

Modeling the Performance and Energy of Storage Arrays

Sankaran Sivathanu

Georgia Institute of Technology
Atlanta, Georgia 30332
Email: sankaran@cc.gatech.edu

Ling Liu

Georgia Institute of Technology
Atlanta, Georgia 30332
Email: lingliu@cc.gatech.edu

Cristian Ungureanu

NEC Laboratories America
Princeton, New Jersey 08540
Email: cristian@nec-labs.com

Abstract—We propose a novel framework for evaluating techniques for power optimization in storage. Given an arbitrary trace of disk requests, we split it into short time intervals, extract a set of simple statistics for each interval, and apply an analytical model to those statistics to obtain accurate information regarding the performance and energy characteristics of the system for that workload. The key abstraction used in our analytical model is the *run-length* - a single sequential run of requests at the disk level. Using this abstraction, the model is able to account for arbitrary interactions of random and sequential I/Os in the context of a RAID array, and obtain accurate results with less effort than a detailed individual request-level simulation. Various layout and migration policies aimed at power conservation can be easily expressed as transformations on this set of statistics for each time interval.

We demonstrate the efficacy of our framework by using it to evaluate PAROID, a recently proposed technique for power optimization in storage arrays. We show that the performance and power predicted by the model under the migration and layout policies of PAROID accurately match the results of a detailed simulation of the system. The analytic model allows us to identify key parameters that affect PAROID performance, and propose an enhancement to the layout of data in PAROID which we show to perform superior to the original technique. We use both the analytic model and detailed simulations to illustrate the benefit of our new layout. This also demonstrates the significant simplicity of evaluating a new technique by applying a high-level model to the extracted trace statistics, compared to the current alternative of either implementing the new technique or simulating it at the level of individual requests.

I. INTRODUCTION

The power consumption of storage systems is a predominant component of overall data center power consumption, costing millions of dollars a year for large enterprises [5]. As a result, various techniques have been proposed to reduce the power usage of storage systems. These techniques typically involve opportunistically spinning down a subset of the existing disks when they do not serve I/O requests, and using data placement strategies (potentially including temporary copies) designed to increase the probability that disks can be turned, or remain, off. However, there are two major performance costs associated with power-efficient techniques. First, a smaller number of spindles reduces the effective bandwidth of the system. Second, techniques that

rely on migrating or copying data produce additional load that contends with the foreground workload for resources, again impacting performance. Thus, it is desirable that techniques provide their energy savings while minimizing their impact on performance.

Despite the importance of such techniques for power management, estimating the impact on the system of a specific technique is quite difficult, especially when the workloads are complex (e.g. containing various distributions of idle time, and a mixture of random and sequential I/O). On top of that, complexity is also introduced by the power conservation technique itself, which might require not only turning disks on or off, but also background copying or migration of data. Simulation is a well-accepted method of evaluating power-savings techniques, and can be used as an alternative to running the workload on an actual system that implements that technique. Typically, storage simulators use models of key system components (such as hard drive, RAID controller, etc.) to determine the effect on the system of each request in the workload trace, and have been shown to produce accurate results [4]. However, they require considerable effort to simulate new techniques, if those techniques can not be expressed in terms of the already existing building blocks of the simulator, and also require considerable simulation time. The difficulty of evaluating these techniques makes their deployment in practice less likely, thus reducing their potential utility.

In this paper, we make two key contributions. First, we present a novel approach to modeling the performance and energy of a storage system, that can be used for evaluating techniques aimed at power optimization. Our approach strikes a balance between analytical models that handle only well-defined and simple synthetic workloads [23], [16] and full-blown simulation at the level of individual requests [9]. Our approach is to use an analytic model of the system that can be applied to key statistics extracted from short sequences of requests. The entire trace is split into small time intervals, each interval is summarized through some key statistics, which in turn serve as input to our model to accurately evaluate that trace under various layout mechanisms and migration policies. By using our model, the task of evaluating a new

technique reduces to simply extracting these simple workload statistics once, which can then be reused across different instantiations of our model corresponding to different layout techniques.

The key abstraction in our analytical model is the *run-length*: a single sequential block of requests as seen by the disk; for example, a single I/O request of 128 KB will be treated as a run-length of 128 (the run-length can be in any granularity, as long as the same value is used throughout the model). Similarly, 16 consecutive requests of 8K each to sequential blocks would also be treated as a run-length of 128. By aggregating these various run-lengths in the trace, our model accurately accounts for the exact mix of random and sequential I/O. The model captures the effect of fine-grained queueing due to burstiness in the workload. Unlike synthetic workload models, our model faithfully replays the exact duration and extent of burstiness in the real-world traces. We show that even complex workloads with arbitrary mix of interleaved random and sequential I/O can be effectively modeled based on this simple insight of separating them into run-lengths and then aggregating the bandwidth and latency characteristics for each individual run-length.

