Each node can contain multiple disks. Each of these disks can further contain multiple partitions.

The internal structure of a storage node has been illustrated in Figure 1. A partition can essentially be perceived as a directory located on a disk with a corresponding hash table of its contents.

It is important to acknowledge that a partition in Swift is not a physical entity, i.e., a hardware component but a virtual container for the objects.

When a bunch of nodes collectively run the complete set of processes and services required to function as a distributed storage system, they form a Swift cluster.

Objects are added to partitions, which are the smallest units that Swift operates on. Each of these partitions is replicated, and each replica is placed as uniquely as possible across the cluster. In most cases, a replica count of three is chosen, which is also the default provided by Swift.

The Swift Ring is a modified consistent hashing ring (a set of lookup tables) that helps determine the physical location i.e., on which disk each object will be stored. The nodes containing these disks are referred to as the primary nodes associated with that particular object [4].

B. Energy consumption states of HDDs

Object storage systems use an enormous number of HDDs as the media for storage, which consumes a significant amount of energy. Each of these HDDs has a set of different energy consumption states.

B.1. Active mode

An HDD is said to be in the active mode or state when data is either being written to it or being read from it. When an HDD is in the active state, the disk platter rotates at high speed to serve the I/O requests [12].

B.2. Idle mode (E_i)

An HDD is said to be in idle mode or idle state when there is no data transfer in progress on the HDD, i.e. when data is neither being written to nor being read from the HDD. When an HDD is in idle state, even though it is not serving any I/O requests, the disk platter within is rotating at a slow speed causing the HDD to consume a considerable amount of energy [12].

B.3. Standby mode (E_s)

An HDD is said to be in standby mode when the power supply is cut off to the mechanical components of the disk. Hence, when an HDD is spun down, it enters the standby mode wherein the disk platter is not rotating. In standby mode, power is consumed only by the electronic components of the HDD and this energy consumption is negligible compared to that of the HDD in the active and idle modes [12].

C. Architectural Overview of Swift

In Swift, the proxy server stitches together the rest of the components. For each request, it will look up the partition locations of the object in the Swift ring and route the request to the corresponding storage nodes, where the object is stored [4]. The architectural overview of Swift is depicted diagrammatically in Figure 1.

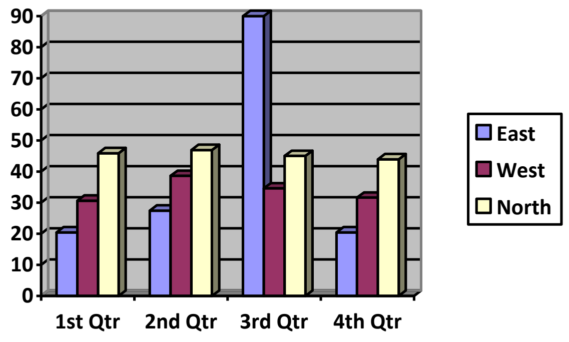

Fig. 1.Architectural Overview of Swift

D. Functional Overview of Swift

Now that the architectural overview of Swift has been established, we present the functional overview of Swift. In this subsection, we describe the critical events involved in the execution of a given operation in Swift.

Of all the operations that Swift supports we have focused on the most fundamental and critical operations which are the GET, PUT, UPDATE, and the DELETE operations.

D.1. GET Operation

In Swift, the time taken to respond to connection requests could be used as the primary factor to determine which one of the three nodes (containing the replicas) to retrieve the object from. When a proxy server requests a connection to a storage node, it tracks how long it took to create the connection.

To provide the best performance, Swift always tries to retrieve the object from the node that would respond the fastest.

A busy server will take longer to respond to connection requests. Hence, once the proxy server has the list of primary nodes containing replicas of the object to be retrieved, it ranks the nodes based on the saved connection timings. The object is then retrieved from the node that responds the fastest.

D.2. PUT Operation

In traditional Swift, to execute a PUT operation, a write request is sent to all the three primary nodes simultaneously. An acknowledgment is sent to the user that the operation is successful only after the write operation is completed successfully on a majority of the nodes, i.e., the quorum has been attained. In this case, with a replication factor of 3, the quorum is attained when the object is successfully written to any two out of the three nodes.

D.3. UPDATE Operation

Swift treats objects as discrete entities, hence it does not partially update already uploaded objects. Swift replaces the previously uploaded object with the current object and frees the space occupied by the older one. However, when object versioning is enabled, the older versions of an object are stored in a separate container. Hence, Swift does not require a dedicated UPDATE operation and makes use of the PUT operation for the same. A PUT request needs to be made with the same object name to update/modify an already uploaded object.

D.4. DELETE Operation

Object deletion in Swift is handled by creating a tombstone file (a zero-byte file with a .ts extension) as the latest version of the object. This tombstone file is pushed as the latest version to the other replicas by the replicator, eventually removing the object from the system. Since no disk activity is associated with a DELETE operation, we have not included any DELETE operations in our workloads.

E. Architectural Overview of SEA modeled for Swift

As shown in Figure 2, which represents the architectural overview of SEA modeled for Swift, an additional SSD staging layer has been introduced between the proxy server and the storage nodes. As mentioned earlier, SEA leverages this SSD staging layer to store the objects which are most likely to be requested for retrieval soon. This drastically reduces the number of objects to be retrieved from the object storage system layer lying underneath, comprising the storage nodes. This allows for the spinning down of a significant number of the HDDs in the storage nodes.

The HDDs in the object storage layer are logically classified into two zones, the active zone, and the standby zone to achieve this. The HDDs comprising the active zone are up and running at all times. The HDDs in the standby zone are typically spun down to conserve energy. Additionally, when the amount of free space in the SSD staging layer goes below a certain threshold, we spin up the HDDs in the standby zone and offload some objects to it. We refer to this process as the flush. Once the flush is complete, these HDDs can return to the inactive state.

This active and standby zone, into which the HDDs are logically classified are different from the active and standby states or modes of an HDD which refers to the inherent energy consumption states of an HDD as explained in Section II.B. On the other hand, the classification of the HDDs into the active and standby zones is to determine which set of HDDs are always up and running, i.e. in the active zone and which would be spun down regularly, i.e. in the standby zone. Hence, any HDD in the active zone would be in one of two possible energy consumption states, i.e. active state or idle state at any given point in time as they are always up and running. If the HDD is serving an I/O request it would be in the active state; else, it would be in the idle state. On the other hand, since HDDs in the standby zone could be spun down, an HDD in the standby zone could be in one of three possible energy consumption states, i.e. active, idle or standby state. When spun down, the HDD would be in the standby state, whereas when it is spun up during the flush, it is in either the active state or idle state based on whether it is serving an I/O request at that instant in time.

