Storage Workload Isolation via Tier Warming: How Models Can Help

Ji Xue and Feng Yan, *College of William and Mary;* Alma Riska, *EMC Corporation;* Evgenia Smirni, *College of William and Mary*

https://www.usenix.org/conference/icac14/technical-sessions/presentation/xue

This paper is included in the Proceedings of the
11th International Conference on Autonomic Computing (ICAC '14).
June 18–20, 2014 • Philadelphia, PA
ISBN 978-1-931971-11-9

Storage Workload Isolation via Tier Warming: How Models Can Help

Ji Xue[1], *Feng Yan*[1], *Alma Riska*[2], *and Evgenia Smirni*[1]

[1]College of William and Mary, Williamsburg, VA, USA, xuejimic,fyan,esmirni@cs.wm.edu

[2]EMC Corporation, Cambridge, MA, USA, alma.riska@emc.com

Abstract

Storage systems are often deployed in a tiered form to enable high performance and availability. These tiers utilize all possible volatile and non-volatile storage technologies, including DRAM, SSD, and HDD. The trade-offs among their cost, features, and capabilities can make their effective integration into a single storage entity complex. Here, we propose an autonomic technique that learns user traffic patterns in a storage system over long time-scales to optimize user performance but also volume of completed system work. Our purpose is to multiplex as best as possible user workload with storage system features (e.g., voluminous internal system work) such that the latter is not starved but rather completed with minimal impact on user performance. Key to achieving the above is to use an autonomic learning engine to predict *when* the user workload intensity increases/decreases and then *proactively* stop/start bulky internal system work. Being proactive allows the system to effectively bring into the fast tier the active user working set just-in-time and right before it is needed most, i.e., when user traffic suddenly peaks. We illustrate the effectiveness of this mechanism by using both trace driven simulations from production systems as well experiments on a real testbed.

1 Introduction

As data storage technologies such as flash make their way into either enterprise storage [18, 11] or cloud storage [1], it is essential to integrate them with existing and (usually) slower, cheaper, and even vintage ones, e.g., hard disk storage, in order to strike a good balance among overall performance, availability, and cost [10, 16]. Following tradition in costs across the data path of a computer system, the fastest data tier, e.g., DRAM, is the most expensive, while the slowest tier, e.g., hard disk storage, is the least expensive per unit of data stored. In high-performing enterprise storage systems, it is common to find large DRAM caches, reaching as much as hundreds of GByte capacity, SSD caches reaching tens of TBs capacity, and a variety of high-end HDDs (15KRPM SAS HDDs) and low-end HDDs (7200RPM Nearline-SAS HDDs), all together exceeding the PByte-range of storage capacity. Given such a structure in the IO hierarchy, the expectation is that the bulk of user workload is served by the fast tiers, while slow tiers are used to persistently store the majority of data as well as improve on storage capacity and data reliability. The challenge lies in determining *what* portion of the voluminous data set to bring up to the smaller fast tiers and *when* to do that such that the benefits with regard to user performance and overall system operation are the highest.

Storage systems, both enterprise ones and those supporting web services, often receive the bulk of user traffic during business hours on weekdays. In addition, the storage system generates itself a considerable amount of internal traffic as a result of complex features that aim to enhance performance, reliability, availability, and integrity. Such system internal work includes, but is not limited to, making additional copies of the data off-site for added disaster recovery capabilities, snapshooting, deduplication, and policy compliance in a multi-tenant system. Recently, other sources of work have emerged in scaled-out storage systems such as virtual data analytics clusters that are brought up on demand to conduct analysis on large data sets without moving the data to a separate compute cluster, requiring effective interleaving of user and system workloads. In Figure 1, we show the average arrival intensity of user and system internal work over three days in a single data node of a large distributed storage cluster supporting a web application. During the day, the data node receives user requests that peak in intensity at around noon. During night, at regular intervals, the system generates its own work, which clearly is bulky and would have impacted user performance significantly if not scheduled during off-peak hours.

Because the system work shown in Figure 1 is the result of several features, it greatly surpasses in intensity all other user traffic. More importantly, its working set is (usually) much larger than the user working set. As a result, in a well balanced multi-tiered system, system work could negatively impact data placement policies which

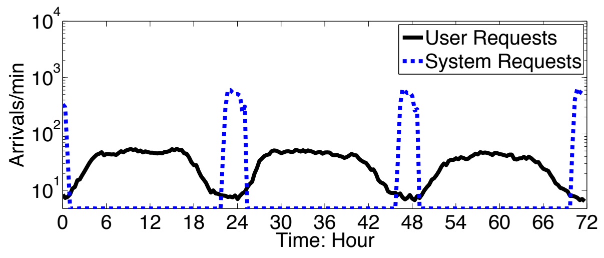


Figure 1: User and system arrival intensity (number of arrivals per minute) over a three day period.

ensure that the most active working set is in the highest performing tier. To schedule system work at regular intervals (e.g., at around midnight, as in Figure 1) does not necessarily guarantee good user performance (note that the figure illustrates only the arrival intensity, not work that needs to be done). Here, we contend that it is necessary to pair scheduling of system work with other system metrics measuring user activity and workload, such as utilization due to user-traffic, user traffic intensity, and fast storage tier hit rates, in order to improve overall system usage while ensuring that system work completes timely.

A critical difference between system internal work and user traffic is that the corresponding working sets are vastly different in both footprint and location within the storage system. Generally, the system working set is much larger than the user working set. As a result, in a multi-tiered storage system with tiers having different capacities and performance characteristics, standard efforts to isolate system and user workload in-time may still result in poor user performance. As the system transitions between system work to user work, high performing tiers could experience high user miss rates. Warming up the faster tier with the user working set is not instantaneous [25] and unless done proactively, performance degradation of user traffic can become unsurmountable.

In this paper, we propose an autonomic technique that learns the intensity patterns of user work within a probabilistic model over long time-scales, i.e., days and hours. The prediction of user intensity is paired with the knowledge of tier capacity, performance differences across tiers, and other metrics such as active user data set to derive a schedule for system work, i.e., *when* to start and stop it. Predicting user intensity patterns at large time scales can support

- proactively stopping system work *before* the user intensity increases and warming up the fast tier with the user working set,
- scheduling system work according to predicted user activity patterns such that large amounts of system work is completed with minimal impact on user performance, and
- avoiding instability due to short-lived, low user intensity that may erroneously initiate voluminous system work if short time-scale prediction or if reactive (i.e., feedback-based) scheduling is used.