We demonstrate the efficacy of our model by using it to model the various components of the PARAID system [29]. PARAID saves power by switching to a lower ‘gear’, i.e. lower number of disks, when the utilization is low; writes generated during that time to the shut-off disks are re-mapped to remaining active disks. These are propagated back to the original disks when the load again ramps up, at which point the array does a ‘gear up-shift’. We model the performance cost of this multi-gear operation, and validate it through detailed simulations using DiskSim [4]. We show that the results from the model match those of the detailed simulation quite accurately, thus demonstrating the accuracy of the model despite its conceptual simplicity.

Our second key contribution is proposing a new layout strategy for the multi-gear operation in PARAID, called *enhanced PARAID*, or *e-PARAID*. Specifically, instead of mapping blocks of shut-down disks at the latter half of the live disks in PARAID, we interleave the blocks at a finer granularity, thus reducing the seek incurred when accessing data that was originally in the shut-off disks. Essentially, the layout in the disks now leaves *buckets* at periodic intervals to accommodate data from the shut-down disks. By means of detailed simulations, we show that this technique is superior to the base PARAID technique from the viewpoint of performance, when conserving same amount of energy. Evaluations show that e-PARAID has 10-27% lower average response time than the original PARAID.

The generality of our modeling framework is evident from the fact that we represent different types of storage systems namely a single disk, a normal RAID array, power-aware RAID array called PARAID and our enhanced version of PARAID called e-PARAID. We were able to accurately

model these different storage types using our framework with straight-forward modifications. Our modeling technique can therefore be useful to evaluate the performance and power trade-offs of new energy efficiency schemes in a macro perspective, the results of which can enable one to make decisions whether to implement/simulate the new scheme for further fine-grained assurance. Overall, we find that our model is simple yet general enough to capture typical policies concerning the layout and migration of data from the viewpoint of power reduction in storage, thus providing a simple way to evaluate a variety of power-saving techniques in the context of a given workload. We also find that our enhancement to the layout strategy employed by PARAID works quite well under a range of workloads, alleviating the performance cost of the PARAID mechanism.

The rest of the paper is organized as follows: we present some background information on the current class of energy conservation techniques, and an overview on the PARAID system, in Section II. We introduce our model in Section III and validate it through comparison with detailed simulation in Section IV. We then present our new enhancement to PARAID in Section V. We discuss the overall goals and limitations of our model in section VI. We discuss related research in Section VII and conclude in Section VIII.

II. BACKGROUND

In this section we first present a short overview of several energy conservation techniques for storage systems, and then provide more details about one such technique, PARAID, which we use in the rest of the paper to demonstrate the efficacy of our performance and energy model.

A. Energy conservation techniques

Disks form prominent building blocks of modern storage systems deployed in data centers. Most modern disks have at least two power modes of operations: *active mode*, where the disk spins at full speed (and consumes maximum power), and *standby mode*, where the disk stops spinning completely (and consumes minimum power). The common philosophy behind conserving energy in storage systems is to switch the disk between active and standby modes opportunistically. One of the challenges in deploying an energy conservation technique is that often there is a direct trade-off between energy conservation and the system’s performance, reliability and availability; the time it takes to switch a disk between different modes results in reduced performance, temporary unavailability of data contents, temporary decrease in data resiliency, or (often) combinations of them. Consequently, power conservation techniques aim to reduce energy consumption in a way that still allows the system to maintain its service level agreements.

Based on the effect on performance or reliability of a system, energy conservation techniques can be broadly classified as (i) techniques that guarantee that at least one

instance of data is readily available in the storage system at all times and can be serviced immediately [32], [29], [17], [20] and (ii) techniques that do not make this guarantee [25], [7]. Techniques in the first class are generally deployed in enterprise data centers with stringent SLA requirements, as they avoid having data requests be blocked for the large amount of time required to bring a disk in active mode from standby. Techniques in the second class are commonly deployed in archival storage systems where data is very rarely accessed, and occasional delays in servicing data requests are acceptable. One idea proposed in the context of techniques in the first class above, is to use multi-speed disks [22]. Such disks can be used to save energy in the lower rotational speeds, and at the same time preserve availability of data. However, these types of disks are not yet commercially available on a large scale, and therefore this paper and our model focuses on techniques that leverage just the active and standby power modes of disks. However, our model is readily extensible to handle such variable rotation speeds as well.

