Using Working Set Reorganization to Manage Storage Systems with Hard and Solid State Disks

Junjie Chen, Jialin Liu
Department of Computer Science
Texas Tech University
Lubbock, USA
Email: {junjie.chen, jaln.liu}@ttu.edu

Philip C. Roth
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA
Email: rothpc@ornl.gov

Yong Chen
Department of Computer Science
Texas Tech University
Lubbock, USA
Email: yong.chen@ttu.edu

***Abstract*—Scientific applications from many problem domains produce and/or access large volumes of data. To support these applications, designers of high-end computing (HEC) systems have greatly increased the capacity of storage systems in recent years. However, because hard disk drives (HDDs) are still the dominant storage device used in HEC storage systems, and because HDD performance has not improved as quickly as the capacity, it can be challenging to deploy a storage system that provides both extreme capacity and extreme performance at a reasonable cost. Solid State Drives (SSDs) are a promising high-bandwidth and low-latency alternative to HDDs for HEC storage systems, but they too have deficiencies: small capacity, limited write cycles, and high cost when compared to HDDs. Because of their complementary characteristics, storage system designers are beginning to consider heterogeneous storage system designs that include both HDDs and SSDs. However, managing the workload so as to take advantage of the strengths of each type of storage device while controlling overhead is a major challenge. In this study, we propose a novel approach for managing a heterogeneous storage system called the *Working Set-based Reorganization Scheme (WS-ROS)*. With WS-ROS, applications write to both HDDs and SSDs using all the available storage system bandwidth. Later, a background process reorganizes the data so as to place the data most likely to be read on SSDs while relegating the data most likely to be written and the data not likely to be accessed onto the slower but higher-capacity HDDs. For our evaluation workloads, the WS-ROS approach provided a $3\times$ to $10\times$ performance improvement compared to a heterogeneous storage system without a working set-based data reorganization scheme, suggesting the value of lazy reorganization of data based on data access working sets.**

***Keywords*-High-end computing, data-intensive computing, storage, parallel file systems, Solid State Drives**

I. Introduction

Scientific applications from many problem domains produce and/or access extremely large volumes of data. For instance, climate models, combustion models, and astrophysics models manipulate 100s of terabytes or even a petabyte of data within a single application run [18]. Scientific applications targeting future exascale platforms may operate on tens to hundreds of petabytes [6]. By 2010, Facebook was storing several petabytes of data (compressed), and loading tens of terabytes more per day (also compressed) [19]. The storage on systems supporting these applications must provide extreme capacity, but also exceptional performance.

Most on-line storage systems for high-end computing (HEC) systems use hard disk drives (HDDs) as the storage device. HDDs are inexpensive, have large capacity per device, and can provide impressive performance for sequential data accesses. However, the bandwidth and latency of HDDs have not increased as quickly as their capacity, and thus it can be challenging to deploy a storage system that provides both extreme capacity and extreme performance at a reasonable cost. Solid state disks (SSDs) that use non-volatile flash memory instead of rotating magnetic media have emerged as a promising alternative to HDDs for HEC storage systems [2, 3, 11, 21–23]. SSDs are free of mechanical components and do not suffer the seek time delays and rotational latencies that negatively impact HDD performance, so the data access performance of SSDs is usually much better than HDDs, especially for random accesses [1, 8, 14]. However, SSDs have lower capacity and a higher cost-to-capacity ratio compared to HDDs. Also, because of the storage technology used in SSDs, they allow only a limited number of write cycles (although recent SSD devices have increased this limit compared to early SSD devices). Because the workloads for many applications run on HEC systems are write intensive and generate large volumes of data, these disadvantages of SSDs keep them from replacing HDDs entirely as the building block of HEC storage systems. Instead, designers are looking for ways to combine HDDs and SSDs in heterogeneous storage systems that take advantage of the strengths of both types of devices. A major challenge for such designs, however, is how to manage the mix of devices to provide exceptional capacity and performance while controlling cost and management overhead.

To address this challenge, we propose a new approach for managing storage systems that combine HDDs and SSDs. Our approach, called the *Working Set-based Reorganization Scheme (WS-ROS)*, leverages the merits of HDDs and SSDs to provide a highly efficient storage system for HEC workloads. Under our approach, applications write to the storage system using all available bandwidth of both HDDs and SSDs so as to complete the write operation as quickly as possible, similar to the idea of a burst buffer [13]. Later, a background process reorganizes the data when devices are idle. Unlike burst buffers, WS-ROS uses a working set model to guide data placement during reorganization. WS-ROS maintains separate regions for read and write requests. Using a sliding window, the approach tracks data access history in locality tables. Read accesses and write accesses are tracked independently. When triggered by the WS-ROS scheduling algorithm, the background data reorganization process uses the access history information to redistribute data between the available HDD and SSD devices, based on their likelihood of being accessed in the future and based on whether that access is likely to be a read or a write.

Our study makes three main contributions:

- It characterizes the motivation and requirements for a well-managed storage system that combines HDDs and SSDs in support of HEC workloads (Section II);
- It proposes and details the Working Set-based Reorganization Scheme (WS-ROS) to manage such a heterogeneous storage system (Section III); and
- It evaluates the proposed approach using a prototype WS-ROS implementation and simulation with traces taken from real applications (Section IV). For the example workloads we tested, WS-ROS achieved a $3\times$ to $10\times$ speedup over an approach that did not use a working set-based data reorganization approach to managing a heterogeneous storage system.