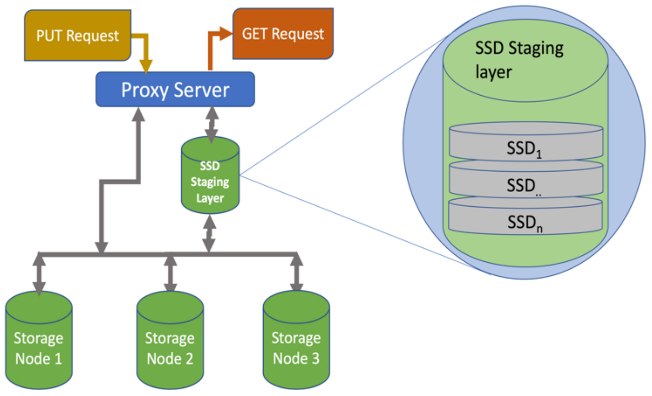

Fig. 2.Architectural overview of SEA modeled for Swift

Assuming that the replication factor is 3 (which is the default), to preserve the availability, we always ensure that at most, only one replica of any partition is placed in the standby zone. Hence, as soon as an object is uploaded, one replica is present in the staging layer, and two replicas are in the active zone, thus resulting in a total of three replicas. Post flush, one more replica of the object is created in the standby zone and the corresponding replica in the staging layer is deleted, thus maintaining a total of at least three replicas at any point in time as assured.

The core principle involved in classifying the data storage disks into the active and standby zones is to have as many disks as possible in the standby zone with no two of them containing replicas of the same partition. The steps involved in classifying the disks into the active and standby have been described in detail in Algorithm 1. As we iterate through each of the HDDs in the object storage layer, we check for common partitions between this HDD and the HDDs that are already part of the standby zone. If there are any common partitions, then the HDD is added to the active zone. However, if there are no common partitions, then the HDD is added to the standby zone.

F. Functional Overview of SEA modeled for Swift

In this subsection, we present the crucial differences between the workflow, execution of the basic operations in SEA modeled for Swift and traditional Swift.

```
INPUT: allDisks, diskPartitionMapping
OUTPUT: standbyZoneDisks, activeZoneDisks

procedure
 1:  standbyZoneDisks ← []
 2:  activeZoneDisks ← []
 3:  standbyZonePartitions ← []
 4:  for disk in allDisks do:
       //get all partitions associated with this disk
 5:    diskPartitions ← diskPartitionMapping.get(disk)
       //check if this disk has any partitions, whose replica
       //has already been allotted to standby zone
 6:    commonPartitions ← setIntersection (diskPartitions,
            standbyZonePartitions)
       // if none of the replicas of these partitions
       // are in standby zone
 7:    if commonPartitions.length == 0 then :
          // we can add this disk to standby zone
 8:       add disk to standbyZoneDisks
 9:       add diskPartitions to standbyZonePartitions
10:    else :
11:       add disk to activeZoneDisks
12:  return standbyZoneDisks, activeZoneDisks
```

Algo.1. Classification of disks into standby and active zones

F.1. GET Operation

The workflow of the GET operation is modified in the SEA model to have only the replicas placed in the active zone HDDs and the SSD staging layer is considered as candidates to retrieve the object and ignoring the standby zone HDDs as described in Algorithm 2. Hence, an HDD in the standby zone is never spun up to serve a customer request, thereby avoiding the additional latency involved in spinning up the HDD. It is important to note that the object is not always retrieved from the staging layer. Though, the SSDs in the staging layer is much faster than the HDDs in the object storage layer, directing all the read requests to the SSD staging layer alone could result in the queuing up of a large number of requests. Hence, we still determine the node to retrieve the object from based on the connection timings, as in traditional Swift, thus ensuring that the object is retrieved in the fastest possible manner.

```
INPUT: SwiftRing, requestedObjectName
OUTPUT: requestedObject

procedure
     // get the list of all the primary nodes in
     // active zone* containing replicas of requested object
 1:  activePrimaryNodes ← SwiftRing.getPrimaryNodesFromActiveZone(
          requestedObjectName)
     // consider all the data storage nodes in active zone
 2:  nodes ← activePrimaryNodes
 3:  If requestedObject in SSD staging layer then :
        // get the SSD node containing requested object
 4:     ssdNode ← getSSDNode(requestedObjectName)
        // add this to the list of nodes containing the replicas
 5:     nodes ← nodes + ssdNode
     // fetch the time taken to respond to
     // connection requests by these nodes
 6:  nodeResponseTimes ← getTimeToRespond(nodes)
     // get the node that responds the fastest
 7:  fastestNode ← getFastestNode(nodes, nodeResponseTimes)
     // fetch object from this node
 8:  requestedObject ← getObject(fastestNode, requestedObjectName)
 9:  return requestedObject
```

Algo.2. GET operation in SEA modeled for Swift

F.2. PUT Operation

The PUT operation workflow in the SEA model is modified for the write requests to be initiated to all of the primary nodes in the active zone and the SSD staging layer. The objects are not written immediately to the nodes in the standby zone as described in Algorithm 3. Additionally, when the amount of free space in the SSD staging layer goes below a certain threshold, the flush operation is triggered. As a part of the flush operation, the nodes in the standby zone are spun up, and the staging algorithm in place determines the objects to be offloaded. These objects are copied onto the HDDs in the standby zone either by retrieving them from the staging layer or from the active zone, whichever is faster. Once this transfer is complete, the memory is freed on the SSD staging layer, and the nodes that were spun up can return to the inactive state as described in Algorithm 4.

