VENU: Orchestrating SSDs in Hadoop Storage

Krish K.R., M. Safdar Iqbal, Ali R. Butt
Department of Computer Science, Virginia Tech
Email: {kris, safdar, butta}@cs.vt.edu

***Abstract*—A major obstacle in sustaining high performance and scalability in the Hadoop data processing framework is managing the growing data and the need for very high I/O rates. Solid State Disks (SSDs) are promising and are being employed alongside the slower hard disk drives (HDDs) in emerging storage architectures. However, we observed that SSDs are not always a cost-effective option for all Hadoop workloads, and there is a critical need to identify usecases where SSDs can help. To this end, we present VENU, a dynamic data management system for Hadoop. VENU aims to improve overall I/O throughput via effective use of SSDs as a cache for the slower HDDs, not for all data, but for only the workloads that are expected to benefit from SSDs. In addition, we design placement and retrieval schemes to efficiently use the SSD cache. We evaluate our implementation of VENU on a medium-sized cluster and show that it achieves 11% improvement in application completion times when 10% of the available storage is provided by SSDs.**

I. Introduction

MapReduce [7] and Hadoop [2] have become the de facto framework for large-scale data processing and analytics. This is mainly due to the ability of the framework to efficiently handle both large batch processing workloads, such as building search indexes, and short interactive jobs, such as ad hoc data mining queries. The key component enabling the big data applications is the underlying Hadoop Distributed File System (HDFS), which offers desirable scale-out ability without performance degradation and while ensuring data availability and fault tolerance.

A challenge faced by researchers and IT practitioners in sustaining Hadoop clusters is evolving the storage and I/O infrastructure to deal with the exponentially growing data volumes, and to do so in an economically viable fashion. This is non-trivial, especially as the network bandwidth provided by the cluster networking infrastructure is growing an order of magnitude faster than the I/O bandwidths of hard disk drives (HDDs) [22]. In a typical large-scale Hadoop deployment, the intra-rack and inter-rack network has a bandwidth $200\times$ and $400\times$ that of the disk bandwidth [22], respectively.

Solid-state drives (SSDs) can help mitigate the above performance gap. Recent research [14], [13], [9], [19] has shown that SSDs are a viable alternative to HDDs for Hadoop I/O. Moreover, the hot data is too large to fit in RAM and the cold data is too large to easily fit entirely in flash memory [9]. Thus, adding a flash tier can improve overall performance. This approach is promising, but introduces the challenge of effectively managing the distribution of data among different tiers and selecting a tier for servicing read/write requests with the goal of improving application efficiency. In this paper, we address this challenge and explore the design space of incorporating tiered storage in Hadoop.

A promising trend observed in recent analysis is the significant heterogeneity in HDFS I/O access patterns. GreenHDFS [15] observed a news server like access pattern in HDFS audit logs from Yahoo!, where recently created data service a majority of the data accesses compared to old data, and more than 60% of used storage remained untouched for at least one month (during the period of the analysis). Scarlet [1] analyzed job history logs from Bing production clusters and observed that 12% of the most popular files are accessed over ten times more than the bottom third of the data. Similarly, the characteristics of the intermediate I/O, such as the size of the data and impact of intermediate I/O latency on execution time, is not constant across all Hadoop applications and may also vary across different executions of the same application [16].

In this paper, we introduce VENU, a system that reduces the overall I/O latency and execution time of applications in Hadoop by introducing *application-aware storage management* in HDFS. VENU employs a tiered storage system with two tiers designed to better match the heterogeneous Hadoop I/O access patterns. The tiers include a *fast SSD tier* that aggregates the SSDs provisioned in each node, and a *secondary HDD tier* comprising of HDDs. We use the SSD tier as a cache in front of the HDD tier. To this end, the key contribution of VENU is observing the HDFS I/O access patterns and load popular files into the SSD tier using our popularity predictor.

A concern in employing SSDs as a cache is that such devices have limited erase cycles, and may affect the MTTF. We stress that incorporating SSDs to form a caching tier is not unique to our approach, and other state-of-the-art works [20], [24] have also purported the same. Moreover, numerous SSD optimization approaches are available [23], [6] to remedy this, which can be leveraged in VENU.