In this paper, we focus on the first class of techniques, that guarantee quick servicing of all requests. Most of these systems aim at preserving peak performance and conserving energy whenever the load is minimal. For example, Hibernator [32] uses multiple tiers of variable speed disks and chooses the disk speed based on load characteristics. Similarly, PARAID [29] operates on RAID configuration constructed over varying number of disks, choosing the right configuration based on the intensity of the load. In order for the system to have all data available even when some disks are in standby mode, data needs to be moved or copied across disks within or across arrays. For example, write-offloading [17] directs writes to another storage array in order to increase/batch the idle time in its original array (with an implicit assumption that reads are rare and local enough to be served from cache). The process of data movement or copying introduces additional load to the storage array. Mitigating the impact of this background load, that contends with foreground load produced by applications, is an important design requirement for energy-efficiency techniques.

B. PARAID - A gear-shifting power-aware RAID

Here we provide some background on PARAID technique [29] that conserves energy in RAID arrays while preserving peak performance and reliability. We discuss in detail the PARAID system in this section because we use these concepts in the later sections of the paper.

PARAID exploits the fact that many workloads exhibit cyclic fluctuations with respect to load intensity and therefore it is not always necessary for the RAID system to run at peak performance (and power). The basic idea is to dynamically adjust the number of disks servicing requests based on load intensity. PARAID does this by converting a single RAID array into a set of overlapping layers –RAID arrays–, each constructed over a different number of disks. These layers

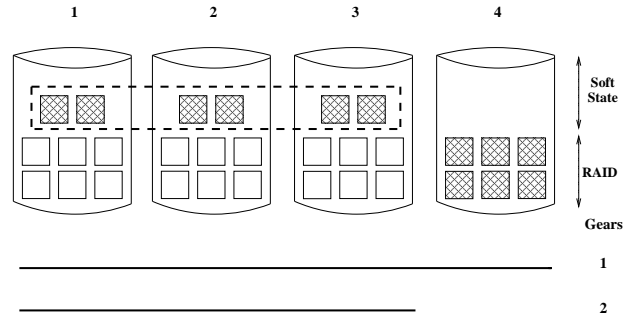


Fig. 1. PARAID system with two gears

are hierarchical in nature; the top layer will have the most disks, and the bottom layer will have the fewest. Each of these RAID arrays will hold sufficient data to service requests independently. These different layers of RAID arrays constructed on top of the original RAID array are called *gears*. The number of gears that can be supported by a system depends on the amount of desired capacity of the system. In order for the system to be capable of servicing the requests independently even in lower gears (when the array has disks in standby mode), the data is heavily replicated, and unused storage space must exist for this purpose. The space for replicated data is called *soft state* because it can be reclaimed later if it is needed. When the system operates in lower gear (with fewer number of disks), the remaining disks are switched to standby (low-power) mode, which provides opportunity to conserve energy.

The top-most gear in PARAID is synonymous to the original RAID array (with no power-awareness) in performance and data resiliency, since it is constructed from all the disks in the array. Therefore, whenever peak performance is needed the system shifts to the top-most gear. The lower gears contain fewer disks, with a soft-RAID constructed over part of their unused region. It is important to note that a soft-RAID is constructed only over the unused region, and not over the entire disks. Constructing a RAID over entire disks would require the entire data in the array to be reshuffled. In the case of PARAID, original data present in disks are still intact and the contents of the to-be-shutdown disks alone are copied to the newly constructed RAID. Therefore, additional data transfer during a gear-shifting process is proportional only to the updates that happened in the shut down disks after the previous gear-shifting process. Figure 1 shows an example of a PARAID system with two gear configurations. Gear 1 (top gear) includes all four disks while gear 2 consists of three disks. When the system shifts to gear 2, contents of the fourth disk are copied to the soft state region of the remaining three disks.

PARAID is implemented as a software layer that resides below the RAID controller layer and its job is to control the mappings done by the RAID controller. When the system is in lower gear, requests are initially mapped to the original

array by the RAID controller. The PARAID layer would then re-map to the running subset those requests intended for the shut down disks. One of the pertinent issues handled in PARAID is preserving the redundancy configuration across gears even though the number of disks reduces for each lower gear. To accomplish this, the RAID level used in all gears are configured to be the same. For example, if the top-most gear is RAID-5 over 8 disks, and if one of the lower gears is RAID-5 over 5 disks, the system can tolerate a single disk failure in either mode.

III. MODELING STORAGE ARRAYS

In this section we present our modeling framework and use that to derive performance and energy models for different class of storage systems.

A. Need for a new model

We present the need for developing a new model in two perspectives here – first, in comparison to existing models and next, in comparison to a system simulation.