It is possible for the HDDs in the standby zone to contain outdated data, for instance, an older version of an object or replicas of already deleted objects.

```
INPUT: Object, SwiftRing
OUTPUT: ACK once a quorum is achieved

procedure
     // get the list of all the primary nodes in the active zone*
 1:  activePrimaryNodes ← SwiftRing.getActivePrimaryNodes(objectName)
     // get SSD node to write the object to
 2:  ssdNode ← getSSDNode(objectName)
 3:  nodes ← activePrimaryNodes + ssdNode
     // initiate the write to all these nodes simultaneously
 4:  putObject(nodes, object)
     // wait until the write is complete on a majority of the nodes
 5:  while write not complete on a majority of primaryNodes do:
 6:     wait
     // quorum achieved
     // if the free space on SSD staging layer is too low then trigger flush
 7:  if SSDstagingLayer.freeSpace < lowerThreshold do:
        // do not wait for flush to complete, it will execute in the background
 8:     triggerFlush() // defined in algo6
 9:  return ACK
```

Algo.3. PUT operation in SEA modeled for Swift

```
INPUT: SwiftRing

procedure
 1:  Based on the staging algorithm and flush thresholds
        in place, determine objects to be offloaded from
        the SSD staging layer to standby zone
 2:  From these objects, determine the disks in
        standby zone that need to be spun up
        (with the help of the SwiftRing)
 3:  Retrieve each of these objects from either the staging
        layer or from the active zone, whichever is faster
 4:  As each object is transferred to HDDs in the standby
        zone, delete it on SSD staging layer
 5:  Once complete, spin down the standby zone disks
 6:  End of flush
```

Algo.4. Flush

This is because UPDATE and DELETE operations are not immediately replicated to the HDDs in the standby zone instead, these changes are propagated once these HDDs are spun up during a flush operation. However, this does not violate Swift's principle of consistency as Swift offers eventual consistency in place of immediate consistency.

Also, since none of the customer requests are served by the HDDs in the standby zone, an older version of an object is never returned to the customer due to this design decision. Hence, without any undesirable effects, the energy consumption and the number of times the HDDs are required to be spun up and down is reduced due to this design.

III. METHODOLOGY

We have looked to evaluate our architecture against a large, real- life-sized data center and workload. Considering the cost and practicality of setting up such a physical environment, we have used simulation which provides an expeditious and cost-effective way of studying the behavior of large data centers and working with massive workloads.

We have used CloudSim [13], an event-based standard open source cloud simulator for our simulation. This enables us to design and build a data center that can be customized based on several parameters such as hosts, VMs, storage, and scheduling policies. We have also used CloudSimDisk [14] an extended module in CloudSim to simulate storage devices with high precision while factoring in the disk characteristics such as speed, latency, and capacity.

The formulae for the calculation of the energy consumed in spinning up/down a disk and the total amount of energy consumed by the HDDs and SSDs throughout the duration of the simulation are presented below.

A. Spinning up a disk

The total amount of energy required to spin up an HDD is computed using the following formula:

$$E_{spU} = P_{spU} \times t_{sr} \qquad (1)$$

where E_{spU} is the energy consumed by the disk to be spun up, P_{spU} is the spin-up power consumption rate of the disk, and t_{sr} is the time taken by the disk to switch from standby to ready state.

B. Spinning down a disk

The total amount of energy consumed in spinning down an HDD is computed using the following formula:

$$E_{spD} = P_{spD} \times t_{rs} \qquad (2)$$

where E_{spD} is the energy consumed by the disk to be spun down, P_{spD} is the spin-down power consumption rate of the disk and t_{rs} is the time taken by the disk to switch from ready to standby state

For an HDD, the total amount of energy consumed is calculated using the formula:

$$E_{tot} = E_a + E_i + E_s + (n_{su} \times E_{spU}) + (n_{sd} \times E_{spD}) \qquad (3)$$

where E_{tot} is the total energy consumed by the disk throughout the duration of the experiment; $E_{a,}$ $E_{i,}$ E_s are the total energy consumed by the disk in active mode, idle mode, and standby mode respectively, n_{su} is the number of times the disk has been spun up and n_{sd} is the number of times the disk has been spun down.

For an SSD, the total amount of energy consumed is calculated using the formula:

$$E_{tot} = E_a + E_i \qquad (4)$$

Since the SSDs are continuously running, the total energy consumed by each SSD equals the sum of the energy consumed by the SSD in active state and idle state.

IV. WORKLOAD

In this section, we describe the software used for the generation of the workload. We also detail the nature and composition of the workload used for the simulation.

A. Workload Generation

We believe that the best way to evaluate SEA would be to use a workload that is realistic and similar to one captured from a production environment of a large object storage system in terms of the amount of data uploaded and the rate at which the requests arrive. Since we wanted to use workloads that had a realistic inter-arrival time between the requests, we decided not to use object storage benchmarking tools such as COSBench [2]. Instead, we used CloudGen [15], a synthetic workload generator capable of reproducing the behavior of Dropbox customers. Also, the amount of data uploaded needed to be on par with realistic large-scale data centers. For instance, we found that around 1 PB of data was being uploaded to YouTube daily [16] as of 2015.

Thus, in CloudGen, we set the synthetic workload duration to 24 hours and tuned the other parameters until we got a workload such that the sum of the sizes of all the objects in the system at the end was above 2.5 PB. The last operation in the workload arrived at 86399.916 seconds after the arrival of the first operation. The sum of the sizes of all the objects present in the system at the end of the simulation was 2.511 PB. This workload is described in further detail in the following subsection.

B. Workload Composition

Table 1 depicts the workload composition split across the different types of operations. This distribution of the percentage of each type of operation is also modeled based on DropBox customer behavior. Hence, this is a read-intensive workload, with GET operations constituting 61.14% of the total workload.

Table 1 depicts the workload composition split across the different types of operations. This distribution of the percentage of each type of operation is also modeled based on DropBox customer behavior. Hence, this is a read-intensive workload, with GET operations constituting 61.14% of the total workload.

Table- I: Workload Composition

Operation	% constitution in terms of no. of operations	Data transferred (in PB)
PUT	16.34	1.543
UPDATE	22.52	1.958
GET	61.14	7.0901

V. EVALUATION

In this section, we present the framework for the evaluation of the energy efficiency and the performance of SEA modeled for Swift.

A. Data Center Characteristics

For this evaluation, along with a realistic workload we also needed to model a realistic data center. Therefore, we decided to build the data center with disk models that are currently being used in large-scale enterprise data centers. Hence, for HDDs, we chose the Seagate ST4000DM000 model, which is currently being used in the data centers of Backblaze, a popular data storage provider [17]. For SSDs, we chose the 1.6TB variant of Intel Solid State Drive (SSD) Data Center S3510 Series which was used by Tarox, an IT service provider to improve the 3D rendering process for one of their clients [18].