Specifically, this paper makes the following contributions:

- We design a popularity predictor that predicts popularity of HDFS files based on access pattern of the file.
- We realize enhancements for HDFS to support the SSD caching tier and support data prefetching between the proposed tiers.
- We design and implement VENU to track the usage characteristics and dynamically propose appropriate storage tiers for HDFS data.
- We validate the VENU design and techniques therein using experiments on a real deployment.

Evaluation of VENU using a medium-sized Hadoop cluster,

978-1-4799-5666-1/14/$31.00 ©2014 IEEE

shows an 11% speed-up in application completion times when the SSD tier accounted for only 10% of the total available storage.

II. Background

Hadoop offers an open-source implementation of the MapReduce framework that provides machine-independent programming at scale. A Hadoop cluster node consists of both compute processors and directly-attached storage. A small number of nodes (typically $12-24$ [3]) are grouped together and connected with a network switch to form a rack. One or more racks form the Hadoop cluster. Intra-rack network bandwidth in large deployments is typically 20 GB and the inter-rack is 40 GB [22].

The compute component is managed by the *JobTracker* component that accepts jobs from the users and also manages the compute nodes that each run a *TaskTracker*. Each job consists of several map and reduce functions specified by the programmer. Each TaskTracker has one or more map and reduce slots, and applications will have tens of hundreds of map and reduce tasks running on these slots.

The data management for a Hadoop cluster is provided by the Hadoop Distributed File System (HDFS). HDFS manages the persistent data associated with the Hadoop cluster such as the input and the output of applications. The main functions of HDFS are to ensure that tasks are provided with the needed data, and to protect against data loss due to failures. HDFS uses a *NameNode* component to manage worker components called *DataNodes* running on each cluster node. HDFS divides all stored files into fixed-size blocks (chunks) and distributes them across DataNodes in the cluster. Moreover, the system typically maintains three replicas of each data block, two placed within the same rack and one on a different rack. The replica placement policy distributes each data block across multiple racks to ensure fault tolerance against node and rack failure. For data retrieval, a list of DataNodes ordered with respect to network proximity to the application instance is obtained from the NameNode and the nearest replicas are used.

III. Design

In this section, we present the design of VENU and how we address the challenges faced therein.

A. VENU *Overview*

The motivation for the design of VENU is that the access rate and the number of accesses vary for each file in HDFS input data, leading to popularity skewness over a subset of files. The goal of VENU is to improve the overall read throughput and storage efficiency of Hadoop clusters. To this end, we foresee Hadoop clusters comprising nodes with attached SSDs. VENU divides the different storage types into tiers, i.e., HDD tier and SSD tier, and enables effective utilization of the SSD tier by using it as a cache. Figure 1 illustrates the main components of VENU and their interactions. The *Popularity Predictor* dynamically keeps track of the access counts of input data files in HDFS. Based on this information, the component periodically makes a prediction for the popularity of each file in the upcoming interval and instructs HDFS to move the popular files into the SSD tier if needed.

We make the following choices in VENU design. First, we consider prefetching at file granularity because Hadoop jobs access entire files [1], [18]. Thus, prefetching at the block level is detrimental to the read performance of files as the blocks left behind in the slow tier will become the bottleneck. Second, we assume that the target resources are under load with large working sets — as is the case in production Hadoop clusters — thus any file system level caching has negligible effect, and the majority of reads are serviced from the storage layer. Third, all the nodes in the cluster contain an SSD. Finally, VENU is designed for the standard Hadoop deployment where all nodes contribute storage and compute resources.

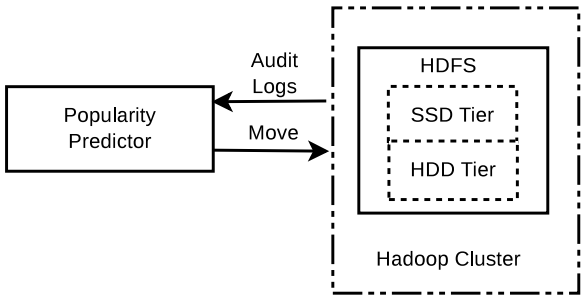

Fig. 1. VENU architecture overview.