With increasing deployment of energy-aware storage systems, the need for a performance model that accounts for energy implications increases as well. The energy conservation policies for RAID arrays often affect performance in complex ways, that may be difficult to represent with traditional models. For instance, most energy conservation policies result in movement of data within a RAID array [20], [7] or among set of RAID arrays [17]. While traditional performance models handle response time of application requests or RAID array throughput with good accuracy, they do not easily handle the additional request-load generated within the array due to the energy conservation technique used. Also, most traditional models assume synthetic workloads with well-defined distributions for inter-arrival times, locality of access, etc., and do not consider trace-specific distributions [23], [16]. Because of this, one needs to categorize their workload into a particular distribution in order to apply a model. We have developed a system that directly takes in a block-level trace and extract useful parameters from it, which are fed as input to our model. Our model can be used to predict performance (latency and bandwidth) and energy behavior of a system for a given block level trace.

Simulations of energy aware storage systems can also be used to accurately predict performance and energy behavior of a particular system for a given workload, e.g. Mantis[9]. However, using an analytical model can be a better alternative to a detailed full system simulation in many cases. First, simulating a storage system with new energy conservation policies can often be difficult than modifying a simple analytical model. Our experience with DiskSim is that implementing, correctly, even a straightforward algorithm such as PARAID is a substantial effort. In most cases, we would be more interested in predicting the trend in a macro perspective than in actual performance/energy numbers. These macro

predictions will be helpful in deciding if it is worth the effort to build such a system, or implement the technique for detailed simulation. Second, simulations are prone to implementation bugs and can lead to inaccuracies that may never be noticed. Whereas, in the case of analytical models the inaccuracies are invariably due to deliberate, tolerable generalizations and assumptions. Therefore, an analytical model can also act as a way to cross-check the correctness of a detailed simulation. Third, and more importantly, a model can provide intuitions about further improvements to a system. For example, the importance of each parameters in a model on the final result can be more easily visualized, which could also lead to ideas on how to improve the technique being modeled.

B. Model Overview

Given a trace, we model fine-grained latency and bandwidth over time and the overall amount of energy spent by the system in running the workload. This is done by splitting the input trace into time windows and modeling average latency of requests within each time window. By keeping the window size small, it is possible to model fine-grained latency of the trace requests. We model overall energy spent by the system from the latency values which includes the seek time, disk rotation time, etc., and from the number of power-mode switches the disks made for the entire trace. Energy values for each of these micro operations are obtained from disk-manufacturer specifications and they are aggregated to get the overall energy consumed by the workload.

Table I lists the basic set of parameters encountered in our model. Our modeling technique comprises of two phases. The first phase is called the fingerprinting phase, where we take as input a block-level request trace and extract some key parameters and statistics from it, by splitting the trace into small time intervals, each of size τ (in our validation results, we use the value of 100 ms). The parameters collected are namely $runcount[len]_{\tau}$, nr_{τ} and $dsize_{\tau}$ for each of the time windows. The $runcount[len]$ is a vector, each representing the number of requests of a particular size. For example, if in the current time window, there are four requests of size 8, 12, 8 and 32 blocks, the runcount values are such that, $runcount[8]=2$, $runcount[12]=1$, $runcount[32]=1$. The size of the block doesn't matter here as long as the same unit is used throughout the model. Also, the len value represented in blocks can be changed to any meaningful unit like bytes, stripes, etc., as long as it is consistent with the representation of other parameters in the model. Sequentiality of trace requests is directly captured by the notion of $runcount[len]$; a highly sequential workload will have lesser number of $runcount$ and larger value for len while a fully random workload will have large number of requests (or $runcount$) of small len . In addition to extracting the various trace characteristics, we also take into account the hardware configurations namely (i) the total number of disks

Parameter	Description	Units
τ	Time window size	seconds
N_{total}	Total number of disks in the array	–
N_{active}	Number of disks in the array that are currently spinning	–
N_{sleep}	Number of disks in the array that are currently spun down	–
max_len_τ	Size of the largest request in τ	bytes
$dsize_\tau$	Total data transferred in window τ	bytes
sd_τ	Average Seek Delay for window τ	msec
rd	Average Rotational Delay	msec
st_τ	Total seek time in window τ	msec
rP, oP, sP	rotational, operating, and seek power	Watt
E	Total energy consumed	Joules
L_τ	Average request latency in window τ (wait times accounted)	msec
ART_τ	Average Response Time in window τ (oblivious to wait times)	msec
$runcount[len]$	Vector of request counts of various sizes	–
Seq	Per-disk sequential bandwidth	blocks/second
nr_τ	Number of requests in each time window	–

TABLE I
MODEL PARAMETERS

in the array, (ii) number of disks that are active in the current window, (iii) number of disks that are in standby mode in the current window, (iv) stripe-unit size, (v) average sequential bandwidth of the disks used and (vi) internal disk parameters like head switch, track switch, head settle, full stroke and single seek times.