Our methodology is light-weight and robust. Its benefits are evaluated via trace-driven simulation and actual experiments on a real test-bed. We do comparisons against feedback-based techniques that are usually applied in industry. Our experiments indicate that the larger the fast tiers and the larger the active user working set, the higher the benefits of having predictive models to schedule bulky system work.

This paper is organized as follows. Section 2 presents a workload characterization of user workload in a storage system supporting web data services. In Section 3, we present our predictive algorithm. Section 4 presents experiments on a real system and trace driven simulations that demonstrate the effectiveness of our technique. Section 5 discusses related work. We conclude in Section 6.

2 Trace Overview and Motivation

In this section, we present a set of production traces that describe how a storage system is utilized by a large scale web application. The traces contain the user IO intensity (in IO requests per second) in a scale-out storage back-end of a mid-size web service provider[1]. The web service has multiple locations that serve the user workload based on geography. As a result, in each location the workload intensity follows well the day/night pattern (working hours vs. non-working hours) as well as weekday/weekend patterns. Here we focus on the traffic received by a single data node. However, because of the load balancing in the storage system, the behavior observed in a single node persists across all other data nodes in the cluster.

Figures 2 and 3 show the average arrival intensity of user requests per minute averaged over 10 minutes and 1 hour intervals for 10 days and 35 days periods, respectively. There is a clear daily and weekly pattern in the workload intensity. This is expected since nowadays large scale web services, although available 24/7 worldwide, are deployed in geographically distributed data enters, resulting in clear day-and-night patterns in each of the available locations. Similar patterns are seen also in enterprise storage, which although different in nature from web storage, serves heavy traffic during business hours and much less during night hours. These patterns suggest opportunities for predicting user traffic intensity. The ability to predict these drastic changes can be used to prepare the system proactively for the heavy user workload, for example by moving the active user data set to the fast tier *before* it starts being accessed. Effective prediction should also ensure that the system schedules long and resource demanding internal work only when it is safely predicted that the system is to enter a long period of low utilization.

[1]Due of confidentiality agreements, the traces or provider details can not be made publicly available.

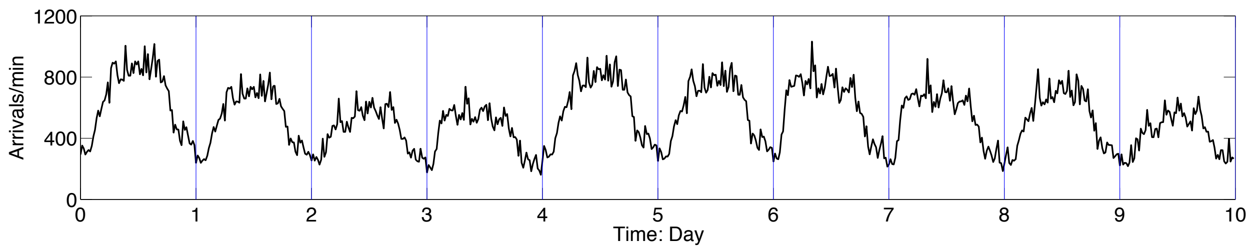


Figure 2: IO request arrival intensity (number of arrivals per 10 minutes) over 10 days.

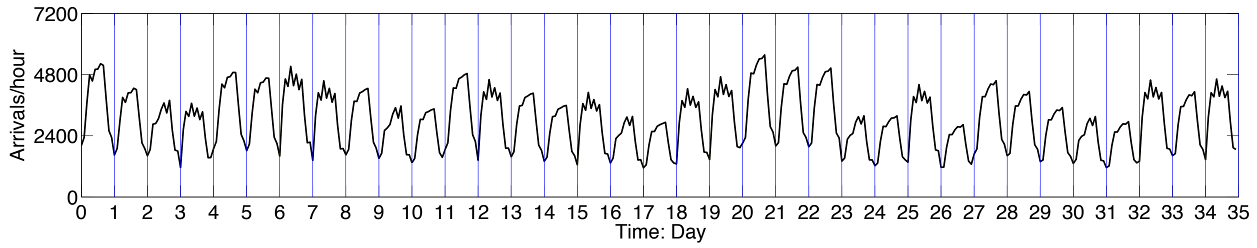


Figure 3: IO request arrival intensity (number of arrivals per hour) over 35 days.

We plot the empirical density of the user arrival rate at a granularity of a minute in Figure 4 for all 35 days. The graph illustrates a clear bi-modal pattern, which confirms that the arrival intensity changes between two general states that we roughly classify as high/low. In the next section, we show how we incorporate the stochastic characteristics of the user arrivals to derive a model that predicts the duration of each high and low intensity period.

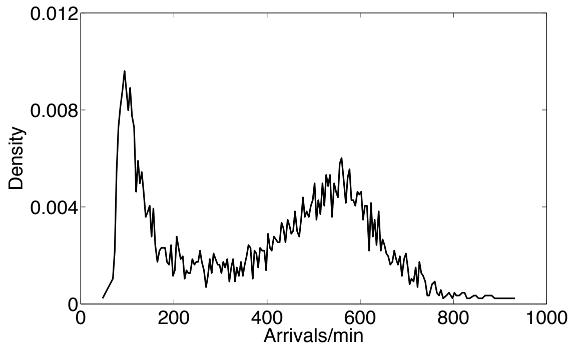


Figure 4: Empirical density of the arrival rate of user requests.

Prediction of low/high intensity periods provides the system with the information it needs to intelligently interleave user workload with voluminous system internal work. For example, it can proactively stop the system work and warm up the fast tier storage just before the high user intensity period so that the majority of user traffic is served by the fast tier rather than the slow tier. As discussed later in the paper, we do not determine here what data to bring but rather when to bring them in the fast tier. Determining the user working set (i.e., what to bring) is outside the scope of this paper.

To illustrate the benefits of predicting arrivals of high and low intensity periods and motivate our work, we evaluate three scenarios on handling system work in a data node:

- *user only*: no system work is interleaved with user traffic,

- *reactive*: system work runs only during low user utilization periods; when user high utilization is detected, the system work is stopped and reactively the fast tier cache is warmed up with active user data,

- *proactive with future knowledge*: system work runs only during low user utilization periods; since we know *a priori* when user high utilization starts, we stop system work early enough to allow for the fast tier cache to be warmed up with active user data right before the surge of user work.