B. Enabling Technology: *hat*S

hat*S* [16] is an enhancement to HDFS that provides heterogeneity-aware tiered storage in Hadoop. hat*S* logically groups all storage devices of the same type across the nodes into an associated "tier." A deployment has as many tiers as the different type of storage devices used, and a node with multiple types of devices is part of multiple tiers. For instance, if a deployment consists of nodes with an SSD and an HDD, all the SSDs across the deployment will become part of a SSD tier, and similarly all the HDDs will form the HDD tier. By managing the tiers individually, hat*S* is able to capture the heterogeneity in hardware and exploit it to achieve high I/O performance. While HDFS considers only network-aware data placement and retrieval policies, hat*S* proposes additional policies to replicate data across tiers in a heterogeneity-aware fashion. This enhances the utilization of the high-performance storage devices by efficiently forwarding a greater number of I/O requests to the faster tier, thus improving overall I/O performance.

We leverage hat*S* to provide an enhanced HDFS component for the system. We employ the default Hadoop data placement policy but ensure that by default the data is placed only in the HDD tier. Also, in order to avoid network contention during accesses and tolerance against node failure, we ensure that whenever a block is moved to the SSD tier, the block is not replicated at the node that already stores the block. The data retrieval policy that we employ is to always access the data from the fastest available tier. Moreover, we also provide APIs to be used by the Popularity Predictor to enable data movement between tiers.

C. Popularity Predictor

The Popularity Predictor uses the HDFS audit log information to determine the popularity of a file in HDFS in order to proactively fetch the popular data from the HDD tier into the persistent region of the SSD tier. This is an effective approach for identifying hot data as also shown in our previous work [18]. The popularity predictor uses Algorithm 1 to analyze the access patterns of each file after periodic time intervals and predict the file's expected popularity value for the next interval. The length of each interval is called Reference Time (RT). The choice of RT is critical; a very large RT can result in a stale SSD cache tier, whereas a small RT can increase network traffic. Previous works [1], [18] suggests that an RT between 12 and 24 hours is sufficient.

Input : HDFS audit logs
Output: HDFS files with their new popularity based on the access count.

F *is the set of files in the file system*;
foreach *access* $i+1$ *to the file* $f \in F$ *in* RT **do**
  if i==*0* **then**
    $P_{i+1}(f) \leftarrow AVG(P)$;
  end
  else
    $P_{i+1}(f) \leftarrow P_i(f) + 1$;
  end
  $IP = IP + P_{i+1}(f) - P_i(f)$;
end
foreach *deletion of the file* f *in* F **do**
  $IP = IP + AVG(P) - P(f)$;
end
$MIP \leftarrow \frac{IP}{size(F)}$;
foreach *file* f *in* F **do**
  $P_i(f) \leftarrow P_i(f) - MIP$;
  where i is the most recent access to the file f.
  $P_{predicted}(f) \leftarrow P_i(f) + (P(f) - P_i(f))$;
  $P(f) \leftarrow P_i(f)$;
end

Algorithm 1: Algorithm used by the Popularity Predictor.

When a file is created, its popularity $P_1(f)$ is initialized to average file popularity observed in the system, $AVG(P)$. For each access to the file, the popularity of the file is increased by one. Similarly, whenever a file is deleted, the popularity of other files is modified based on the popularity of the deleted file. When a popular file is deleted, the popularity of other files in the system increases. Conversely, when an unpopular file is deleted, the popularity of other files decreases. After the accesses of all files in the RT are processed, the popularity value $P_i(f)$ of a file f for the most recent access i is decreased by the mean increase in the popularity of the file f during that RT. This is done to makes sure that the popularity of the file $P(f)$ does not grow arbitrarily. The mean increase in popularity is a fraction of increase in popularity, IP, during RT over F, the set of all files in the system. Finally, we compute the popularity of the file for the next RT by linear extrapolation. The files in HDFS are sorted in the order of their predicted popularity, and popular files are prefetched into the SSD cache tier, until it is full.

D. Discussion

VENU can co-exist with the master component of each cluster or in a separate node. The computational overhead of VENU is negligible. The Popularity Predictor predicts the popularity of the files by linearly processing the file system audit logs. However, the prediction is done every RT, i.e., 12 to 24 hours, amortizing the associated overhead.

Hadoop performance is sensitive to network bandwidth, particularly during the shuffle phase that involves moving large amounts of data across the network. The Performance Predictor rearranges the data during every RT. This entails additional network overhead. To balance the bandwidth consumption, HDFS employs multi-location replication [1], where the data to be moved across tiers is read from multiple sources thereby spreading the traffic across nodes.