The second phase of our modeling technique is deriving performance and energy models for the storage system with the help of parameters collected in the fingerprinting phase. First, we start with considering a storage layer with only a single disk. As the $runcount[len]$ vector captures the degree of sequentiality and the exact mix of random versus sequential workloads, we introduce seek and rotational delays for random workloads and consider the disk's average sequential bandwidth parameter seq for sequential workloads. Substituting these values for the statistics obtained over a single time interval gives the total time needed by the disk for executing all the requests in that interval, et_τ .

$$et_\tau = \sum_{len_\tau=1}^{max_len_\tau} runcount[len]_\tau \left(\frac{len_\tau}{Seq} + sd_\tau + rd \right) \quad (1)$$

where, sd_τ is the average seek delay for window τ and it is a function of the seek distance between requests in the window τ . Seek time is computed as $settle_time$ for very short seek distances (in the order of less than 10 cylinders), proportional to $\sqrt{seek_distance} + settle_time$ for short seek distances (in the order of 500-1200 cylinders), and proportional to $seek_distance + settle_time$ for longer seek distances. This seek model was proposed by Ruemmler [21] and we found that the latency values are reasonably accurate with this model.

Therefore, the average response time for every requests in the time interval is given by,

$$ART_\tau = \frac{et_\tau}{nr_\tau}$$

However, wait times or request arrival rate is not accounted when calculating the average response time in the above equation. In order to model this, we assume that requests arrive at a uniform rate within a time window. Note that we assume uniform request rates only within a single time window(100ms) and not the entire trace. As the time window size is reasonably small, near-accurate request rates are captured. Let t_τ be the inter-arrival time between requests in a window τ , computed as τ/nr . Therefore, average request latency within a time window after considering the wait times is given by,

$$L_\tau = \frac{1}{nr_\tau} \sum_{i=1}^{nr_\tau} [(i \times ART_\tau) - ((i-1) \times MIN(ART_\tau, t_\tau))] \quad (2)$$

Therefore, if the amount of data accessed during τ is $dsize_\tau$, storage bandwidth for that time interval is given by,

$$bandwidth_\tau = \frac{dsize_\tau}{et_\tau} \quad (3)$$

Equations (2) and (3) represents the average request latency and average storage bandwidth values for a single time window. Applying this model for the entire trace gives a latency and bandwidth curve that varies over time in the granularity of τ .

To model the storage energy consumption for a given trace, we start from the performance model described above, and find the aggregate seek time, operating time and rotation time. In the case of a single disk storage system, the rotation power is the entire period of the trace, operating power is the elapsed time et_τ aggregated over all time windows and seek time is the aggregate seek delays from each of the time windows. We combine the timing measurements for each of these micro-operations with their corresponding power values, which are obtained from the detailed disk specification. Therefore, the

total energy consumed by the disk in running the workload is given by,

$$E = \sum_{\tau=1}^{max} [\tau \cdot rP + st_{\tau} \cdot (sP - oP) + et_{\tau} \cdot (oP - rP)] \quad (4)$$

where, rP , sP and oP are rotational power, seek power and operating power respectively and st_{τ} , et_{τ} are total seek time and elapsed time for a time window τ . Seek power is the power measured when the disk moves its disk head seeking a particular sector and rotational power is the power measured when the disk spins. In a system devoid of energy conservation techniques, the disk spins all the time (even when it is idle) consuming rotational power. Operating power is the power measured when the disk is busy servicing requests. The equations (2), (3) and (4) shows the performance and energy of a storage system containing a single disk, when running a given workload.

We now see how the above equations get tailored by making very little modifications to express the bandwidth of more specialized systems like RAID and PARAID.

C. Modeling RAID

We now use our modeling framework to represent the performance and energy consumption of a RAID array. Consider a RAID-0 constructed over N_{total} disks. Since all the disks are in active mode, we have $N_{active} = N_{total}$. Assuming that every request is striped across the entire RAID array, the RAID bandwidth is the aggregate bandwidth of all the disks in the array, capped by the RAID controller bandwidth. We therefore find the bandwidth achieved by individual disks, aggregate it and then cap the value by a static parameter, max_ctrl_bw . The elapsed-time calculation in this case is similar to equation (1)

$$et_{\tau} = \sum_{len_{\tau}=1}^{max_len} runcount[en]_{\tau} \left(\frac{len_{\tau}/N_{active}}{Seq} + sd_{\tau} + rd \right) \quad (5)$$