Table 2 and Table 3 show the hardware specifications of these disks, which we have used for our simulations. Also, given that 2.511 PB of data is being added to the object storage system with a replication factor of 3 and with the capacity of each HDD being 4TB, around 2048 HDDs are required in the object storage layer to accommodate this workload.

Table- II: The Seagate ST4000DM000 specifications [19]

Attributes	Values
Model name	Seagate ST4000DM000
Type	HDD
Capacity (TB)	4
Speed (MB/s)	146
Latency (ms)	5.16
Average Seek Time(ms)	12
Active Power (W)	7.5
Idle Power (W)	5
Standby Power (W)	0.75
Spinup Power (W)	24[a]
Time to switch from Standby to ready (s)	15
Time to switch from Ready to Standby (s)	10

Table- III: The Intel Solid State Drive (SSD) Data Center S3510 1.6TB variant specifications [20]

Attributes	Values
Model name	Intel S3510 Series
Type	SSD
Capacity (TB)	1.6
Read Latency (ms)	0.055
Write Latency (ms)	0.066
Read rate (MB/s)	500
Write Rate (MB/s)	460
Active Power (W)	5.6
Idle Power (W)	0.6

[a.] Spinup Power = Voltage * Spinup current = 12V * 2.0 A = 24W

VI. RESULTS AND DISCUSSION

A. A comparison between traditional Swift and SEA modeled for Swift

Most traditional data centers consist solely of HDDs. However, of late SSDs are being added to the data centers alongside the HDDs to achieve better performance [9]. Since most data centers fall in either of these two categories, we consider these as the baselines against which we compare SEA. First, we evaluate the impact that the addition of SSDs alongside HDDs in a traditional Swift setup has on the total energy consumption and performance. We then compare this model against the addition of SSDs as part of a staging layer. This helps determine whether the addition of SSDs to a traditional data center improves energy efficiency and performance in the first place. If yes, we could determine which of the two approaches to leverage these small number of SSDs to maximize energy efficiency and performance enhancement is more effective, i.e., whether to add them alongside the HDDs or to introduce an SSD staging layer above the object storage system.
The first baseline, i.e., Baseline1 is that of the traditional Swift setup comprising only HDDs in the object storage layer and the second baseline, i.e., Baseline2 comprises a few SSDs alongside the HDDs in the object storage layer. Neither of these two has any form of a staging layer.

Table- IV: Logical Distribution of Disks across the different Swift setups

Variations	Total No. of HDDs	No. of HDDs in active zone	No. of HDDs in standby zone	No. of SSDs
Baseline 1	2048	2048	-	-
5% SSD Baseline 2	2048	2048	-	256
10% SSD Baseline 2	2048	2048	-	512
15% SSD Baseline 2	2048	2048	-	768
5% SSD SEA	2048	1439	609	256
10% SSD SEA	2048	1439	609	512
15% SSD SEA	2048	1439	609	768

Table 4 depicts the logical distribution of the disks across the different Swift setups, each having varying SSD capacities. In the setups running SEA, the HDDs that are part of the standby zone are spun down. The SSDs in the Baseline2 setups are part of the object storage layer, located alongside the HDDs whereas the SSDs in the SEA setups form a separate staging layer.

A.1 A comparison between Baseline1 and Baseline2

Figure 3 shows that the Baseline1 variation of the traditional Swift setup comprising just the 2048 HDDs consumed 1226.702 MJ. On adding more disks to the object storage layer, we would expect the total energy consumption to increase considerably, but this is not the case when a small number of SSDs are added. This scenario is represented by Baseline2, and we see that the addition of SSDs having 5% of the capacity of the HDDs has reduced the total energy consumption to 1196.089 MJ.

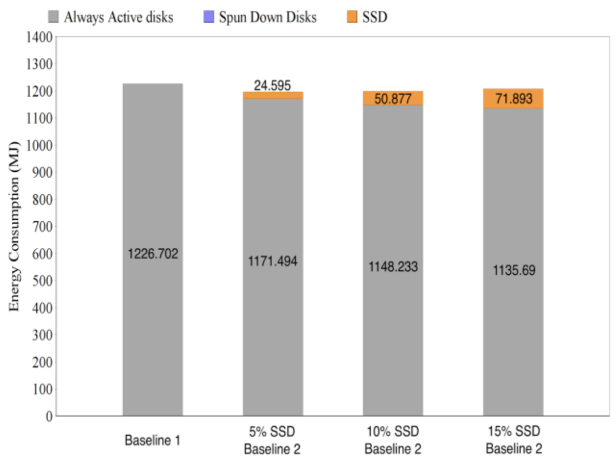

Fig. 3.Baseline 1 vs 5% SSD Baseline 2 vs 10% SSD Baseline 2 vs 15% SSD Baseline 2 energy consumption comparison

In Baseline2, a small number of requests are served by the SSDs, which reduces the load on the HDDs. Thus, compared to Baseline1, the HDDs in Baseline2 are in an idle state for longer intervals of time. For the HDDs that we have chosen for this simulation, the power consumption in the idle state (i.e., 5W) is 33% lesser than that in the active state (i.e., 7.5W). The power consumed by the SSDs to serve these requests is significantly lower as SSDs have a much lower active state power consumption (5.6 W) and shorter transfer times as they are faster than HDDs (more than three times as fast in this case). Additionally, the idle state power consumption of SSDs is negligible (0.6 W). Hence, the decrease in the energy consumption of HDDs is greater than the additional energy consumed by the SSDs to serve this set of requests, resulting in a modest decrease in the total energy consumed by the entire setup.