Data movement across tiers during the network-intensive shuffle phase may adversely affect the performance of the jobs in progress. To remedy this, the Performance Predictor's data rearrangement is made a low priority background process that yields to the shuffle phase to avoid negative performance impact.

IV. Evaluation

In this section, we present the evaluation of VENU using both a real deployment on a medium-scale cluster and simulations. We first study the characteristics of 10 representative Hadoop applications on SSD and HDD storage configurations. Next, we evaluate the impact of our HDFS enhancements, performance prediction and adaptive placement. Finally, we compare the overall performance of VENU against the extant application-oblivious storage placement strategy.

A. Experimental Setup

Our testbed consists of a master node and 8 worker nodes. Each node has two 2.8 GHz quad-core Intel Xeon processors, 8 GB of RAM, and one SATA HDD. The HDDs are 500 GB 7200 RPM Seagate Barracuda ES.2 drives. In addition to HDDs, each worker node is provisioned with an OCZ RevoDrive series PCIe 128 GB SSD. Table I shows the performance specifications of these storage devices. In our setup, all DataNodes contribute to both the SSD tier and the HDD tier. The nodes are connected using a dedicated 10 Gbps InfiniBand switch. Each worker node is configured with six map slots and two reduce slots so as to use all of the available cores on the node. The benchmark applications are mostly map intensive, so there are more map slots than reduce slots. The master node runs both the Hadoop JobTracker and NameNode for all the experiments, and all the worker nodes contribute to both TaskTracker and DataNode. The replication factor is fixed at the default of three, and the block size used is 64 MB.

TABLE I
SPECIFICATIONS OF DIFFERENT STORAGE DEVICES USED IN OUR TESTS.

Device Type	Write BW MB/s	Read BW MB/s	IOPS	# of devices
PCIe SSD	245	533	70k	3
HDD	46	61	3.5k	27

TABLE II
REPRESENTATIVE HADOOP APPLICATIONS USED IN OUR STUDY.

Application	Map		Reduce	Number	
	Input	Output	Output	Mapper	Reducer
NutchIndex	1.5 GB	2.8 GB	1 GB	1	81
Bayes	128 MB	256 KB	4.5 GB	16	1
Kmeans	1 GB	64 KB	1 GB	20	1
Hive-bench	5 GB	3.2 GB	256 MB	8	16
PageRank	128 MB	1 GB	12.5 MB	16	8
Sort	3 GB	11.5 GB	3 GB	64	8
TeraGen	–	–	15 GB	16	0
TeraSort	15 GB	15 GB	15 GB	249	8
WordCount	12 GB	30 GB	12 KB	102	8

TABLE III
DIFFERENT STORAGE CONFIGURATIONS USED IN OUR STATIC PROFILING.

Configuration	HDFS data	Intermediate data
Config-1	HDD	HDD
Config-2	SSD	SSD

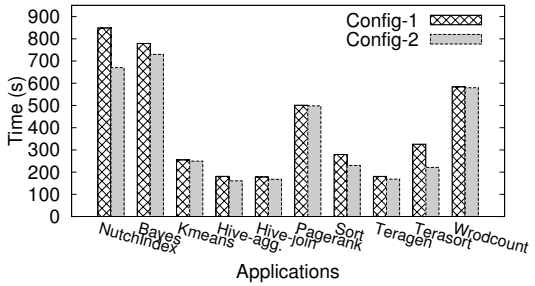

Fig. 2. Application completion times under *Config-1* and *Config-2*.

B. Benchmark Applications

We have used 10 applications from the well-known Hadoop HiBench Benchmark Suite [12] in our study. These applications are representative of batch processing jobs, iterative jobs and interactive querying jobs. Table II lists the applications, and for each summarizes parameters such as the input and output data size, and the number of mappers and reducers.

C. Observing the Effect of Storage on Performance

In our first set of experiments, we analyze the performance of our benchmark applications under four different storage configurations as shown in Table III. The input parameters of the applications are specified in Table II. The results discussed below are average of five executions; the standard deviation across the executions was observed to be negligible.