Equation (2) accounts for wait times where the requests are queued to get serviced from a single disk. The same technique can be used in the case of RAID also, by considering the requests queued at the controller level. Therefore, based on the time taken by the RAID array to execute requests within a time window, the average response time ART_{τ} is calculated, which in turn is used to model the wait times of each requests. The bandwidth of the storage array is given by,

$$Bandwidth_{\tau} = \min \left(\frac{N_{active} \times dsize_{\tau}}{et_{\tau}}, max_ctrl_bw \right) \quad (6)$$

The ratio between the size of τ and the et_{τ} represents device utilization. Though we have modeled the case of RAID-0, our modeling framework can be used to represent more

sophisticated RAID configurations like the those involving parity. For the sake of simplicity and space constraint, we have shown the case of RAID-0.

From equation (4), the total energy consumed by a disk array with N_{total} disks can be calculated by aggregating the energy consumed by each disk. This is because, disks consume majority of power in a raid array when compared to controller's processor and memory. The energy formula for RAID array is therefore sum of energy consumed by individual disks in the array and is represented as,

$$E_{RAID} = \sum_{d=1}^{N_{total}} E_d \quad (7)$$

where E_d is the energy consumed by each of the disks and it is calculated using equation (4).

D. Modeling PARAID

The PARAID system operates in different RAID configurations (termed as *gears*) and dynamically switches between them based on system load. The different gears in PARAID constitutes to RAID arrays containing different number of disks and all gears follow the same RAID policy. For example, if there are two gears with number of disks N_{total} and $N_{total} - m$, both the gears are constructed in a same RAID level. More details about the working of PARAID is given section II. When PARAID operates in the top gear, it performs exactly like a normal RAID array with $N_{active} = N_{total}$. When operating in a lower gear with $N_{total} - m$ disks, $N_{active} = N_{total} - m$ and requests that come to m disks are re-mapped to a reserve location in N_{active} disks. Therefore just changing the effective number of disks used in equation (5) cannot give a correct estimate on the performance of PARAID during low-gear operation. Because of re-mapping of data that go to m disks, additional seeks occur in the N_{active} disks. Considering a per-disk perspective, chances of a disk incurring a seek to access a data pertaining to m disks is m/N_{active} . We capture these increased seeks by imposing a cap on the len of a request when extracting the $runcount[en]$ values from the trace. The cap for which the request length is limited is given by, $N_{total} - m$.

The parameter sd_{τ} used in equation (5) is calculated from seek distances between trace requests in a time window. It cannot be used unmodified for modeling PARAID. This is because the data corresponding the m disks are placed in a reserve space on disk which is usually at an increased seek distance from the actual data footprint of the disks. We assume that the reserve space in every disk is situated at a seek distance of $max_distance/2$. This increased seek distance applies only for the requests that are re-mapped to the reserve space on disk. First, sd' for seeking a distance of $max_distance/2$ is calculated by using the seek model described in [21]. A weighted seek delay can be represented using the formula,

$$sd'' = (\delta \times sd') + ((1 - \delta) \times sd)$$

where, δ is the fraction of requests that were mapped initially to the m disks. This fraction can be obtained by a single pass through the trace and finding the number of requests whose length is more than the product of stripe-unit size and the number of disks in active mode.

Substituting the above seek time value in equation (5) and applying (2) and (3) would give the latency and bandwidth of PARaid when running in a specific gear, either low or high. In order to model real-time gear shifts, we include effects of background data movement between gears. This is important to model because, effects of background data copy on foreground requests impact on how fast the gear shifts, which in turn impacts overall benefits of PARaid.

During every gear shift operation, new data that was written after the previous gear-shifting operation are alone copied to destination gear. For example, during down-shifting process (from N_{total} disks to $N_{active} = N_{total} - m$ disks) data that was written in m disks after the previous up-shift process alone are copied because data written prior to the previous up-shift would already be present in the N_{active} disks.

In order to model these extra data movements across different gears, we make an assumption that the updates to be propagated to the destination disks are laid out in a contiguous manner. The performance model can be used to track the utilization of the array in each time window. Whenever a sliding window of past utilization values increases beyond a threshold, the model marks the next time window with a gear-shift operation (in this case, down-shifting). When a time window is marked for gear-shift, the amount of data that were written to the m disks after the last gear-shift process is calculated. This can be done by making a single pass through the trace from the current time window backwards to the previous time window that was marked for gear shift. The fraction of these update requests versus foreground requests from the trace in the current time window, the parameter ρ is calculated.