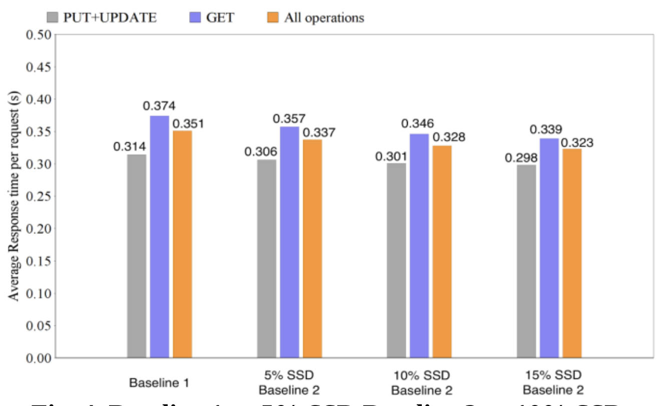

Fig. 4. Baseline 1 vs 5% SSD Baseline 2 vs 10% SSD Baseline 2 vs 15% SSD Baseline 2 performance comparison

Along with this 2.495% drop in energy consumption, we also see a 3.989% improvement in the performance of the 5% SSD Baseline2 setup when compared to Baseline1 as shown in Figure 4. SSDs, being much faster than HDDs, improve the overall performance of Baseline2. Additionally, due to HDDs serving fewer requests, the queueing up of the requests reduces at the HDDs, resulting in faster responses.

We observe similar behavior in the 10% SSD and 15% SSD Baseline2 setups when compared against Baseline1. The 10% SSD Baseline2 setup consumes 2.249% lesser energy as compared to Baseline1. Due to the addition of more SSDs, the 10% SSD Baseline2 setup performs 6.553% better than Baseline1. The 15% SSD Baseline2 setup consumes 1.559% lesser energy as compared to Baseline1. The 15% SSD Baseline2 setup performs 7.97% better than Baseline 1.

From Figure 4, we see that the performance of the system improves with the addition of SSDs. This is a direct consequence of more requests being served by SSDs. However, as shown in Figure 3, the total energy consumption has also increased with the increase in no. of SSDs. When a large number of SSDs are added, the increase in the energy consumption of the newly added SSDs is greater than the decrease in the energy consumption of the HDDs that it has caused.

Due to this reason, the 5% SSD Baseline2 setup is the most energy-efficient of the three, as it has the right proportion of SSDs to HDDs.

In this architecture, the SSDs can serve requests only for the objects associated with the partitions it contains. With a small percentage of SSDs, the number of partitions that could be accommodated in the SSDs is also small and hence the percentage of traffic being served by the SSDs is also small. Though the SSDs can handle a considerably higher load, they are in the idle state for the majority of the time. Thus, only a small amount of load is reduced on the HDDs, and the decrease in the energy consumed by the HDDs is insignificant. Consequently, the addition of SSDs causes the total energy consumed by the entire setup to increase. Though this architecture has shown some energy efficiency and performance improvement on the addition of a small number of SSDs, it does not leverage the presence of the SSDs effectively.

However, this problem is overcome in SEA as the SSDs are strategically placed in a staging layer, wherein the whole partitions are not required to be stored. Only the most frequently requested objects across all partitions are to be stored in the staging layer. Thus, a majority of the traffic is served by the staging layer, thereby minimizing the number of requests needed to be served by the HDDs. This provides a massive performance boost and drastically reduces the energy consumed by the HDDs. Additionally, it also supports the spinning down of a substantial number of HDDs (29.736% of the HDDs are in the standby zone for the simulations shown), which significantly reduces the energy consumed by the HDDs

A.2 A comparison between Baseline1 and SEA

The 5% SSD SEA setup consumes 12.426% lesser energy as compared to Baseline1, despite having an additional 256 SSDs as shown in Figure 5. With 5% SSD SEA, the performance has also improved by 14.815% as shown in Figure 6. Similarly, as compared to Baseline1, the 10% SSD SEA and 15% SSD SEA consume 18.917% and 19.967% lesser energy respectively. Additionally, as compared to Baseline1, the performance of the 10% SSD SEA and 15% SSD SEA has also improved by 23.362% and 25.641% respectively. Hence, we see that despite the introduction of additional SSDs into the system, the energy consumption has decreased significantly while improving the performance.

A.3 A comparison between 5% SSD SEA, 10% SSD SEA and 15% SSD SEA

Figure 5 shows the energy consumed (in megajoules) across SEA setups with varying SSD staging capacities (as a % of the total HDD capacity). When we moved from the 5% SSD SEA to 10% SSD SEA setup, we observed a drop in the energy consumption by 3.976% and an improvement in performance by 10.033%. This drop in the energy consumption is due to the decrease in the energy consumed by the HDDs in active zone.

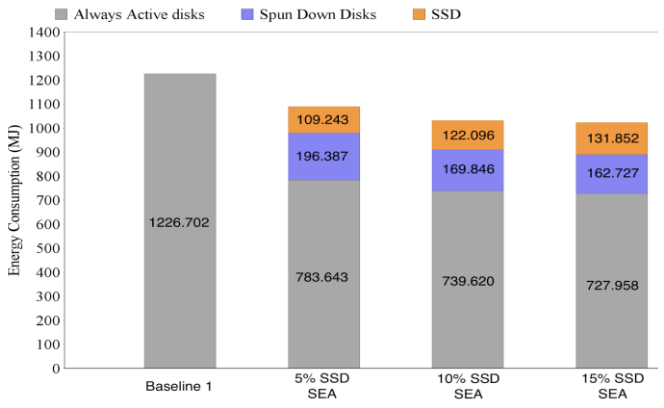

Fig. 5. Baseline 1 vs 5% SSD SEA vs 10% SSD SEA vs 15% SSD SEA energy consumption comparison

As the staging layer capacity increases, it can serve a higher number of requests, thus reducing the number of requests needed to be served from the disks below and also reduces the frequency of flushes needed for the staging layer as it would take longer for it to run out of memory. Hence, the HDDs in the standby zone are required to be spun up less frequently thereby consuming much lesser energy. However, when we moved from the 10% to 15% SSD staging layer scenario, we saw a drop of only 0.875% in energy consumption. When the percentage of SSDs is increased to 15, the energy conserved by the slight decrease in the number of flushes (due to the addition of more SSDs) and misses in the staging layer is negated by the energy consumed by the newly added SSDs as shown in Figure 5. This is because the nature of the workload is such that only a few of the objects are requested frequently, as is the case with most workloads. Hence, there is no advantage gained by the movement of the objects which are not requested frequently from the HDD object storage layer to the staging layer as the total number of misses reduced is trivial. Hence, the addition of more SSDs to this setup would not help reduce energy consumption significantly as the majority of the traffic is already served by the staging layer.