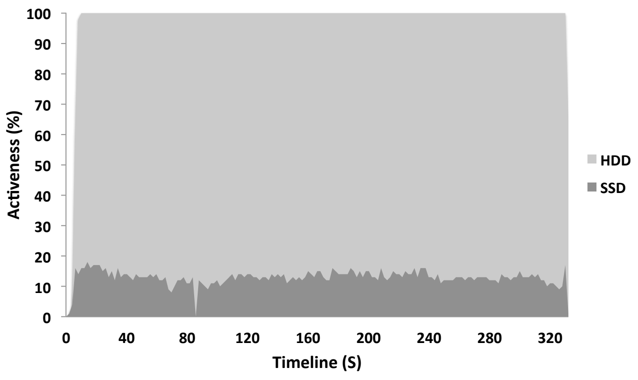
II. Motivation

SSDs and HDDs have complementary strengths, making it attractive to consider augmenting an HDD-based storage system with SSDs. In this part of our study, we sought to understand how much performance benefit we would observe by integrating SSDs with HDDs in an HEC storage system.

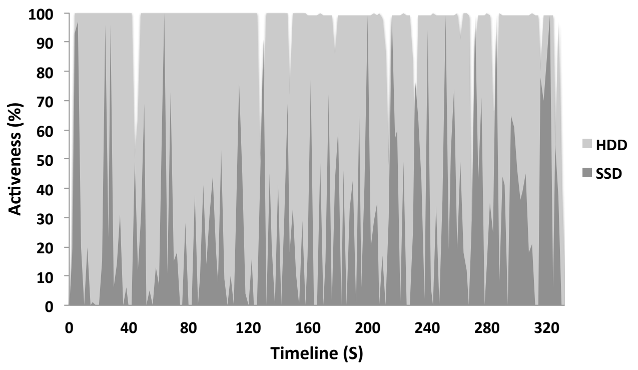
A. Motivating Observations

In previous work, we investigated the impact of different SSD and HDD deployment strategies on HEC storage system performance [5]. On our test platform, we expected the SSD to provide a speedup of over $5\times$ compared to an HDD, but the performance benefits were much lower than expected—only around $1.3\times \sim 2.5\times$.

We investigated the reason for this discrepancy by evaluating the *activeness* of HDDs and SSDs when running an I/O benchmark. We define activeness as the amount of time a storage device spends servicing read/write requests. Thus, activeness is the opposite of idleness. Our test system was a four node cluster configured with two compute nodes and two storage nodes. The storage nodes provided a PVFS2 [12] file system. As a workload, we used the IOR benchmark [10] to access 16 GB data total. We used *nmon* [15], a system administration, tuning, and benchmarking tool for Linux systems, to measure bandwidth and device activeness as the IOR benchmark ran. As expected, we discovered there was a clear performance distinction between writes and reads to the SSD. The read bandwidth was $5\times \sim 8.5\times$ better with SSDs than with HDDs, but the write bandwidth was only $1.2\times \sim 1.8\times$ better. In addition to bandwidth, we sampled device activeness every two seconds over an interval of approximately 330 seconds. The activeness results are shown in Figure 1, separated



(a) Reads



(b) Writes

Fig. 1. Activeness of HDD and SSD

by access type. We made two observations based on these activeness measurements. First, the system's SSDs were active much less frequently than the HDDs for both reads and writes. In fact, when servicing read requests, the SSDs never exceeded 20% activeness. Second, the activeness of the SSDs was less than the HDDs when servicing write requests also, but the gap was much smaller than when servicing read requests. We attribute this difference in SSD activeness behavior as being due to a lack of device-specific support in the upper layers of the file system, which does data layout and distribution using a round robin or similar strategy without awareness of the distinct capabilities of the underlying storage devices.

Our performance and activeness measurements suggest a storage management approach that primarily uses SSDs to service read requests and HDDs to service write requests. A straightforward way to implement this approach is to adjust the data placement by directing more data to the SSDs in the file system, such as by increasing the stripe unit size so that it allocates more data to the SSDs. However, such a straightforward solution has two disadvantages: 1) distributing more data to the SSDs could further amplify the capacity imbalance between SSDs and HDDs, leaving the HDDs underutilized and eventually making the SSDs a performance bottleneck; 2) writing more data to the SSDs causes more frequent Program/Erasure operations on the SSDs' flash memory which can significantly decrease SSD lifetime and thus decrease the overall reliability of the storage system.

B. Motivating Analysis

To further understand the potential performance benefit of using SSDs to service read requests in a heterogeneous storage system, we constructed a performance model for read requests to a simple storage system with one SSD and one HDD. We assume that compute nodes can read from the HDD and SSD simultaneously. Our modeled workload consists of n read requests, each of size s. The read latency of the SSD is l, and the read latency of the HDD is γl (i.e., the SSD is $\gamma\times$ faster than the HDD for reads). In the baseline case where data is distributed evenly between the HDD and SSD, $n/2$ of the requests are serviced by the HDD and $n/2$ are serviced by the SSD. Thus the time required to perform all read requests is given by Equation 1, which shows the total read time is dominated by the time required to read the data from the HDD. In contrast, if the data were redistributed to take the SSD's superior read performance into account, γ times as much data would be read from the SSD as is read from the HDD. In this case, the time required to complete all reads is given by Equation 2. If we assume $\gamma = 5$, the speedup over the baseline scenario is 3.

$$T = \max\{\frac{n \times s \times l}{2}, \frac{n \times s \times \gamma l}{2}\} = \frac{\gamma lns}{2} \tag{1}$$

$$T = \max\{\frac{\gamma n \times s \times l}{\gamma + 1}, \frac{n \times s \times \gamma l}{\gamma + 1}\} = \frac{\gamma lns}{\gamma + 1} \tag{2}$$

This simple model ignores several realities, such as the fact that current SSDs have smaller capacities than HDDs and that there is a cost to reorganizing data (both in terms of time and on the number of writes to the SSD device). Nevertheless, both the observations and the analysis suggest the promise of an approach like WS-ROS that reorganize data so that data likely to be read is read from the SSD.