Figure 5 illustrates the CDF of user response time when a single day of trace data (see Figure 3) is used to drive a simulation of the above three scenarios. For the two policies that allow system work, the simulation starts and stops it at the same time, with the purpose of evaluating only the benefit of proactive vs. reactive fast tier warm up (which takes the same amount of time in both scenarios). Clearly, with a reactive warm up a large portion of user requests experience long response time by being served from the slow tier of the system. Both the body and the tail of the user response time distribution benefit greatly by a proactive fast tier warm up, which can be possible only if a model enables prediction of arrival of high user utilization periods.

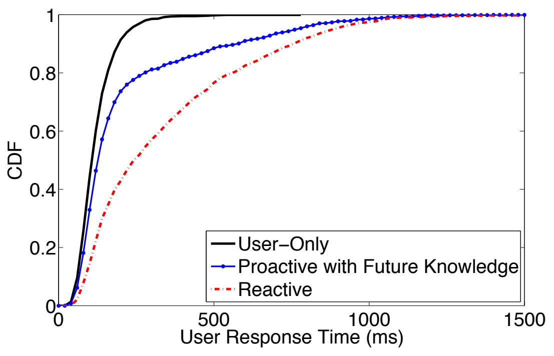


Figure 5: CDFs of user response time of day 20 for different algorithms.

3 Model-based Storage Tiering

The data patterns described in Section 2 (see Figures 2 and 3) favor prediction. User intensities go through a clear high/low pattern which if captured accurately can be used to intelligently interleave workloads with widely different demands. Indeed, this is the current state of the practice in most data centers [6], a practice that is also illustrated in Figure 1. Yet, rather than schedule system work conservatively as in Figure 1, we aim to develop a model that would allow for better overall system resource utilization by completing aggressively system work and achieve better performance isolation across user and system workloads.

We first present a Markovian-based model that captures the duration of low/high traffic intensities in user arrivals across different time scales (i.e., daily distinguishing between weekday/weekend and hourly distinguishing between day/night activity). We also develop a model that captures the changes in user performance as a function of fast tier hit rate. Finally, we apply these models to predict when such periods of high/low intensities arrive to schedule system work and cache warm up with the goal of optimizing performance.

3.1 Traffic Intensity Prediction Model

The preliminary workload analysis in Section 2 shows that there are repeatable low/high daily intensity patterns. We refer to the state with high average arrival intensity as the *High* state and the lower one as the *Low* state. The threshold for distinguishing the *High* and *Low* states can be discovered via statistical analysis. Alternatively, the threshold could be user-defined. We need to determine the following: how long does the user traffic resides in each state, i.e., the *duration* of each state and the conditional probability that there is a transition from one state to the next. The analysis of Section 2 also shows that weekend *High* state intensity is different from weekday *High* state intensity. We aim to capture these patterns in order to distinguish days with overall less intensity from days of higher intensity.

To capture this effect, we use a hierarchical model that captures different *types* or *classes* of high/low intensities. The difference between these is that the average intensity for the *High* or *Low* states may be different as well as the duration of each state. Note that in addition to transitions within the *High*/*Low* states within each class, there are probabilistic transitions from the states that represent one class (e.g., weekdays) to another class (e.g., weekend or holidays). This hierarchical model is shown in Figure 6. The model in Figure 6 has two classes of high/low intensities (capturing day/night and weekday/weekend patterns). However if more classes are detected then the hierarchy of the model can grow to accommodate them.

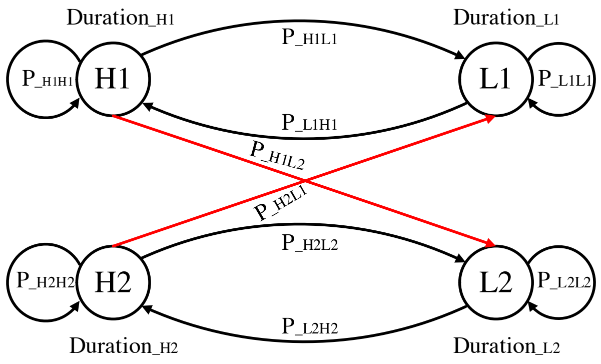


Figure 6: High level Markovian model.

We start building the model by first categorizing the observed arrival intensities. We use clustering to determine how many types of low/high intensities exist in the workload using *Silhouette* [19] and *K-means*. *Silhouette* is used to calculate the dissimilarity value $s(i)$ of the average arrival intensity of day i. The dissimilarity value $s(i)$ is defined as:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}},$$

where i is the day index, $a(i)$ is the average dissimilarity of day i to all other days within the same cluster, and $b(i)$ is the lowest average dissimilarity of day i to all days in a different cluster. Distance measures are the most common for calculating the dissimilarity values $a(i)$ and $b(i)$. The values of $s(i)$ are in $[-1, 1]$ and the larger its value the better, e.g., when $s(i)$ approaches to 1, $a(i) \ll b(i)$, which means that the distance between data within each cluster is the smallest. More specifically, the following three steps are performed to determine the number of clusters in the model:

1. Define the upper bound of the number of clusters as $\sqrt{\frac{n}{2}}$, where n is the total number of days in the historical information[2];

2. Calculate the average $s(i)$ for a different number of clusters;

[2] $\sqrt{\frac{n}{2}}$ is used as a rule of thumb in *K-means* to avoid too many clusters and unnecessary overheads [13].

3. Choose the number of clusters with the highest $s(i)$.

After the number of clusters is determined, we calculate the transition probabilities between them. We also estimate the duration of *High*/*Low* states within each cluster as well as their transition probabilities.

In a live system, the goal is to have an initial model built with the data collected over the first few weeks of operations. Then, the model is updated continuously as new data on user workload is collected, so that any changes in system operation and user access patterns are reflected in the model.

Figure 7 illustrates the effectiveness of prediction by comparing it with actual state changes. The dashed lines illustrate the points where the model detects a change in the state (from *High* to *Low* or *Low* to *High*). The dotted line illustrates the actual state changes. The graph shows that the model predicts effectively changes from one user intensity state to the next.

3.2 Fast Tier Hit Rate

The fast tier hit rate in a storage system is related to many factors, including its capacity and active user working set. Here, we provide an estimation method for the instant fast tier hit rate with the goal of estimating how it changes as active user data moves from the slow tier up to the fast tier and vice versa.