Figure 6 shows that there is a significant improvement in performance with the staging layer capacity in SEA. Since with a larger staging layer, a fewer number of objects will need to be retrieved from the underlying HDD, we observe an improvement in performance for the GET operations, as most objects will be retrieved from the SSD staging layer which is ~three times faster than HDDs. The quorum is achieved a little earlier for the PUT operation. For a replication factor of 3, the quorum is attained when the object has been written to any two of the three nodes. Given that one of these three nodes is an SSD, the quorum is attained a little earlier than against Baseline2. However, the improvement in the performance of PUT operations is not as large as that observed in the GET operations. The PUT operation requires that the object is written to at least one HDD before the acknowledgment is sent to the user, which acts as a bottleneck, whereas for GET operations, no reads from the HDDs are required for a majority of the requests.

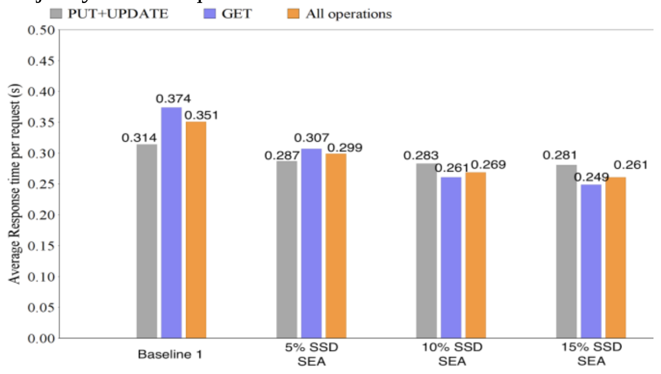

Fig. 6. Baseline 1 vs 5% SSD SEA vs 10% SSD SEA vs 15% SSD SEA performance comparison

Thus, this architecture using SSDs effectively and significantly improves energy efficiency and performance for the majority of the traffic.

Compared to Baseline2, SEA leverages the presence of SSDs more effectively to improve performance and energy efficiency. Even though the 5% SSD Baseline2 and the 5% SSD SEA setups have the same number of HDDs and SSDs, the 5% SSD SEA setup is 10.185% more energy-efficient and offers a performance improvement of 11.276% compared to the 5% SSD Baseline2 as shown in Figure 7 and Figure 8. This is because SEA leverages the high speed and low power consumption characteristics of SSDs to serve the majority of the requests and spins down a significant number of HDDs.

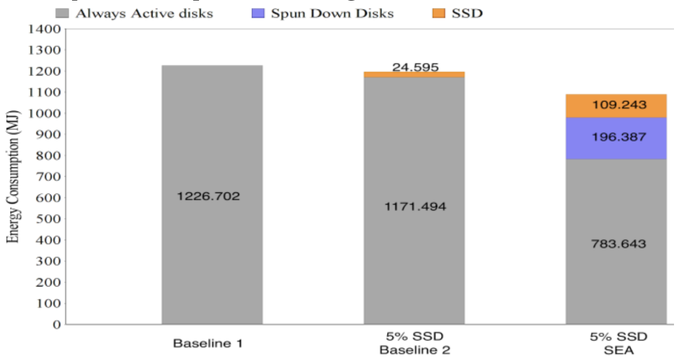

Fig. 7. Baseline 1 vs 5% SSD Baseline 2 vs 5% SSD SEA energy consumption comparison

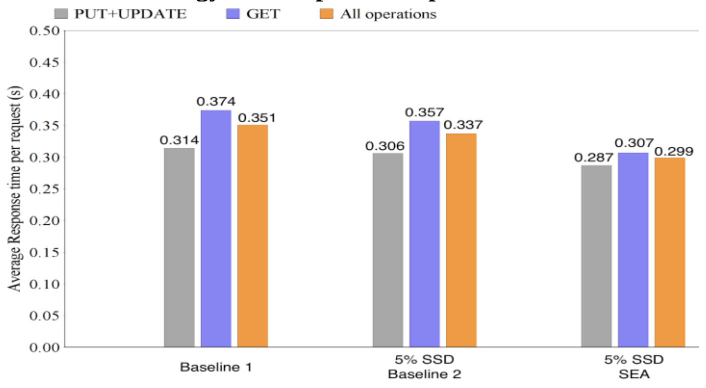

Fig. 8. Baseline 1 vs 5% SSD Baseline 2 vs 5% SSD SEA performance comparison

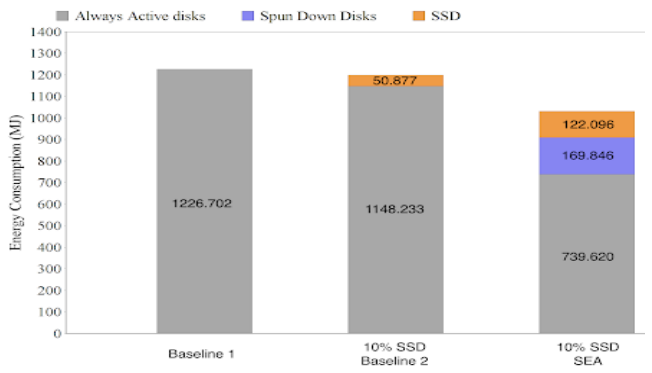

Fig. 9. Baseline 1 vs 10% SSD Baseline 2 vs 10% SSD SEA energy consumption comparison

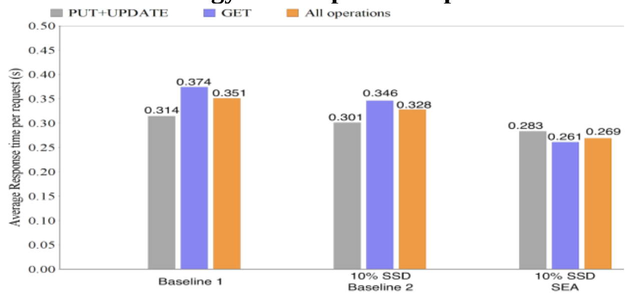

Fig 10. Baseline 1 vs 10% SSD Baseline 2 vs 10% SSD SEA performance comparison

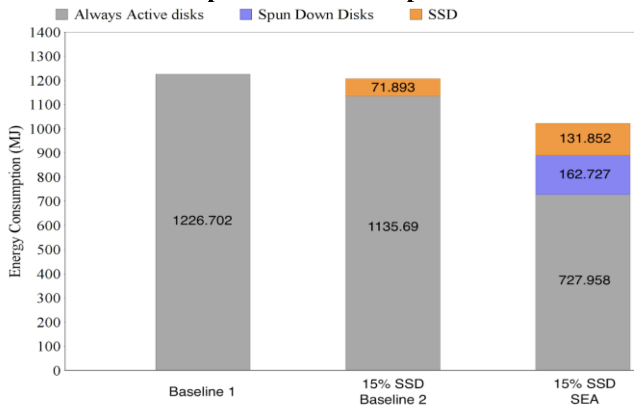

Fig. 11. Baseline 1 vs 15% SSD Baseline 2 vs 15% SSD SEA energy consumption comparison