III. Data Structures and Algorithms

WS-ROS is a management approach for heterogeneous storage systems that reorganizes data amongst the available SSDs and HDDs using a working set model. We have three goals for WS-ROS: 1) to make SSDs service more of the read traffic to better utilize the SSDs' superior read performance compared to HDDs; 2) to use the capacity of both SSDs and HDDs efficiently; 3) to increase the write endurance of SSDs while not impacting the write performance of HEC applications. In this section, we provide an overview of the WS-ROS data structures and workflow, then detail its working set model, its data reorganization algorithm, and our prototype implementation.

A. Overview of Data Structures and Workflow

Figure 2 shows the WS-ROS workflow and the primary data structures supporting that workflow. When presented with a workload, WS-ROS tracks read requests and write requests independently within a sliding time window using separate *locality tables* associated with a *read region* and *write region*.

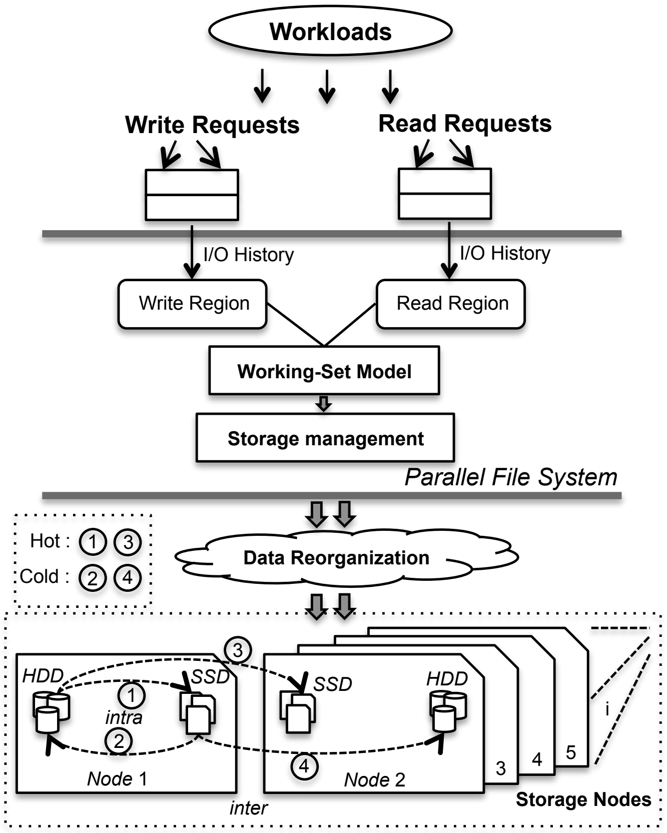


Fig. 2. WS-ROS Workflow

TABLE I
Locality Table in Working Set

Location	{loc1}	{loc2}	{loc3}	{loc4}	⋯
Hotness	1	5	3	4	⋯
Timestamp	T_1	T_2	T_3	T_4	⋯

In a parallel file system that uses the WS-ROS approach, the read region and write region are likely to be implemented in the file system's storage management module. An example locality table is shown in Table I. For each data access, WS-ROS adds the location of the data block and the time the access occurred to the appropriate locality table. (It also initializes a *hotness* counter for the location, which is described later in this section.) If the access is a new write (i.e., one to a location that is not being tracked), it is spread across the available SSDs and HDDs within the system's storage nodes to take advantage of all of the available write bandwidth. Thus write bandwidth of data such as an application checkpoint is not sacrificed by WS-ROS.

When the WS-ROS scheduling algorithm determines the storage system is idle, it uses the information in the locality tables to reorganize the data within the storage system. As shown in Figure 2, this reorganization may occur by moving data within a node (transfers 1 and 2 in the figure) or between nodes (transfers 3 and 4 in the figure). In general, WS-ROS moves data likely to be read onto SSDs (transfers 1 and 3) to improve read performance, and other data onto HDDs (transfers 2 and 4) to balance storage utilization and to ensure that the system's SSDs have sufficient free space to support reorganization. The cost of moving data between devices in distinct nodes is larger than the cost of moving data between devices within the same node.

B. Working Set Model

The *working set* is a concept for defining the set of recently and frequently referenced data, e.g., by the memory management subsystem of an operating system that supports demand paged virtual memory. We use the idea of a working set of I/O accesses to guide the WS-ROS data reorganization approach.

1) Working Set Model in WS-ROS: Let $d(t)$ denote the location of the data block accessed at a time t, and τ be the width of the WS-ROS sliding window (i.e., τ is the size of a time interval). WS-ROS uses the timestamps associated with data accesses in its locality tables to determine which data blocks are within the working set for a given time t_0 and window size τ according to Equation 3.

$$ws(t_0, \tau) = \{d(t) \mid (t_0 - \tau) \leq t \leq t_0\} \qquad (3)$$

Note that this view of working set is process-independent. Although many processes may be accessing the same data block, this definition of working set is from the perspective of the storage system, which sees the sequence of requests aggregated from all processes.