As we focus mostly on large capacity fast tiers as well as large active data sets, it becomes necessary to warm up the fast tier cache rather than allow it to be warmed up gradually by the user accesses. Figure 5 clearly illustrates that warming up the cache can tremendously affect performance.[3]

The average IO service rate for user traffic is a combination of fast storage tier access speed and slow storage tier access speed and can be expressed as follows:

$$\mu(t) = (1+S(t)) * \mu_{origi} = H * R_{fast} + (1-H) * R_{slow}, \tag{1}$$

where $\mu(t)$ is the average service rate of user traffic at time t. μ_{origi} is the original average service rate of user traffic, e.g., when there is no system work. $S(t)$ is the service slowdown which describes how the average service rate changes from the original one. H is the fast tier hit rate, R_{slow} is the average slow storage tier access speed and R_{fast} is the average fast storage tier access speed, implying that the fast tier hit rate can be defined as follows:

$$H = \frac{(1+S(t)) * \mu_{origi} - R_{slow}}{R_{fast} - R_{slow}} \tag{2}$$

with $0 \leq H \leq 100\%$.

[3]The model presented here can be trivially extended to capture the no warm up case, i.e., passively move data when first accessed, by changing the parameter of the average transfer speed between the fast storage tier and slow storage tier to a function that is determined by the intensity of arrivals.

When system work is served, the average service rate of user traffic unavoidably decreases due to sharing of the fast storage tier with the working set of the system workload. We assume that the Service Slowdown increases linearly over time during the periods of serving system work:

$$S(t) = S(t_{i-START}) + a * t, \tag{3}$$

where $S(t_{i-START})$ is the Service Slowdown at the beginning of the time window i serving the additional work. This parameter is necessary because the slowdown effects may propagate through several windows of time. Finally, a is an coefficient that describes how fast the slowdown increases during system work serving periods.

The maximum slowdown occurs when the fast storage tier is filled with the system working set, and unavoidably all user traffic is served from the slow storage tier, therefore

$$R_{slow} = (1+S_{max}) * \mu_{origi}, \tag{4}$$

or

$$S_{max} = \frac{R_{slow}}{\mu_{origi}} - 1. \tag{5}$$

Note that μ_{origi} may not equal to R_{fast} because the Fast Tier Hit Rate may not equal to 100% even when no system work is served.

When $S(t_{i-START}) = 0$, i.e., when the fast storage tier is filled with the user working set, T_{fast} expresses the period before system work data starts occupying the entire fast storage tier. After the user working set is removed from the fast storage tier, the user service slowdown reaches its maximum, i.e., no user IO requests can use the fast storage tier to improve performance. According to Eq. 3, we have:

$$S_{max} = a * T_{fast}. \tag{6}$$

By definition, the capacity C equals to the transfer speed F multiplied by time, therefore:

$$C = F * T_{fast} \tag{7}$$

and

$$\mu(t) = (1+S(t)) * \mu_{origi}. \tag{8}$$

From Eq. 3 and Eq. 5 – Eq. 7, when $t > t_{i-END}$, we have :

$$\begin{aligned} S(t) &= S(t_{i-START}) + \tfrac{S_{max}*F}{C} * t. \\ &= S(t_{i-START}) + \tfrac{F}{C} * (\tfrac{R_{slow}}{\mu_{origi}} - 1) * t, \end{aligned} \tag{9}$$

which shows that the user service slowdown is related to the average fast and slow storage tier access speed, the average transfer speed between fast and slow storage tiers, the original average service rate of user traffic, and the fast tier capacity.

When the system stops serving system work, the user service slowdown due to sharing of the fast tier with system workload decreases over time. We assume that this decrease is linear across time:

$$S(t) = S(t_{i-END}) - b * t. \tag{10}$$

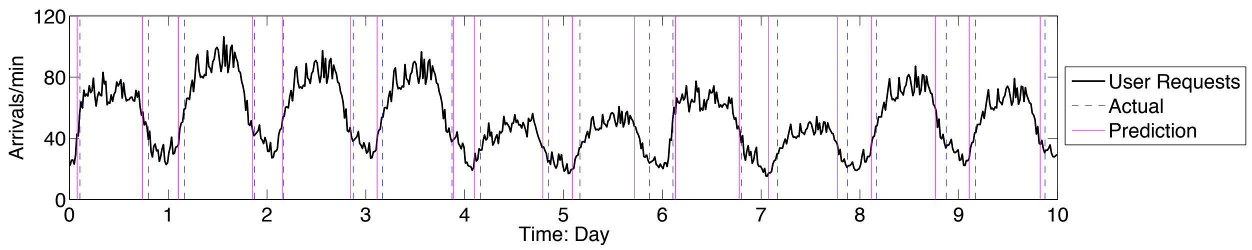


Figure 7: Comparison of actual and predicted arrival intensity state changes.

$S(t_{i-END})$ is the user service slowdown at the end time of system work serving window i and b is an coefficient that describes how quickly the slowdown decreases over non-system work serving periods.

Similarly, when $S(t_{i-END}) = S_{max}$, i.e., when the fast tier is filled with all system work data, it takes T_{fast} time units before the user working set refills the entire fast tier. After the user working set is restored, the user service slowdown reaches its minimum, i.e., to the original service rate without any system work. According to Eq. 10, when $S(t_{i-END}) = S_{max}$, $S(t) = 0$:

$$0 = S_{max} - b * T_{fast}. \tag{11}$$

By comparing Eq. 3 and Eq. 10, we have $b = a$. Therefore, for $t_{i-START} \leq t \leq t_{i-END}$, we have:

$$\begin{aligned} S(t) &= S(t_{i-START}) - \frac{S_{max}*F}{C} * t \\ &= S(t_{i-START}) - \frac{F}{C} * (\frac{R_{slow}}{\mu_{origi}} - 1) * t. \end{aligned} \tag{12}$$

Using Eq. 9 and Eq. 12

$$S(t) = \begin{cases} S(t_{i-START}) - \frac{F}{C} * (\frac{R_{slow}}{\mu_{origi}} - 1) * t, \\ \qquad \text{for } t_{i-START} \leq t \leq t_{i-END}, \\ \\ S(t_{i-END}) + \frac{F}{C} * (\frac{R_{slow}}{\mu_{origi}} - 1) * t, \\ \qquad \text{for } t > t_{i-END}. \end{cases} \tag{13}$$