As we assume that the updates that need to be propagated to the destination disks are laid out contiguously, the $runcount[len]$ will be 1, where the len is the entire update footprint. However, copying the entire updates in one operation might affect performance of foreground operations in the destination disks. In order to avoid it, we specify a cap to the length of each operation which makes $runcount[len] = updatesize/cap$, where maximum value of len is cap . Taking into account the update requests, we alter sd'' value as,

$$sd''' = (\Delta \times sd') + ((1 - \Delta) \times sd)$$

where, $\Delta = \rho + \delta$

The amount of update data can also be divided into chunks and coalesced into more than one time windows, in which case the fraction ρ will be altered to affect all those time windows. The seek distance for update operations is considered as $max_distance/2$ for both high-to-low shift and low-to-high shift because, in both cases, the remote disk area is either read or written (which involves increased seeks). We coalesce the extra update requests into $runcount[len]$ parameter to account for extra data transfer due to background update operations. We call the modified $runcount[len]$ as $runcount'[len]$.

$$et_\tau = \sum_{len'_\tau=1}^{max_len'_\tau} runcount'[len]_\tau \left(\frac{len'_\tau / N_{active\tau}}{Seq} + sd'''_\tau + rd \right) \quad (8)$$

The above equation gives the time taken for the PARaid system to execute the requests from the trace in the current time window along with a fraction of update requests due to gear-shift operations. Therefore, the above formula is used for every time window and when the system is in high gear, it reduces to equation (5), and depending on whether there is a gear shift operation in the current time window, the run-length vector and the seek delay parameter are chosen.

The average response time of trace requests in the current time window is given by the total time taken for both trace requests and background update requests, over the number of trace requests. This value is used in equation (2) to find the average per-request latency of the trace requests in the current time window. The PARaid bandwidth can be represented by equation (6), however with a different et_τ value.

Energy consumed by PARaid cannot be modeled by just aggregating the overall seek times and latencies for each time window as in the case of RAID. This is because, number of disks changes throughout the trace whenever the gear shifts. Another problem is, the energy consumed for disk spin-up has to be accounted separately as it is predominant. Here we explain the process of modeling gear shifts with respect to time in detail and then provide the energy model for PARaid system.

PARaid shifts gears based on the device utilization values during a past time interval. It may start off in the highest gear (with all disks in active mode) and then when the utilization value falls below a threshold, the gear is down-shifted. Similarly when utilization value exceeds a threshold, the gear is again up-shifted. We model gear-shifts in the same way. For every time window τ , the elapsed time is calculated using equation (8) and the ratio between time window size and the elapsed time represents device utilization. We extract this parameter from the workload and if moving averages of the utilization falls below a lower threshold or exceeds a higher threshold, the next time window is marked for gear-shift. The total number of gear shifts for the entire trace is extracted and used in the energy model.