In WS-ROS, the sliding window size τ is dynamically adjusted to strike a balance between working set completeness and the cost of maintaining the working set and using it to reorganize data. Additionally, in our prototype WS-ROS implementation, the sliding window size can be tuned by system administrators. We envision this working set model would be integrated into the parallel file system layer alongside with the read and write regions to support storage management using the WS-ROS approach.

2) Read Region: WS-ROS uses a *read region* to track and manage the workload's read requests. The read region is used by the WS-ROS to construct the working set model. The read region tracks the read requests using a locality table, using this information to construct the working set for read requests. When a read request is seen by WS-ROS, the access information including file name, offset, and request size (translated to storage device ID and a sequence of block IDs) are recorded in the read region's locality table. When a block that is already present in the locality table is accessed, the hotness counter associated with the block is incremented. These counters are cleared as the sliding window advances through time. WS-ROS removes a block's information from the locality based on the value of the hotness counter as described in Section III-C.

WS-ROS uses two thresholds to determine whether a data block is considered to be hot. When a block's data hotness counter is higher than *threshold-H*, the block is considered hot, and if the hotness is lower than *threshold-L*, the block is considered cold. Both hot blocks and cold blocks may be moved during a data reorganization. However, any data blocks with a hotness counter between *threshold-H* and *threshold-L* are considered lukewarm and are not moved. These thresholds *threshold-H* and *threshold-L* are configurable by the system administrator, and thus can be set to tailor the behavior of WS-ROS data reorganization (e.g., to control WS-ROS overhead).

Two distinct read requests might read overlapping data. Overlapping requests can be classified into two categories: nested and non-nested. In both cases, WS-ROS divides overlapped requests into three sub-requests and treats the three sub-requests individually. So, the hotness counter for the sub-requests that appear in both original requests is incremented by two, whereas the hotness counter for sub-requests appearing in only one of the original requests is only incremented by one. To detect overlapping requests, our prototype WS-ROS implementation uses a binary search algorithm over data block information sorted by block address and considering the size of each accessed data block.

3) Write Region: The WS-ROS *write region* functions similarly to the read region—it also tracks write requests using a locality table and manages write requests. The two regions differ mainly in how WS-ROS uses the information they track. During data reogranization, the write region's information is used in two ways. First, blocks that are indicated to be hot within the write region and that are stored on an SSD are relocated to an HDD to reduce the number of program/erasure cycles performed by the SSD's flash memory. Second, blocks that are indicated to be cold within the write region and that are stored on an SSD are also relocated to an HDD to free SSD capacity for use by more frequently-read data.

C. WS-ROS Algorithms

WS-ROS uses two algorithms to track and manage read and write requests from a workload, and to perform data reorganization. Algorithm 1 indicates how a parallel file system's metadata server handles incoming I/O requests under the WS-ROS approach. In this paper, the algorithm is expressed from the perspective of the metadata server of the parallel file system simulator we used to evaluate WS-ROS (see Section IV). This simulator uses I/O request traces as input, and this form of the workload is reflected in the algorithm. For each I/O request it handles, the algorithm updates the locality tables and working set model in preparation for later data reorganization by the WS-ROS background process. It also handles the parallel file system's normal striping and scheduling tasks for a read or write request.

Algorithm 2 details how WS-ROS' background data reorganization process relocates data. Not shown is the procedure by which the parallel file system detects the file system is idle and invokes this reorganization algorithm. The algorithm first examines the read region's locality table and determines whether to redistribute the data based on each data block's hotness counter. Note that most data moves are done immediately, but cold data on an SSD is left in place but marked for reorganization. Later, if the SSD's free space falls below a low water mark, this data is moved to create more free space

```
Input: I/O requests in trace files.
for Each request in the input trace do
    Get the access records from trace files;
    if Write request then
        if file already exists then
            Get stripes from the metadata server through
            offset;
            Schedule the request;
            update the locality table in write region;
        else
            Create file on storage;
            Schedule the request with striping;
            Create an item in the locality table in the
            write region;
        end
    else
        Get stripes from the metadata server through
        offset;
        Retrieve data from storage;
        Update the locality table in the read region;
    end
end
```

Algorithm 1: WS-ROS I/O request handling algorithm

```
Input: Working set data with locality tables.
for Each item in the read region's locality table do
    if Hotness > Threshold-H then
        Check data block location;
        if Data located on HDD then
            Distribute data from HDD to SSD;
        end
    else
        if Hotness < Threshold-L then
            Delete the item from the locality table;
            if Data located on SSD then
                Distribute data from SSD to HDD
                through a delayed strategy;
            end
        end
    end
end
for Each item in the write region's locality table do
    if Hotness > Threshold-H then
        Check data block location;
        if Data located on SSD then
            Distribute data from SSD to HDD;
        end
    else
        if Hotness < Threshold-L then
            Delete the item from the locality table;
        end
    end
end
```

Algorithm 2: WS-ROS data reorganization algorithm

on the SSDs. Because cold data on an SSD is not likely to be accessed, this delayed strategy avoids paying the cost of transferring the data until it is actually necessary.