From Eq. 2 and Eq. 13, it follows that the hit rate is:

$$H = \begin{cases} \frac{\mu_{origi}(S(t_{i-START})+1) - R_{slow} - \frac{F}{C}*(R_{slow}-\mu_{origi})*t}{R_{fast}-R_{slow}}, \\ \qquad \text{for } t_{i-START} \leq t \leq t_{i-END}, \\ \\ \frac{\mu_{origi}(S(t_{i-START})+1) - R_{slow} + \frac{F}{C}*(R_{slow}-\mu_{origi})*t}{R_{fast}-R_{slow}}, \\ \qquad \text{for } t > t_{i-END}. \end{cases} \tag{14}$$

```
1. if system in characterization state do
   a. collect arrival intensity information and use
      Silhouette and K-means to do clustering.
   b. Compute duration of High/Low states per cluster
      and transition probabilities within and across clusters
   c. Build the Markovian model with the computed
      parameters and continue updating the model
      while the system is in operation
2. if the system is in serving system work state do
   a. Predict how long the system will be in Low state
      and compute warmup time for fast storage tier $T_{fast}$
   b. if residual time in Low state $> T_{fast}$
      i.   compute the Fast Tier Hit Rate and
           average utilization $UTIL_{past}$ of past $t$ minutes
      ii.  if no outstanding user request
              and $H >= H^{low}_{threshold-lower}$
              and $UTIL_{past} <= UTIL_{threshold}$
              schedule system work
      iii. else if $H <= H^{low}_{threshold-lower}$, stop serving
              system work until $H = H^{low}_{threshold-upper}$
           go to Step 2.b.iv
      iv.  else process user request or stay idle
      v.   go to Step 2.b
   c. else if the residual time in Low state $< T_{fast}$
      i.   stop serving system work and warm up the fast tier
      ii.  go to Step 2.b
   d. else if system in high arrival intensity state
      i.   if no outstanding user request
              and $H >= H^{high}_{threshold-lower}$
              and $UTIL_{past} <= UTIL_{threshold}$
              schedule system work
      ii.  else if $H <= H^{high}_{threshold-lower}$, stop serving
              system work until $H = H^{high}_{threshold-upper}$
           go to Step 2.d.iii
      iii. else process user request or stay idle
      iv.  go to Step 2.b
      go to Step 1
```

Figure 8: Prediction-based deployment of systems work.

3.3 Storage Tiering

Figure 8 presents an algorithm for scheduling system work in a multi-tier storage system. In the *characterization state*, the algorithm collects arrival intensity information to compute the parameters and build the Markovian model. Based on the *Low* and *High* states duration and the fast tier warm up time, the algorithm schedules system work. For example, during the *Low* state, the system work is served concurrently with the low user traffic because the overall performance impact is small. The thresholds of the fast tier hit rate can be considered as a control knob. For example, the thresholds of *Low* state can be set much smaller than the threshold of those of the *High* state so that more system work can be finished.

The algorithm proactively warms up the fast tier by stopping system work ahead of the predicted arrival of the *High* state. The warm up time depends on the fast tier capacity and can be computed via Eq. 7. Such proactive action is critical because the fast tier can not be warmed up instantly. For large fast tiers, the warm up may take a long time, hours or in some cases even days [25]. Without proactive warm up, the user requests that arrive in the initial period of the *High* state are to be impacted significantly.

4 Experimental Evaluation

In this section, we evaluate the proposed scheduling framework via an extensive set of experiments in a real system and through trace-driven simulations. We first describe the testbed and the workload we use in Section 4.1 and then show the respective experimental results that validate our method in Section 4.1.1. Then we use our traces from Section 2 to drive a set of simulation experiments for more sensitivity analysis of our predictive model. Throughout this section we compare our framework with other common practices such as feedback-based techniques.

4.1 Experimental Testbed and Workloads

Our testbed consists of a server with a disk enclosure attached to it, which provides data services to a host. Its memory is 12GB and the disk enclosure has 12 SATA 7200RPM HDDs of 3TB each. In our experiments the system memory emulates the fast tier and the disk enclosure the slow tier used for the bulk of the data. The benefits of effective workload prediction are high for system with large gaps in the performance characteristics across tiers. For the shake of presentation clarity, we evaluate here the predictive framework on a system with two-tiers only that provide services that differ by one order of magnitude. We stress that our approach can be directly applied in a system with *any* number of storage tiers.

The workload is generated and measured at the host machine. We use `fio` [2] as the IO workload generator for the flexibility it provides to generate a wide range of IO workload intensities and general patterns. We generate two types of IO workloads, user and system, which differ on the active working set size rather than their access pattern. The working set size for the user workload is 1GB[4], i.e., such that it always fits into the memory of the server that emulates the fast tier. The system work has an active working set of 24GB, i.e., it does not fit fully into the fast tier and the large slow tier is accessed to retrieve the data. The access pattern for both user and system workload is 100% small random reads to emulate common enterprise workloads that would benefit from prefetching (warm-up) only if the working set can fully (or almost) fit in the high performing tier (i.e., the SSD).

[4]Experiments with 4GB and 8GB user working sets yielded similar results and are omitted here due to lack of space.

Our framework determines only *when* to warm up the cache with a pre-determined user data set. Determining what data should be brought into the cache is outside the scope of this paper. The user active working set can be determined by evaluating statistically access patterns such as the number of accesses per storage location. Here we also assume that the system is provisioned in such a way that the fast tier can fit the entire (or the majority of the) user active working set. The fast tier is warmed up via a sequential read of the user working set.

We measure the following system work scheduling policies:

- *user-only* - used only as a baseline to evaluate the impact of the additional system work,
- *feedback-based* - a reactive policy that monitors the current load intensity in the system and determines if it is in a high or low intensity period,
- *prediction-based* - a proactive policy (see Figure 8) that uses the proposed Markovian model to predict user traffic intensity by having learned from past data the duration of periods of high and low intensity.

Rules that determine the change of state (from *High* to *Low* or vice versa) for both the feedback-based and the prediction-based policy are the same and follow the discussion in Section 3. The main difference is that by predicting the arrival of the *High* state the system can prefetch the user working set before the state changes and avoid performance penalties in a large portion of user requests. The feedback-based policy is a reactive one: it acts *after* it detects a state change. As a result, user requests arriving right after the state change suffer from performance penalty of being served at the slow tier, till the fast tier is warmed up. The larger the fast tier, the longer it takes to warm it up and the higher the performance penalty of the feedback-based policy.

The prediction-based policy is designed to fall-back on the feedback-based policy: if the prediction time for a *High* state is in the future but the *High* state is already detected, then system work is stopped and the fast tier is warmed-up with the working set reactively.