Figure 2 shows the performance of our test applications under storage configurations *Config-1* and *Config-2*. We observe that under *Config-2*, the jobs perform 12% faster on average over *Config-1*. On closer examination, we find that *NutchIndex*, *Bayes*, *Sort* and *TeraSort* experience an average performance improvement of 19.7%, while the other applications only see an average performance improvement of 2.9%. This validates that all applications do not benefit uniformly from the exclusive use of the SSD tier. Moreover, by monitoring the disk accesses we find that *NutchIndex*, *Bayes*, *Sort* and *TeraSort* contribute 60% of the total disk usage under *Config-2*, while all the other applications combined only contribute 40%. Therefore, if we were to place the input and intermediate data for only these four applications in the SSD tier and use the HDD tier for all other applications, we will be able to achieve an average performance improvement of 11% over *Config-1*.

D. HDFS Enhancement

In our next experiment, we evaluate how our HDFS enhancements from hat*S* impact VENU. To validate our placement and retrieval policy, we first ran *TeraGen* to generate 10 GB (160 blocks) of data distributed across different nodes. We parsed the HDFS logs to determine the placement policy and found that the distribution of blocks was similar to that of the default HDFS policy. However, more than half of the blocks under the default policy were spilled into the SSD tier, whereas under the VENU placement policy all blocks were located only in the HDD tier. We prefetched the data into the SSD tier and observed that the additional cost of prefetching was negligible (similar to time taken by the creation of the file). We do not quantify the cost of prefetching into a faster tier as such techniques have been evaluated extensively in prior work [16]. To validate the retrieval policy, we ran *Grep* over the data set. Figure 4 shows the tier locality of the accessed data. In the default policy, more than 68% of the data accessed is local data, while only 39% of the data accesses are for the SSD tier. In the proposed policy, all the data is accessed from the SSD tier, while 24% of the data is local to the node accessing it. We further analyzed the effect of these policies on performance using the *DFSIOE* benchmark. Figure 3 shows the overall I/O throughput for each of the map tasks, as well as the average I/O rate across all map tasks. We observe that both the default and the VENU policies have similar write performance, while VENU policies show 100% improvement in the read throughput and 47% improvement in the average I/O rate for reads.

E. Performance Analysis of VENU

In the next set of experiments, we evaluate the overall performance of VENU through whole-system simulation.

1) Trace Driven Simulation: For our simulation, we replayed Facebook-like traces synthetically generated by sampling historical MapReduce cluster traces. These are representative of traces generated from a Facebook cluster from October 2010 for a period of one and half months and are provided by Chen et. al. [5]. The available workload is one day in duration and contains 24 historical trace samples, each of 1 hour long.

The traces contain the record of accesses to approximately 18,000 files (along with filenames) constituting a total input size of 1548 TB. The size of the shuffle data generated is

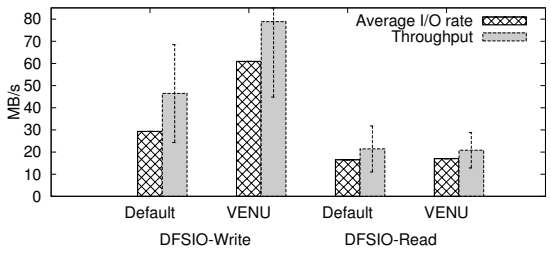

Fig. 3. Overall I/O throughput and average I/O rate per map task in *TestDFSIOE* benchmarks under default policy and VENU.

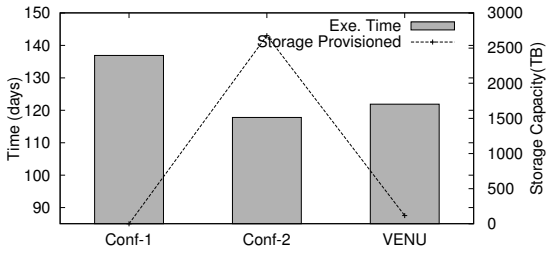

Fig. 5. Effectivenes of the Performance Predictor with increasing SSD capacity.

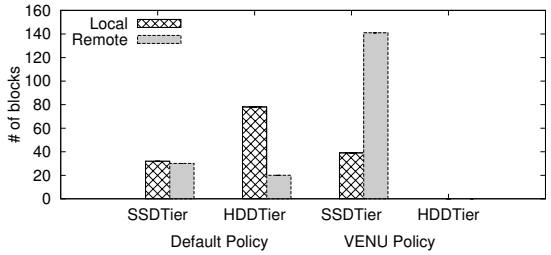

Fig. 4. The breakdown of reads based on tier and network proximity observed under default policy and VENU.