As noted earlier, the parallel file system implementing the WS-ROS approach triggers its data reorganization algorithm when the storage system is idle. After reorganizing its data, there is no reason to trigger the data reorganization algorithm again unless one of the following has happened: applications have written new data to the storage system; or, applications have issued I/O requests to the storage system in a pattern that is significantly different from the pattern that was observed before the data reorganization. So, after a data reorganization, the file system need not even consider triggering the reorganization algorithm again until it has begun receiving additional I/O requests. In a simple approach, the file system might simply set a flag after a data reorganization that indicates it should not trigger the reorganization algorithm even if the storage devices are idle. In this case, the WS-ROS handling of I/O requests by the metadata server would clear the flag the next time it received an I/O request. This strategy, although simple to implement, might result in the triggering of the data reorganzation too frequently for light I/O workloads that do not result in substantial changes to the data layout (e.g., with writes of new data) or working sets (e.g., with reads or writes to existing data).

There are more sophisticated approaches to avoiding triggering the WS-ROS reorganization algorithm that take more than just the next I/O request into account. If the incoming requests fall primarily into Case 1 (newly written data), the file system can use a tunable threshold to determine when enough new data has been written to the storage system's SSDs and HDDs to consider whether the previous data placement might no longer provide good performance. (Recall that writes of new data to the storage system are serviced by all available devices, so these new write I/O requests will result in data being written to the SSD devices as well as the HDDs.) Only if enough new data has been written will the parallel file system allow another data reorganization to occur.

If the new I/O requests fall primarily into Case 2, the file system must periodically examine the working set to determine if it has changed substantially since the last data reorganization. Only if it has changed substantially should WS-ROS consider whether to reorganize data, because a substantial change in working set would indicate that once-hot or once-cold data may no longer have that same status, and thus should be relocated for best performance. The metric used to determine when the working set has changed "enough" must be inexpensive to evaluate. Changes in the size of the working set might indicate such a change, as well as the number of hot and cold data blocks. In case simple metrics like this are not sufficient, more complicated metrics may be used such as noting whether the blocks that are the hottest and/or coldest have changed since the working set was last checked.

IV. Evaluation

To evaluate WS-ROS, we implemented the approach in a simulator that uses I/O request traces as input. For this

Name	Default Value
Number of SSDs	4
Number of HDDs	4
SSD capacity (GB)	64
HDD capacity (GB)	512
SSD read latency (s/GB)	0.1
SSD write latency (s/GB)	0.5
HDD read latency (s/GB)	1
HDD write latency (s/GB)	2
sliding window size (%)	10
stripe size (GB)	64
threshold-H	3
threshold-L	2
SSD capacity threshold (GB)	20

TABLE II
WS-ROS PARALLEL FILE SYSTEM SIMULATOR PARAMETERS

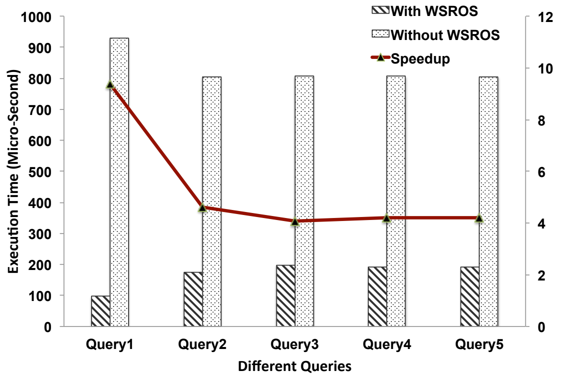
study, we used I/O request traces we collected from running several of the *Maryland Applications* [20], a collection of benchmarks and real applications used in previous I/O studies. Our simulator models a data distribution strategy similar to that used in PVFS2 [12], namely a 64KB default stripe size and a round-robin strategy for allocating new data to the available storage devices. We implemented a metadata server for our parallel file system simulator that manages the locality tables using Algorithm 1. In our simulator, the data block location information is maintained using a structure that includes the file handle, data offset, and request size. When our simulator determines the storage is idle, it triggers the WS-ROS algorithm (Algorithm 2) to reorganize data for better performance. If the WS-ROS algorithm moves data, the simulated metadata server is updated to reflect the data's new placement. To simplify our implementation for the evaluation, we used a very simple strategy to avoid running the data reorganization algorithm too frequently. For these experiments, we used a simple counter to indicate the maximum number of times the algorithm could run during the experiment. For most experiments, we set this counter to 1, so that the algorithm would only run once during the experiment. In Section IV-B3, our evaluation of the sensitivity of WS-ROS to the sliding window size includes consideration of the impact of performing the reorganization twice instead of once per run, but a more complete evaluation of the impact of running the data reorganization algorithm multiple times (perhaps using one of the sophisticated strategies described in Section III-C) is left for future work.

The simulator's parameters and their default values are shown in Table II.

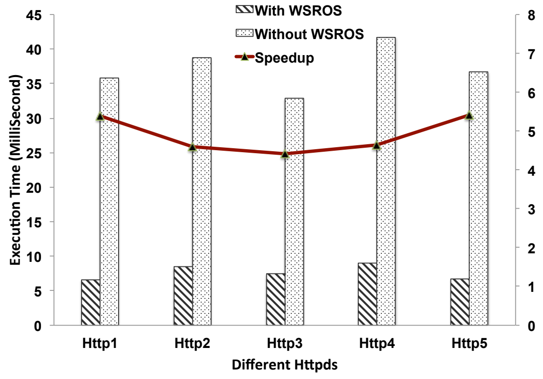
A. Evaluation Workload

For our evaluation workload, we collected I/O request traces from the following *Maryland Applications*:

- **DB2 Parallel Edition** [7] Five consecutive queries to a large relational database, including join, set, and aggregate operations on indexed and non-indexed relations. The database used is a commercial-grade parallel RDBMS from IBM containing 5.2 GB of data.
- **Parallel Web Server** [17] Approximately 1.5 million HTTP requests generated by four clients to multiple Apache servers, resulting in 36 GB of data.