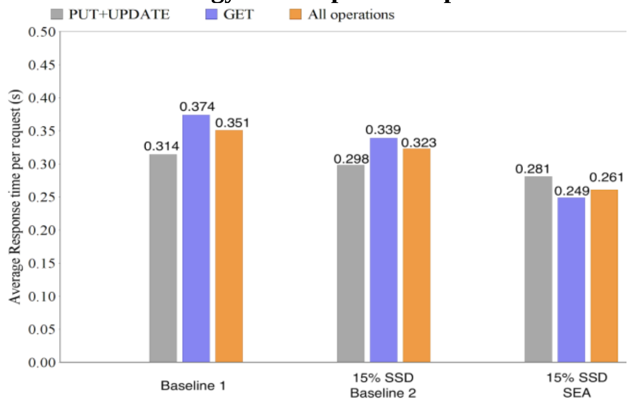

Fig. 12. Baseline 1 vs 15% SSD Baseline 2 vs 15% SSD SEA performance comparison

We observe the same behavior when we compare Baseline2 against SEA with 10% and 15% SSD capacities as shown in Figure 9 to Figure 12. Figure 9 shows that the 10% SSD SEA setup consumes 13.973% lesser energy as compared to the 10% SSD Baseline2 setup. We see that due to the addition of more SSDs, the 10% SSD SEA setup performs 17.988% better than 10% SSD Baseline2 from Figure 10. The 15% SSD SEA setup consumes 15.324% lesser energy as compared to 15% SSD Baseline2 as shown in Figure 11. From Figure 12 we see that the 15% SSD SEA setup performs 19.195% better than 15% SSD Baseline2 setup. Thus, we see that given any number of SSDs, SEA provides significantly better performance and energy efficiency compared to Baseline 2. Interestingly, 5% SSD SEA was more energy-efficient and performed better than any of the Baseline1 and Baseline2 variations. For an object storage system model with a similar workload, 5% SSD SEA is the recommended option if energy efficiency is the primary requirement along with significant performance improvement. Compared to Baseline1, this model achieves a 12.426% decrease in energy consumption and additionally improves the performance by 14.815% while using a reasonable number of SSDs. On the other hand, if extensive performance improvement is the primary requirement, it is recommended to use 10% SSD SEA. Compared to Baseline1, this model was able to improve the performance by 23.362% while still reducing energy consumption by 15.908% by leveraging a few more SSDs compared to the previous 5% SSD SEA model.

VII. CONCLUSION AND FURTHER WORK

In this paper, we proposed SEA: An SSD staged energy-efficient object storage system Architecture that conserves energy and improves the performance of the object storage system either by leveraging the SSDs already present or by adding a small number of SSDs to the data center. We have shown that the introduction of SSDs as part of a staging layer is more energy-efficient and better in terms of performance compared to the addition of SSDs as part of the object storage layer, alongside the HDDs. This architecture is modeled such that it can be customized to suit the specific needs of the data center operator. For example, if the requirement is primarily energy efficiency coupled with significant performance improvement, then an SSD staging layer of a small capacity should provide significant energy savings and performance improvement. With a staging layer comprising SSDs having 5% of the total object storage capacity, SEA was able to reduce energy consumption by 12.426% and improve performance by 14.815% as compared to the traditional Swift setup comprising only HDDs. In this paper, we have introduced a new algorithm that prioritizes customer requests over background tasks and observed that it further improved the performance of SEA by 7.435%. Thus, for more complex requirements involving both energy efficiency and extensive performance enhancement, an SSD staging layer of a slightly larger capacity coupled with the prioritization of customer requests over background tasks would work well. As shown in our results, with a staging layer comprising 10% of the capacity of the object storage layer and with customer requests prioritized over background tasks, the performance improved by 29.06%, and the energy consumption reduced by 15.235% as compared to traditional Swift with only HDDs.
Given the rapid increase in the rate at which unstructured data is being generated, in the coming years, the improvement in energy efficiency and performance brought about by SEA is going to increase further.

Also, when a sudden spike is observed in the traffic, all the disks in the standby zone could be spun up to serve the increasing number of requests without any deterioration in performance. These disks could again be spun down once the traffic has gone down to normal. We intend to further improve SEA, by building an intelligent staging algorithm that leverages machine learning methods to recognize workload patterns dynamically and use that to manage the objects in the staging layer more efficiently. We would like to evaluate this staging algorithm against other staging algorithms such as FIFO and LRU. Planned future works also include studying our architecture from an availability and reliability standpoint.