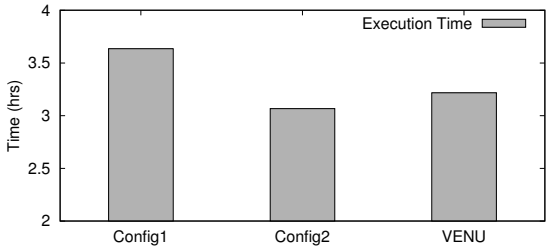

Fig. 6. Observed performance benefit on a real testbed.

375 TB and the output data generated is 755 TB. 55% of files are accessed more than once while the other 45% are accessed only once. The highest number of accesses to any file is 721. These files are accessed by 25,500 jobs, with 1075 jobs accessing the files every hour.

In our simulation we set RT = 1 hour, i.e., the performance predictor ran once every hour, generating a new set of popularity predictions and moving the most popular files to the SSD tier. This created a new HDFS layout on every interval. Our simulation incorporates various factors including the impact of replication, contentions while accessing a data file, and advantages of distribution of data along with those observed in Section IV-C. Figure 5 shows the completion time across the four fixed storage configurations from Table III and VENU. We see that although VENU is 5% slower than *Config-2*, VENU uses 5.5× less storage than that under *Config-2*.

Figure 5 shows the completion time across the four fixed storage configurations from Table III and VENU. We observe that *Config-2* is the fastest, 14% and 3% better compared to *Config-4* and *Config-3*, respectively. Whereas VENU is 10% and 6% faster than *Config-1* and *Config-3*, respectively. We see that although VENU is 5% slower than *Config-2*, VENU uses 5.5× less storage than that under *Config-2*.

2) Implementation Test: In order to validate the results from our simulation, we perform tests on a real mid-sized cluster described in Section IV-A. Our workload is generated from traces described in Section IV-E1 and a slice of this workload (lasting for approximately 3 hours) is executed. We replace the jobs in the trace with the benchmark applications (Table II). We note that our implementation results do not include the Performance Predictor component, as the length workload is too small to observe any significant advantages. For the same reason, we also limit our SSD capacity to be 10% of the available storage. VENU existed in an independent node (so as to avoid influence of Predictor's computations on the running job) and communicated with the master node to modify the storage configurations. Figure 6 shows the observed results. We observe that VENU is 11% faster than *Config-1* and 6% slower than *Config-2*, while utilizing 63% less SSD storage capacity than *Config-2*.

V. Related Work

Several recent projects [20], [24], [8], [21] focus on tiered storage for general purpose enterprise computing, mainly due to ease of management and business value. These systems typically employ SSD based caching atop disks that forms the bulk of storage substrate. This is done in order to get higher I/O rates and cost per performance [13] from SSDs while achieving optimal cost per capacity [13] from HDDs. While these works are complimentary to ours, in VENU we aim to provide similar data management solutions for the Hadoop ecosystem based on the unique characteristics therein.

The impact of SSDs on the performance of HDFS has been studied in many contexts in recent works. Borthakur et. al. [4] studied the impact of using SSDs in HDFS for supporting Apache HBase [10]. Harter et al. [9] extend this work to simulate the use of SSDs with several caching, logging and structural enhancements and observed 3.5× reduction in latency. Few recent works [14], [13], [9], [19] have also began to focus on the use of SSDs as an effective storage alternative to store intermediate Hadoop data for improving performance.

Our own hat*S* [16], ϕSched [17], and HDFS Issue # 2832 [11] both call for enabling support for heterogeneous storage devices in HDFS. They propose a re-design of HDFS into a multi-tiered storage system that seamlessly integrates

heterogeneous storage technologies such as HDDs and SSDs into HDFS and propose data placement and retrieval policies in order to exploit different storage devices. While these enhancements have introduced ways for HDDs and SSDs to co-exist in HDFS, they do not provide a viable solution for managing data movement between HDDs and SSDs. VENU is unique in its ability to understand and exploit the characteristics of the Hadoop applications and propose appropriate storage policy for both intermediate data and HDFS data across tiers based on their I/O characteristics and usage patterns. AptStore [18] and Scarlett [1] have proposed usage pattern-based replication schemes for Hadoop, but the suggested heuristics have not been shown to apply to SSD based systems.