(a) DB2 Parallel Edition



(b) Parallel Web Server

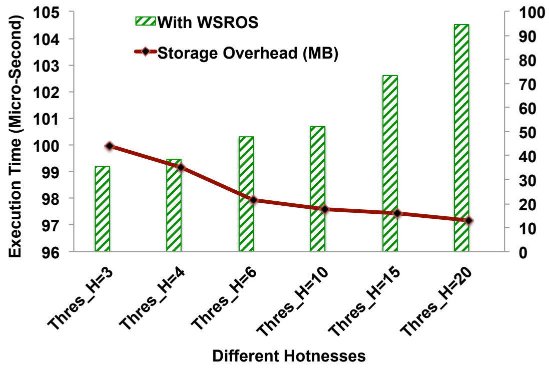
Fig. 3. Performance of heterogeneous storage system with WS-ROS

We used a simple trace capture utility to record each program's I/O calls and saved the call information to ASCII trace files. We ran these applications on a 16-node Dell PowerEdge Linux cluster. This cluster contains one PowerEdge R515 server and 15 PowerEdge R415 nodes, with a total of 32 processors and 128 cores. The PowerEdge R515 has two quad-core 2.6GHz AMD Opteron4130 processors, 8GB memory, and a RAID-5 disk array with 3TB storage capacity composed of 7200 RPM Near-Line SAS drives. Each PowerEdge R415 node has two quad-core 2.6GHz AMD Opteron 4130 processors, 4GB memory, a 500GB 7200RPM Near-Line SAS hard drive, and an OCZ Vertexz/RevoDrive or Crucial M4 SSD. The nodes are fully connected using a PowerConnect 2848 network switch with 42 1Gb Ethernet ports.

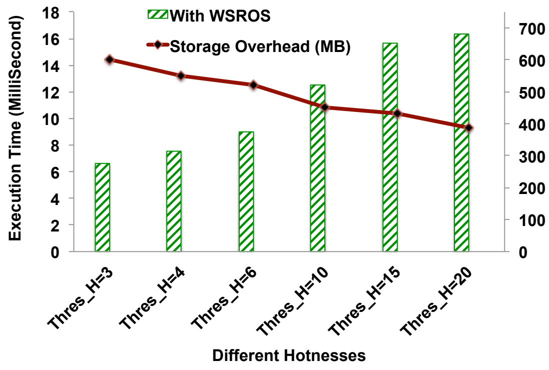
B. Results

To evaluate WS-ROS, we measured the performance of our simulated parallel file system with WS-ROS when serving the workloads described in the previous section, and compared this performance against that of a hybrid storage system that did not use a data reorganization scheme.

1) Performance Results: Figures 3(a) and 3(b)show the simulated execution time for the DB2 Parallel Edition and Parallel Web Server workloads, respectively. For these experiments, we used 3 for the value of the hotness threshold (i.e.,

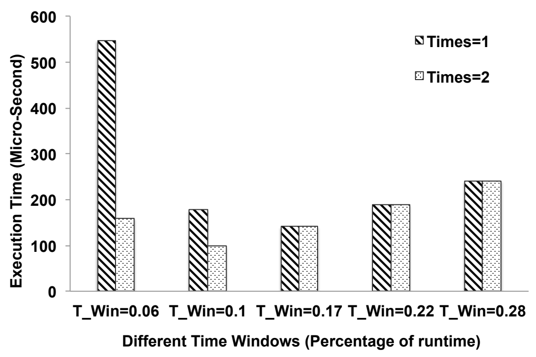


(a) DB2 Parallel Edition

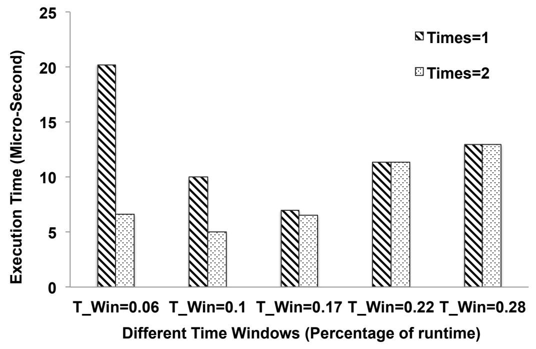


(b) Parallel Web Server

Fig. 4. WS-ROS sensitivity to hotness



(a) DB2 Parallel Edition



(b) Parallel Web Server

Fig. 5. WS-ROS overhead sensitivity to time window size

threshold-H from Section III). In this evaluation, we focus on the effect of the hotness threshold and not the coldness threshold (*threshold-L*). We use a sliding window size of $\tau = 0.1$ (expressed as a percentage of the total runtime of the application). The results shown in the figure are the average of two independent runs with the same configuration. The speedup lines shown in the figures indicate how much faster the application ran using the storage system with WS-ROS compared to the storage system without WS-ROS.

For These two experimental workloads, there is a substantial performance benefit to using a storage system with WS-ROS compared to a storage system without data reorganization. Also, although there were slight variations across the workloads, a similar performance improvement was observed for all of the test workloads.