4.1.1 Measurement Results

Using `fio`, we generate a random reads workload accessing data stored in our server. The intensity of user IO requests is shown in Figure 9 and it emulates very closely the load pattern of user requests shown in Section 2. Note that without any system work, the response time of user IO requests remains in the same range of about 150ms. All IOs are served from the fast tier. The user throughput however does increase by one order of magnitude as the arrival intensity increases. This confirms that the storage

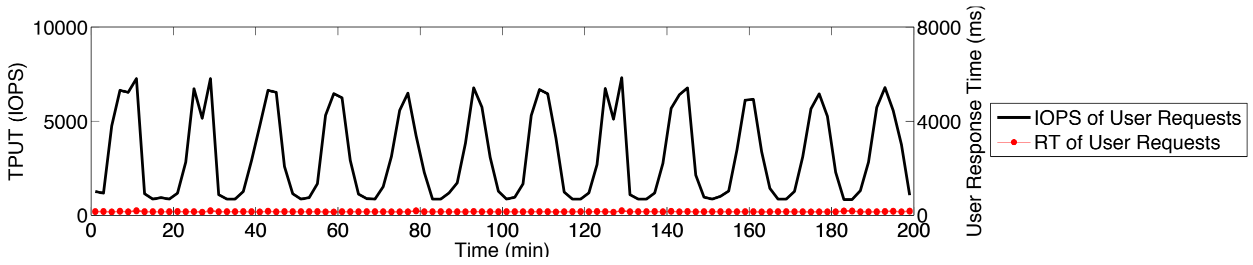


Figure 9: User IOPS (throughput) and user response time over time, *user-only* policy.

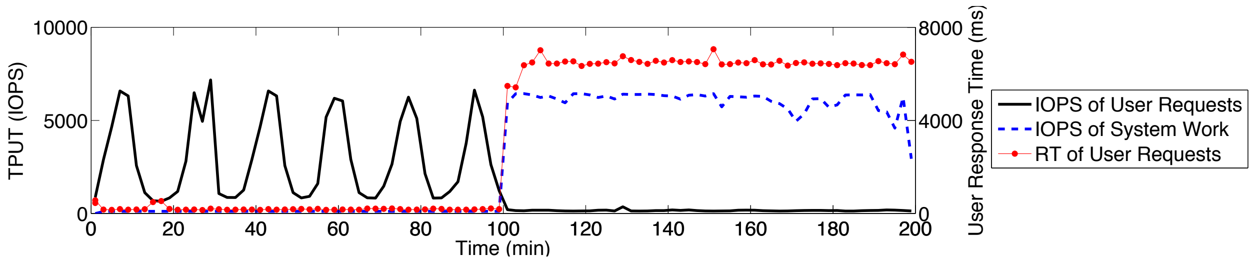


Figure 10: User and system IOPS (throughput) and user response time over time. In the first half of the experiment there is minimal system work, in the second half system work is increased by two orders of magnitude.

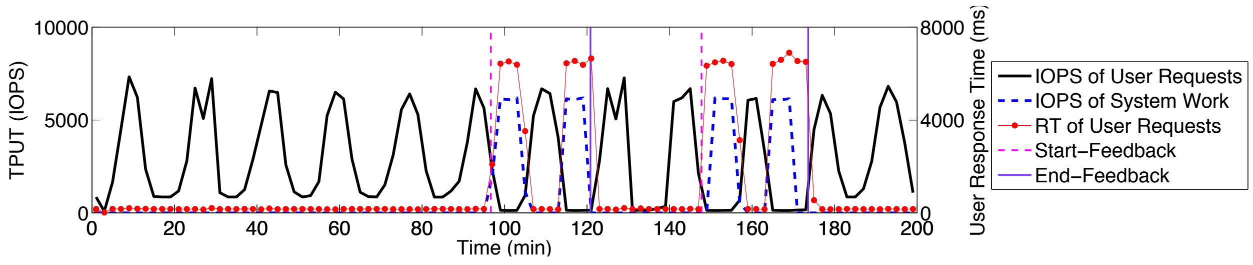


Figure 11: User and system IOPS (throughput) and user response time over time. Till the 100th minute, there is only user workload. In the time periods from the 100th to the 120th minute and the 150th to the 170th minute, the feedback policy is launched. The prediction policy is launched from the 120th to the 150th minute, as well as after the 170th minute to the end of the experiment.

system does not suffer from queuing delays and it has the capacity to sustain more user load.

We add on the same experiment some system work. Initially, the system work is slowed down as to not interfere with the user workload performance. In Figure 10, we show the same user workload but interleaved with system work with very low intensity. For the first 100 minutes, the system throughput reaches up to 121 IOPS. In the next 100 minutes, the intensity of system work increases and its throughput reaches 5968 IOPS, a two-orders of magnitude increase from the first half of the experiment. User performance is not impacted in the first half of the experiment, but system work here is minimal. The figure plots the throughput of both user and system work, as well as the response time of the user workload. In the second part of the experiment (100 min to 200 min) where system work is launched, the increase in user response time grows by two orders of magnitude, while its throughput is very low.

In Figure 11 we do the same experiment but we now activate the feedback and the prediction-based policies in the second half of the experiment, when the system work is launched. The feedback policy is used from the 100th to the 120th minute as well as for the time period between the 150th to the 170th minute. In the rest of the time periods, the prediction-based policy is used. The graph shows that when the prediction policy is activated (i.e., in time periods: 120-150, and 170-200), user performance remains unscathed, both with respect to throughput and response time. In the time periods when the feedback policy is used, there is high throughput of system work but also user response times that are orders of magnitude higher than the user-only case.

What makes the difference in user performance between the feedback and prediction-based policies is the timely fast-tier warm up. Figure 12 captures this effect. In this experiment, we use a very small data set of 1GB and let the fast tier warm up 1) by the accesses of the

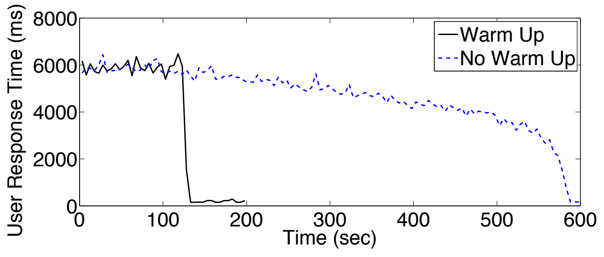


Figure 12: Response time with warm up and without warm up across the experiment time.

regular user workload (i.e., no explicit warm up) and 2) by specifically bringing the working set up to the fast tier (i.e., explicit warm up) via a sequential read of the user working set. While it takes only 130 seconds to bring 1GB of data into our fast tier by reading it sequentially (300 seconds and 700 seconds for 4GB and 8 GB of data, respectively), it takes 600 seconds to fully warm up the cache by the user workload alone (more than one hour for the 4GB and 8GB working sets). As the working sets and fast tier capacities grow to TBytes, it becomes imperative not only to warm up the cache before the high user load starts, but doing it proactively (with the aid of our model) than reactively (feedback). A more fine-grained evaluation of our predictive policy is done via trace-driven simulation in the next subsection.