VI. Conclusions

We have presented the design and implementation of VENU, a dynamic data management tool for Hadoop. VENU aims to improve overall I/O throughput via effective use of SSDs as a cache, not for all data, but for only the workloads that are expected to benefit from SSDs. We observe that managing all Hadoop data in a uniform manner results in increased storage overhead or reduced read throughput. We exploits the heterogeneity in access patterns to achieve overall improvement in I/O throughput by storing the popular files in SSD tier. We evaluate our implementation of VENU on a medium-sized cluster and show that VENU achieves 11% improvement in application completion times when 10% of its storage provided by SSDs.

Acknowledgment

This work is sponsored in part by the NSF under the CNS1422788 and CNS1405697 grants.

References

[1] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris. Scarlett: Coping with skewed content popularity in mapreduce clusters. In *Proc. ACM EuroSys*, 2011.

[2] Apache Software Foundation. Hadoop, 2011. http://hadoop.apache.org/core/.

[3] D. Borthakur. Facebook has the world's largest hadoop cluster!, 2010. http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html.

[4] D. Borthakur. Hadoop and solid state drives, 2012. http://hadoopblog.blogspot.com/.

[5] F. Chen, D. A. Koufaty, and X. Zhang. Hystor: making the best use of solid state drives in high performance storage systems. In *Proc. ACM SC*, 2011.

[6] S. S. Chu and C. V. Ho. Self-recovering erase scheme to enhance flash memory endurance, Mar. 21 1995. US Patent 5,400,286.

[7] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] S. Feldman and R. L. Villars. The information lifecycle management imperative. *IDC White Paper*, 2006.

[9] T. Harter, D. Borthakur, S. Dong, A. S. Aiyer, L. Tang, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Analysis of hdfs under hbase: a facebook messages case study. In *Proc. FAST*, 2014.

[10] A. HBase. The apache hadoop project.

[11] HDFS-2832. Enable support for heterogeneous storages in hdfs, 2012. https://issues.apache.org/jira/browse/HDFS-2832.

[12] S. Huang, J. Huang, Y. Liu, L. Yi, and J. Dai. Hibench: A representative and comprehensive hadoop benchmark suite. In *Proc. ICDE Workshops*, 2010.

[13] K. Kambatla and Y. Chen. The truth about mapreduce performance on ssds. In *Proc. USENIX LISA*, 2014.

[14] S.-H. Kang, D.-H. Koo, W.-H. Kang, and S.-W. Lee. A case for flash memory ssd in hadoop applications. In *IJCA*, volume 6, 2013.

[15] R. Kaushik, M. Bhandarkar, and K. Nahrstedt. Evaluation and analysis of greenhdfs: A self-adaptive, energy-conserving variant of the hadoop distributed file system. In *Proc. IEEE CloudCom*, 2010.

[16] K. Krish, A. Anwar, and A. R. Butt. hats: A heterogeneity-aware tiered storage for hadoop. 2014.

[17] K. Krish, A. Anwar, and A. R. Butt. ϕsched: A heterogeneity-aware hadoop workflow scheduler. 2014.

[18] K. Krish, A. Khasymski, A. R. Butt, S. Tiwari, and M. Bhandarkar. Aptstore: Dynamic storage management for hadoop. In *Proc. IEEE CloudCom*, 2013.

[19] S. Moon, J. Lee, and Y.-s. Kee. Introducing ssds to the hadoop mapreduce framework. In *Proc. IEEE cloud*, 2014.

[20] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating server storage to ssds: analysis of tradeoffs. In *Proc. ACM EuroSys*, 2009.

[21] M. Peterson. Ilm and tiered storage. *Storage Networking Industry Association*, 2006.

[22] Pivotal. Analytics workbench, 2010. http://www.analyticsworkbench.com/.

[23] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber. Extending ssd lifetimes with disk-based write caches. In *Proc. USENIX FAST*, 2010.

[24] G. Zhang, L. Chiu, C. Dickey, L. Liu, P. Muench, and S. Seshadri. Automated lookahead data migration in ssd-enabled multi-tiered storage systems. In *Proc. IEEE MSST*, 2010.