2) Hotness Sensitivity and Overhead Analysis: To understand the sensitivity of WS-ROS performance and overhead to the hotness value used when reorganizing data, we measured both when running our three test workloads with our simulated WS-ROS-based parallel file system. When using large hotness values, WS-ROS is less likely to place data on SSDs. In this case, we expected to see relatively large run times and low data reorganization overhead. Conversely, when using small hotness values, WS-ROS is more likely to place data on SSDs but must pay the overhead cost of reorganizing the data. Figures 4(a) and 4(b), and show the the elapsed time and WS-ROS overhead when using a range of hotness values for DB2 Parallel Edition and Parallel Web Server, respectively. In the figures, we use the amount of data relocated to SSDs (in MB) as a measure of WS-ROS overhead.

These results support our expectations regarding the relationship between hotness, WS-ROS performance, and WS-ROS overhead. Each workload exhibited a similar pattern, with high hotness values giving poor overall performance but low overhead, and low hotness values giving the best performance at a higher overhead cost. The hotness value should be a tunable parameter so that system administrators can tailor the performance-to-overhead ratio for their particular workloads. Alternatively, a future version of WS-ROS may dynamically adjust the hotness value in an attempt to find the "sweet spot" that best balances performance benefit against data reorganization overhead.

3) Sensitivity to Sliding Window Size: We also considered the sensitivity of WS-ROS to the sliding time window size, τ. The sliding window size determines how much history should be considered when reorganizing data. Large sliding window sizes can provide better working set estimates, but require more time to construct from the WS-ROS locality tables. Thus we expect there to be a "sweet spot" where the cost of constructing the working set is balanced by the quality of the information in the working set with respect to predicting future data accesses. For these experiments, we also considered the impact of reorganizing the data more than once during the program's run. The results of our time window sensitivity experiments are shown in Figures 5(a) and 5(b) for

DB2 Parallel Edition and Parallel Web Server, respectively. In general, increasing the window size allows more hot data to be allocated to the system's SSDs. However, increasing the window size also increases WS-ROS overhead, and eventually this overhead outweighs the performance benefit of placing the hot data on SSDs. Indeed, for each test workload, we see that extreme values of the window size result in worse performance. The simulated file system with WS-ROS optimization provided similar performance for each of the three workloads, with a window size of approximately 0.1 providing the best performance (and thus the best trade-off between working set prediction quality and WS-ROS overhead).

The results in Figures 5(a), 5(b), and **??** also show that for very small window sizes, reorganizing the data multiple times provides better performance than only reorganizing once. For larger sliding window sizes, there is no significant difference in performance between reorganizing data once versus reorganizing twice. This is because small sliding window sizes do not allow WS-ROS to capture the entire working set, so the initial data reorganization is not optimal for the true data access pattern of the workload. After WS-ROS has seen more of the workload, its historical record of I/O accesses better captures the actual working set for the workload, thus a second reorganization provides much better performance. With larger window sizes, the initial data reorganization was performed using a much better estimate of the actual working set, and thus there is little benefit to further data reorganizations during the experiment with each workload.

V. Related Work

A preliminary evaluation of an earlier version of WS-ROS was presented in poster form [4].

This study proposes an approach for managing an HEC storage system that uses both SSD and HDD storage devices. Because of the attractive characteristics of SSDs and their complementary nature to HDDs, others have also studied ways of managing heterogeneous storage systems. To our knowledge, however, none of the existing studies have proposed a management approach using working set information to guide lazy data reorganization within the storage system.

Hystor [3] is an exemplar hybrid storage system where hot and performance critical data are placed in an SSD, but other less important data is stored on conventional HDD devices. Hystor maintains a remap area to map addresses and control data flow. It also maintains a write-back area in the SSD to delay the writes of dirty blocks to HDDs, which is especially beneficial for write-intensive workloads. Our proposed data reorganization scheme differs from Hystor in that it uses a working set model to guide data reorganization and considers reads and writes independently. Also, support for HEC storage is a primary focus for WS-ROS, but not for Hystor.

ComboDrive [16] and HybridStore [11] are two other approaches proposed for managing a storage system comprised of SSDs and HDDs. ComboDrive integrates the capacity of an SSD and HDD by maintaining a global mapping table from virtual addresses to physical addresses. Data can be stored on either the SSD or HDD. When initially written, ComboDrive decides whether to place the data on the SSD or HDD based on its file type. Like WS-ROS, ComboDrive can relocate data later depending on how it is accessed. However, ComboDrive relocates data based on whether it is accessed contiguously or randomly. In contrast, WS-ROS considers the working set of data accesses to determine what data to relocate and where it should be relocated. These two approaches are complimentary and could be combined. HybridStore uses a capacity planner to find a cost-effective storage configuration when data is initially written. It uses a dynamic controller to predict the performance of potential configurations using historical data. Instead, WS-ROS uses simpler guidelines for moving data that should result in less overhead for determining a good data reorganization. Even so, WS-ROS could also integrate a performance prediction approach similar to HybridStore to further refine its data placement decisions.

Hui et al [9] examine heterogeneous storage systems from the perspective of power consumption. This study argues that such heterogeneous storage systems can use less energy than a similar HDD-only storage system. The study provides good motivation for using heterogeneous storage systems. Unlike the Hui study, however, our work focuses on the performance advantages of heterogeneous storage systems for HEC systems. Our work also considers situations with more than one SSD.