4.2 Simulation Results

In order to evaluate our predictive approach at a fine-grain level and better understand its statistical properties, we experiment also with a trace-driven simulation that allows us to change the various parameters of the experiment. The simulation, in particularly, allows us to analyze the benefits of the predictive methods as the size of the fast tier increases, without being constricted by the specific hardware as in the case of our limited testbed.

Our simulation is driven by the traces described in Section 2. Since the traces contain only the arrival process, the service process is assumed to be exponentially distributed with a mean service rate that ensures that the response time remains flat during the full range of user arrival intensities. We simulate a two-tiered storage system with configurable capacities and user active data set sizes to experiment with different fast tier warm up times (i.e., 1 minute, 15 minutes, and 60 minutes).

We have implemented the feedback-based and the prediction-based policies for scheduling system work. As a baseline comparison, we also report performance data when no system work is launched (i.e., we also present the user-only case). The simulation, similar to our measurement experiments, is built such that when the system is experiencing high user arrival intensities, the system work is stopped. When the system experiences low arrival intensities then the system serves both user requests and system work. The difference between the predictive and feedback approaches lies in the exact time *when* the system work is stopped and resumed.

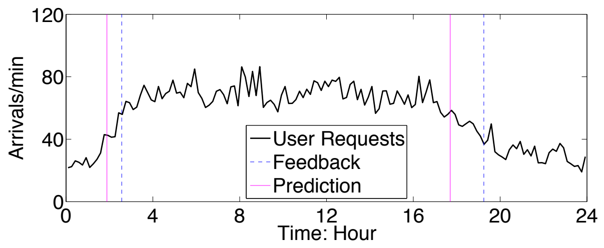
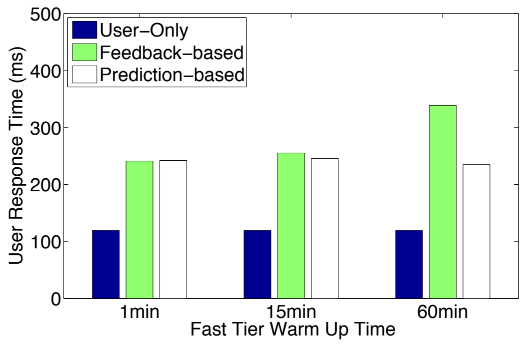


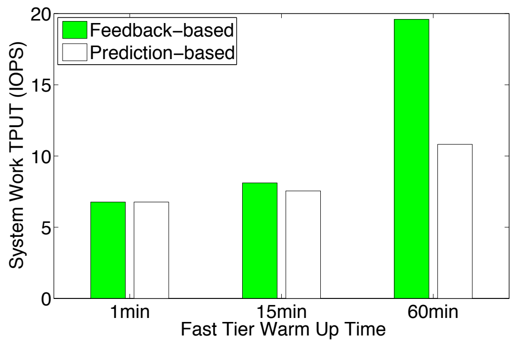
Figure 13: Predicted state change by feedback and prediction methods.

For the results that we present here, the model is already trained with two weeks of trace data and we present results when the model is applied in one of the days (day 20, see Figure 3). Figure 13 illustrates the points where the two models differ. The lines that are marked as feedback illustrate the points in time where the arrival intensity changes after observation. Note the feedback uses on-line detection, so there is a delay between the true change point and detected moment. The prediction lines correspond to the time stamps where the model predicts that there is an imminent load change. Due to the stochasticity of the arrival intensities and the Markovian-based model, the prediction model deviates from the real change that is accurately detected by the reactive, feedback-based method. Yet, this accuracy of the feedback model becomes almost a moot point since it cannot be used to enable tier warm up before the high user load.

In our simulation experiments, we compare the average user response time and the average system work throughput between different fast tier capacities (measured by the time it takes to warm up). The expectation is that the predictive method would detect the incoming *High* state and proactively warm-up the fast tier earlier than the feedback method detects the *High* state after the fact. As a result, system work runs for longer stretches under the feedback method than the predictive method. Consequently, the predictive method completes less system work, but also maintains high user performance. Note that the larger the fast tier, the higher the benefits of the predictive approach, otherwise the system is left to operate under high user arrival intensities and a cold fast tier for longer periods of time. These behaviors are captured in Figure 14 and further corroborated by Figure 15, where the CDF of the user response time is plotted under the scenario of a fast tier requiring 60 minutes to warm up. In the other two cases of smaller fast tiers, the differences between the feedback method and the predictive methods are not as pronounced. As a final note, note that in Figure 15 we have also added the ideal proactive policy that assumes full knowledge of the future workload to initiate the tier warm up. The response time CDF of the prediction-based policy is very close to that of the ideal one, which further argues about the effectiveness of the model prediction.



(a) average user response time



(b) throughput of system work

Figure 14: Performance comparisons via simulation. Note the throughput for system work is null in the user only case.

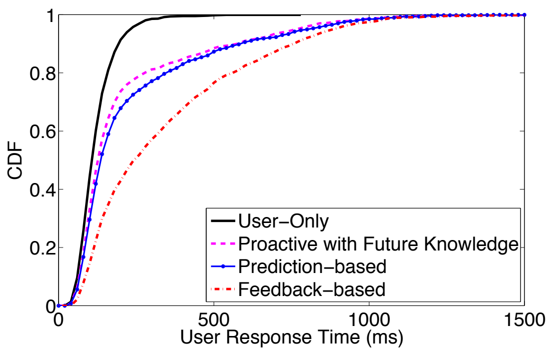


Figure 15: CDF of user response time.

5 Related Work

There is a rich body of work in the literature on storage tiering. Hierarchical storage systems are early examples of storage tiering techniques . HPSS [3] has higher tier (disk) and lower tier (tape), but only allows files to be read or written from the higher tier while the lower tier is treated as an offline device, e.g., data must first migrate to the higher tier before been accessed. VxFS [9] improves the flexibility of the early hierarchical storage systems by allowing user defined placement and migration rules. As the cost of SSDs reduced, SSDs have been introduced in the storage hierarchy. HP'S 3PAR [18] and EMC's FAST [11] are examples of such systems.