REFERENCES

1. I.A.T. Hashem, et al., The rise of “big data” on cloud computing: Review and open research issues, Information systems 47 (2015) 98–115.
2. Q. Zheng, et al., Cosbench: A benchmark tool for cloud object storage services, in: 2012 IEEE Fifth International Conference on Cloud Computing, IEEE, 2012, pp. 998–999.
3. G.G. Mesnier, M.E. Riedel, Object-based storage., in: IEEE Communications Magazine, 41(8), IEEE, 2003, pp. 84–90.
4. J. Arnold, Openstack swift: Using, administering, and developing for swift object storage” O’Reilly Media, Inc.”, 2014.
5. C. Weddle, et al., Paraid: A gear-shifting power-aware raid, ACM Transactions on Storage (TOS) 3 (3) (2007) 13.
6. Q. Zhu, A. Shankar, Y. Zhou, Pb-lru: a self-tuning power aware storage cache replacement algorithm for conserving disk energy, in: ACM International conference on Supercomputing, 2004, pp. 79–88.
7. J. Shuja, et al., Survey of techniques and architectures for designing energy-efficient data centers, IEEE Systems Journal 10 (2) (2016) 507–519.
8. E. Pinheiro, et al., Exploiting redundancy to conserve energy in storage systems, in: ACM SIGMETRICS, Vol. 34, ACM, 2006, pp. 15–26.
9. A. Shehabi, et al., United states data center energy usage report (2016).
10. Amazon web services. amazon simple storage service (amazon s3), https://aws.amazon.com/s3, last accessed on 2019-03- 05.
11. Google cloud platform. google cloud storage (gcs), https://cloud.google.com/storage/, last accessed on 2019-03-05.
12. S.R.R.A. Hylick, A.B. Jones, An analysis of hard drive energy consumption., in: IEEE Modeling, Analysis and Simulation of Computers and Telecommunication Systems, IEEE, 2008, pp. 1–10.
13. R.N. Calheiros et al., Cloudsim: a toolkit for modeling and simulation of cloud computing environments & evaluation of resource provisioning algorithms, Software: Practice and experience 41 (1) (2011) 23–50.
14. B. Louis, et al., Cloudsimdisk: Energy-aware storage simulation in cloudsim, in: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), IEEE, 2015, pp. 11–15.
15. G. D. Goncalves, et al., Workload models and performance evaluation of cloud storage services, Computer Networks 109 (2016) 183–199.
16. E. Brewer, et al., Disks for data centers (2016).
17. A. Klein, Backblaze hard drive stats for 2016, https://www.backblaze.com/blog/2018-hard-drive-failure-rates/, last accessed on 2019-03-05 (2016).
18. Intel Case study. Tarox creates new, more profitable business model with Intel Technology, https://www.intelserveredge.com/wp-content/uploads/assets/Tarox-Case-Study-Final-May-2017-1.pdf, last accessed on 2019- 03-05.
19. Seagate desktop HDD product manual, https://static.bhphotovideo.com/lit_files/96934.pdf, last accessed on 2019- 03-05.
20. Intel Solid State Drive Data Center S3510 Series Product Brief, https://lenovopress.com/lp0056.pdf, last accessed on 2019-03-05.
21. A. Verma, et al., Srcmap: Energy proportional storage using dynamic consolidation., in: FAST, Vol. 10, 2010, pp. 267–280.
22. D. Colarelli, D. Grunwald, Massive arrays of idle disks for storage archives, in: SC’02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, IEEE, 2002, pp. 47–47.
23. T. Xie, Y. Sun, Sacrificing reliability for energy saving: Is it worthwhile for disk arrays? in: International Symposium on Parallel and Distributed Processing, IEEE, 2008, pp. 1–12.
24. J.-W. Jang et al., Energy reduction in consolidated servers through memory-aware virtual machine scheduling, IEEE Transactions on Computers 60 (4) (2011) 552–564.
25. A. Murtazaev, S. Oh, Sercon: Server consolidation algorithm using live migration of virtual machines for green computing, IETE Technical Review 28 (3) (2011) 212–231.
26. N. Kord, H. Haghighi, An energy-efficient approach for virtual machine placement in cloud based data centers, in: The 5th Conference on Information and Knowledge Technology, IEEE, 2013, pp. 44–49.
27. Saving data center power by reducing hdd spin speed, https://www.opencompute.org/news/saving-data-center-power-by-reducing-hdd-spin-speed, last accessed on 2019-03-05.
28. E. Pinheiro, et al., Load balancing and unbalancing for power and performance in cluster-based systems (2001).
29. E.V. Carrera, et al., Conserving disk energy in network servers, in: Proceedings of the 17th annual international conference on Supercomputing, ACM, 2003, pp. 86–97.
30. D. Li, J. Wang, Eeraid: energy efficient redundant and inexpensive disk array, in: Proceedings of the 11th workshop on ACM SIGOPS European workshop, ACM, 2004, p. 29.
31. Q. Zhu, et al., Hibernator: Helping disk arrays sleep through the winter, SIGOPS Oper. Syst. Rev. 39 (5) (2005) 177–190. doi:10.1145/1095809.1095828. URL http://doi.acm.org/10.1145/1095809.1095828
32. X. Yao, J. Wang, Rimac: A novel redundancy-based hierarchical cache architecture for energy efficient, high performance storage systems, in: ACM SIGOPS Operating Systems Review, Vol. 40, 2006, pp. 249–262. doi:10.1145/1217935.1217959.
33. A. G. Yoder, Energy Efficient storage tech for data centers (2010).
34. G. Mathur, et al., Capsule: an energy-optimized object storage system for memory-constrained sensor devices, in: International conference on Embedded networked sensor systems, ACM, 2006, pp. 195–208.
35. L. Wu, et al., Boss:An efficient data distribution strategy for object storage systems with hybrid devices, IEEE Access 5 (2017) 23979–23993.
36. J. Axboe, A.D. Brunelle, Blktrace user guide (2007).
37. Seagate backup plus slim portable drive review, https://www.storagereview.com/seagate_backup_plus_slim_portable_drive_review, last accessed on 2019-03-05.
38. Samsung wireless and Seagate backup plus external hdds, http://www.silentpcreview.com/article1435-page5.html, last accessed on 2019-03-05.

AUTHORS PROFILE

Sai Vishwas Padigi received his BTech in Computer Science and Education from PES Institute of Technology, Bangalore, India. His areas of research interest include sustainable computing, cloud computing, AI and ML.

Sameer Kulkarni completed his BTech in PES Institute of Technology, Bangalore, India in the field of Computer Science and Engineering. His research interests include cloud engineering, sustainable computing, and data science.

Phalachandra HL received his Bachelor’s degree from Karnataka University and Masters from BITS Pilani, India. He has worked extensively in the Industry for 25+ years and currently teaches at PES University for the last 8 years. His research interests are in Storage Networks, Cloud Computing and Software Engineering.

Dr. Dinkar Sitaram completed his Ph.D. in Computer Science from the Univ of Wisconsin Madison. Cloud & Hybrid Clouds, Speech Recognition and Intelligent Big Data systems are his areas of interest. He has authored 2 books ’Moving to the and ’Multimedia Servers’. He has authored over 30 patents and 45 publications. He was awarded the IBM Corporate Innovation Award. He has served as CTO of Novell Software, Andiamo Systems and STSD (R&D Wing of HP, India) prior to joining PES University.