VI. Summary

Many scientific and engineering applications run on high-end computing (HEC) platforms consume and/or produce large amounts of data. Traditional HEC storage systems consisting solely of HDDs provide excellent capacity at reasonable cost, but can fail to provide sufficient performance for some types of workloads such as read-intensive and random traffic patterns. Recently, solid state drives (SSDs) using flash non-volatile memory have emerged as storage devices with complimentary characteristics to HDDs, and storage system designers have begun to explore designs that integrate SSDs with HDDs in HEC storage systems. In this study, we propose a Working Set-based Reorganization Scheme (WS-ROS) for managing heterogeneous storage systems. The goals for WS-ROS are to allow applications to write data at full storage system bandwidth to all devices, but then to change the placement of the data so that data that is likely to be read is located on SSD devices but data that is likely to be written, or unlikely to be accessed at all, is located on HDDs. In WS-ROS, a background process accomplishes this data reorganization when the storage system is idle. The benefit of this reorganization is that it takes advantage of the SSDs superior performance in servicing read requests while avoiding the writes to the SSD that decrease its effective lifetime. We evaluated our proposed approach by implementing our WS-ROS approach in a parallel file system simulator, and then measuring its performance and overhead when servicing I/O requests from three different I/O-heavy workloads. Our results suggest that heterogeneous storage systems using the WS-ROS approach for data reorganization can substantially outperform heterogeneous storage systems that rely solely on the initial placement of the data within the storage system. We are currently integrating our proposed approach into a production parallel file system to support further evaluation of its performance benefit and overhead in real-world settings.

Acknowledgment

This research is sponsored in part by the Advanced Scientific Computing Research program, Office of Science, U.S. Department of Energy. This research is also sponsored in part by Texas Tech University startup grant and National Science Foundation under NSF grant CNS-1162488. The work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for SSD performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70, 2008.

[2] A. Badam and V. S. Pai. SSDAlloc: hybrid SSD/RAM memory management made easy. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, pages 16–16. USENIX Association, 2011.

[3] F. Chen, D. Koufaty, and X. Zhang. Hystor: making the best use of solid state drives in high performance storage systems. In *Proceedings of the international conference on Supercomputing*, pages 22–32. ACM, 2011.

[4] J. Chen and Y. Chen. Unified and efficient HEC storage system with a working-set based reorganization scheme. In *In the IEEE International Conference on Cluster Computing (Cluster'13)*. IEEE, 2013.

[5] J. J. Chen, P. C. Roth, and Y. Chen. Using pattern-models to guide SSD deployment for big data applications in HPC systems. In *In the Proc. of the 2013 IEEE International Conference on Big Data (Big Data'13)*. IEEE, 2013.

[6] A. Choudhary, W. Liao, K. Gao, A. Nisar, R. Ross, R. Thakur, and R. Latham. Scalable i/o and analytics. In *Journal of Physics: Conference Series*, volume 180, page 012048. IOP Publishing, 2009.

[7] DB2 parallel edition. http://www.cs.umd.edu/projects/hpsl/mambo/db2.html.

[8] A. Gupta, Y. Kim, and B. Urgaonkar. *DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings*, volume 44. ACM, 2009.

[9] J. Hui, X. Ge, X. Huang, Y. Liu, and Q. Ran. E-HASH: an energy-efficient hybrid storage system composed of one SSD and multiple HDDs. *Advances in Swarm Intelligence*, pages 527–534, 2012.

[10] Ior: Parallel filesystem i/o benchmark. http://github.com/chaos/ior.

[11] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam. Hybridstore: A cost-efficient, high-performance storage system combining SSDs and HDDs. In *MASCOTS, 2011 IEEE 19th International Symposium on*, pages 227–236. IEEE, 2011.

[12] R. Latham, N. Miller, R. Ross, and P. Carns. A next-generation parallel file system for Linux clusters. *Linux-World*, pages 56–59, Jan 2004.

[13] N. Liu, J. Cope, P. H. Carns, C. D. Carothers, R. B. Ross, G. Grider, A. Crume, and C. Maltzahn. On the Role of Burst Buffers in Leadership-class Storage Systems. In *MSST*, pages 1–11, 2012.

[14] F. Moore. Storage and energy - the heat is on. http://www.horison.com/, 2007.

[15] Nigel's performance monitor for Linux. http://nmon.sourceforge.net/pmwiki.php?n=Main.HomePage.

[16] H. Payer, M. Sanvido, Z. Bandic, and C. Kirsch. Combo drive: Optimizing cost and performance in a heterogeneous storage device. In *First Workshop on Integrating Solid-state Memory into the Storage Hierarchy*, volume 1, pages 1–8, 2009.

[17] Parallel web server. http://www.cs.umd.edu/projects/hpsl/mambo/web.html.

[18] S. Son, S. Lang, P. Carns, R. Ross, R. Thakur, B. Ozisikyilmaz, P. Kumar, W. Liao, and A. Choudhary. Enabling active storage on parallel i/o software stacks. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–12. IEEE, 2010.

[19] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at Facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 1013–1020, New York, NY, USA, 2010. ACM.

[20] M. Uysal, A. Acharya, and J. Saltz. Requirements of i/o systems for parallel machines: An application-driven study. 1998.

[21] C. Wang, S. S. Vazhkudai, X. Ma, F. Meng, Y. Kim, and C. Engelmann. NVMalloc: Exposing an aggregate SSD store as a memory partition in extreme-scale machines. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 957–968. IEEE, 2012.

[22] Q. Yang and J. Ren. I-CASH: Intelligently coupled array of SSD and HDD. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 278–289. IEEE, 2011.

[23] X. Zhang, K. Davis, and S. Jiang. iTransformer: Using SSD to improve disk scheduling for high-performance i/o. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 715–726. IEEE, 2012.