Storage tiering is usually critical for meeting service level agreements (SLA) because it can significantly boost the overall system performance. Amazon provides ElastiCache [1] for improving application performance by adding an in-memory caching layer to the infrastructure. FlashTier [20] proposes an interface that is designed for using an SSD as a fast storage tier. Everest [15] offloads bursty I/O workloads by using spare disk bandwidth to a virtual short-term persistent storage so that the I/O request latency during peaks is improved. There are other storage tiering works focused on improving reliability [5, 17, 7] and cost or energy savings [10, 16].

There are various system storage works that are usually scheduled as a "background" activity for various purposes, including replication [21, 23], security [12], and data analysis [22, 14]. Several workload interleaving techniques [8, 24] have been proposed for scheduling such system or background work, but they do not consider storage tiering.

The work most related to ours is optimizing storage cache warm up. Bonfire [25] accelerates the cache warmup by using more efficient pre-load methods. Windows SuperFetch [4] pre-loads the frequently used system and application information and libraries into memory based on the usage pattern in history to reduce the system boot and application launching time. While Bonfire [25] and SuperFetch [4] focus mostly on identifying the data that should be brought into the fast tier for higher efficiency, our work concentrates on identifying *when* to proactively bring a specific and pre-identified data set into the fast tier such that the system can be best utilized by system work with minimal impact on user perceived performance. In this regard, our proposed predictive framework can be viewed as complementary to Bonfire [25] and SuperFetch [4].

6 Conclusions

In this paper, we examine the effects of various workload interleaving techniques in tiered storage and demonstrate the performance benefits of a stochastic, Markovian-based model that can be used to first learn and then predict cyclic patterns in user workload intensity. Using a variety of user workload traces for production systems, we demonstrate the robustness of the model as it effectively suggests when to start and when to stop the deployment of system storage features in order to serve system work during the most opportune low utilization periods.

Acknowledgments

This work is supported by NSF grants CCF-0937925 and CCF-1218758.

References

[1] Amazon ElastiCache. `http://aws.amazon.com/elasticache`. Last visited on: May 19th, 2014.

[2] FIO Benchmark. `http://www.freecode.com/projects/fio`. Last visited on: May 19th, 2014.

[3] HPSS User's Guide. `http://www.hpss-collaboration.org/user_doc.shtml`. Last visited on: May 19th, 2014.

[4] Inside the Windows Vista Kernel. `http://technet.microsoft.com/en-us/magazine/2007.03.vistakernel.aspx`. Last visited on: May 19th, 2014.

[5] AGUILERA, M. K., KEETON, K., MERCHANT, A., MUNISWAMY-REDDY, K.-K., AND UYSAL, M. Improving recoverability in multi-tier storage systems. In *DSN* (2007), IEEE, pp. 677–686.

[6] BIRKE, R., PODZIMEK, A., CHEN, L. Y., AND SMIRNI, E. State-of-the-practice in data center virtualization: Toward a better understanding of vm usage. In *DSN* (2013), IEEE, pp. 1–12.

[7] CHEN, P. M., NG, W. T., CHANDRA, S., AYCOCK, C., RAJAMANI, G., AND LOWELL, D. The Rio file cache: Surviving operating system crashes. In *ASPLOS* (1996), pp. 74–83.

[8] EGGERT, L., AND TOUCH, J. Idletime scheduling with preemption intervals. In *SOSP* (2005), pp. 249–262.

[9] GANESH KARCHE, MURTHY MAMIDI, P. M. Using dynamic storage tiering. *Symantec Yellow Books* (2006).

[10] GUERRA, J., PUCHA, H., GLIDER, J. S., BELLUOMINI, W., AND RANGASWAMI, R. Cost effective storage using extent based dynamic tiering. In *FAST* (2011), pp. 273–286.

[11] LALIBERTE, B. Automate and optimize a tiered storage environment - FAST! *ESG White Paper* (2009).

[12] LI, Y., DHOTRE, N. S., OHARA, Y., KROEGER, T. M., MILLER, E. L., AND LONG, D. D. Horus: Fine-grained encryption-based security for large-scale storage. In *FAST* (2013).

[13] MARDIA, K. Multivariate analysis. *Academic Press* (1979).

[14] MIHAILESCU, M., SOUNDARARAJAN, G., AND AMZA, C. Mixapart: decoupled analytics for shared storage systems. *USENIX HotStorage* (2012).

[15] NARAYANAN, D., DONNELLY, A., THERESKA, E., ELNIKETY, S., AND ROWSTRON, A. I. Everest: Scaling down peak loads through I/O off-loading. In *OSDI* (2008), pp. 15–28.

[16] OH, Y., CHOI, J., LEE, D., AND NOH, S. H. Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems. In *FAST* (2012).

[17] OUSTERHOUT, J., AGRAWAL, P., ERICKSON, D., KOZYRAKIS, C., LEVERICH, J., MAZIÈRES, D., MITRA, S., NARAYANAN, A., PARULKAR, G., ROSENBLUM, M., ET AL. The case for ramclouds: scalable high-performance storage entirely in dram. *ACM SIGOPS Operating Systems Review 43*, 4 (2010), 92–105.

[18] PETERS, M. 3PAR: Optimizing I/O service levels. *ESG White Paper* (2010).

[19] ROUSSEEUW, P. J. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics 20* (1987), 53–65.

[20] SAXENA, M., SWIFT, M. M., AND ZHANG, Y. Flashtier: A lightweight, consistent and durable storage cache. In *EUROSYS* (2012), pp. 267–280.

[21] SHILANE, P., HUANG, M., WALLACE, G., AND HSU, W. Wan-optimized replication of backup datasets using stream-informed delta compression. *ACM Transactions on Storage (TOS) 8*, 4 (2012).

[22] TIWARI, D., BOBOILA, S., VAZHKUDAI, S. S., KIM, Y., MA, X., DESNOYERS, P. J., AND SOLIHIN, Y. Active FLASH: Towards energy-efficient, in-situ data analytics on extreme-scale machines. In *FAST* (2013).

[23] WALLACE, G., DOUGLIS, F., QIAN, H., SHILANE, P., SMALDONE, S., CHAMNESS, M., AND HSU, W. Characteristics of backup workloads in production systems. In *FAST* (2012).

[24] YAN, F., RISKA, A., AND SMIRNI, E. Busy bee: how to use traffic information for better scheduling of background tasks. In *ICPE* (2012), pp. 145–156.

[25] ZHANG, Y., SOUNDARARAJAN, G., STORER, M. W., BAIRAVASUNDARAM, L. N., SUBBIAH, S., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Warming up storage-level caches with Bonfire. In *FAST* (2013).