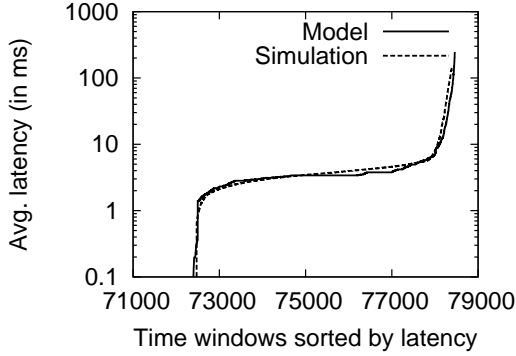


Fig. 2. Workload: 'usr0', System: RAID

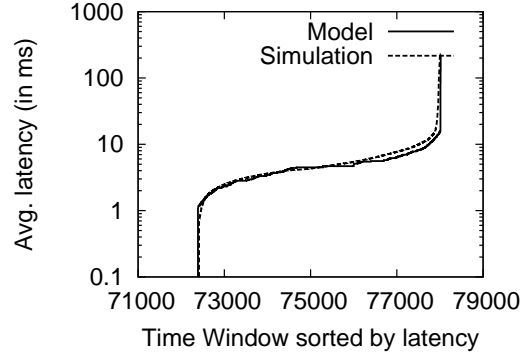


Fig. 4. Workload: 'usr0', System: PARAID

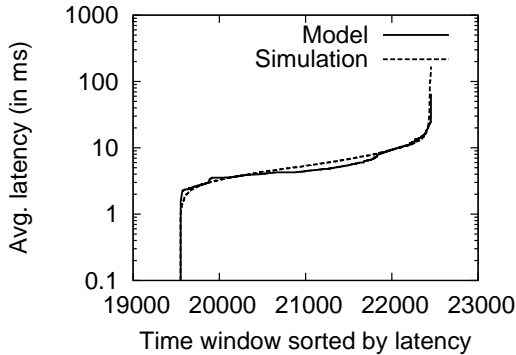


Fig. 3. Workload: 'prn0', System: RAID

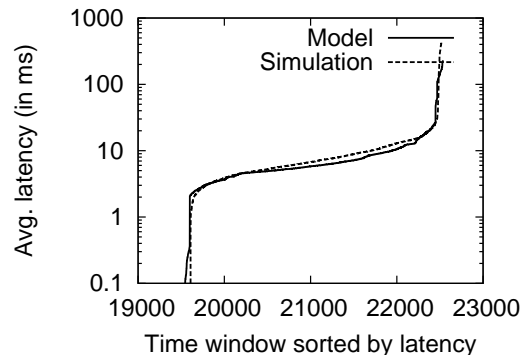


Fig. 5. Workload: 'prn0', System: PARAID

$$E_{PARAID} = \frac{N_{gs}}{2} \cdot n \cdot spE + \sum_{\tau=1}^{max} (N_{active(\tau)} \cdot E + \tau \cdot n \cdot iP) \quad (9)$$

where, $N_{active(\tau)}$ is the number of active disks during time window τ , n is $N_{total} - N_{active}$ for time window τ , N_{gs} is total number of gear shifts over the entire time period of the trace, spE is the spin-up energy, which is the product of spin-up time and spin-up power. E is per-disk energy consumed that can be calculated from equation (4) and iP is the power measured when the disk is in standby mode.

IV. VALIDATION OF THE MODEL

In this section, we validate the representations for RAID and PARAID systems, derived from our modeling framework, by comparing it with its simulation. We pick two real-world disk traces for most of these results [17]. The first is *usr0*, a trace extracted from a server in Microsoft Research Cambridge that contained user home directories. A trace segment containing 50K requests containing 62% reads and average request size of 22KB was extracted. The second trace is *prn0*, a trace from a print server at MSR. We extracted a segment containing 50K requests out of which, there are 48% reads and average request size is 26KB. The metric we use to compare the model with the simulation is the

latency profile of requests in the trace. Given that the purpose of the model is to get a sense of the average aggregate performance, and not quite per-request response times, we adopt the following technique while comparing latencies. We divide the temporal axis into intervals of 100ms (size of a time window) each. Within each interval, we find the average latency predicted by both the simulation and our model. We believe that for the purposes our analytical model is intended to, such an average metric is sufficient and appropriate. For easy readability, we sort time windows based on their average latency values while plotting. Overall we find that our model does a very good job of estimating the performance of both RAID and PARAID.

A. Simulation environment

We use DiskSim simulation environment [4] for simulating both RAID and various components of the PARAID system. DiskSim has been shown to quite accurately model the behavior of complex storage policies, and is thus a good candidate to compare our model with. Our baseline system is a RAID array of 8 disks

We simulated gear up-shift and down-shift operations in PARAID, and update propagation while moving across different gears. Specifically, we use just 2 gears: the lower gear uses 6 disks while the higher gear uses the full 8 disks.

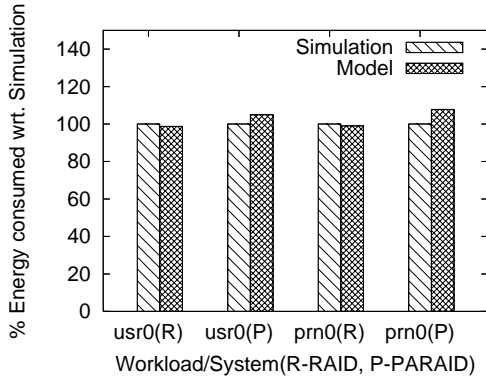


Fig. 6. Energy Model Validation

During normal operation, we keep track of the updates that go to the *transient disks*, i.e. the disks that would be shut down during lower gear. To decide on the right gear to use, we monitor the utilization of the array as the workload progresses over a sliding time window. Once the utilization drops below a certain threshold, we reduce the gear and start copying the set of blocks that were written to the transient disks. We re-map those blocks to the other 6 disks by striping them uniformly across the remaining 6 disks. Similar to PARaid, the location of these "foreign blocks" would be at a reserve location in the other disks, namely at the second half of the individual disks. To provide a simple example where half the disk is allocated for such foreign blocks, let us consider that the disk had 100 blocks. Out of this, the first 50 would be reserved for original array blocks, and the remaining 50 would be allocated for migrations from transient disks. Once disks 7 and 8 are shut down, the 200 blocks in those disks would be uniformly striped across the remaining 6 disks. If we view disk 6 and 7 to form a logical 200 block space, block k in that space will be mapped to block $50 + k\%6$ on disk $k/6$.

During the degraded lower gear operation, we again keep track of the writes to these foreign blocks. Once the utilization again increases beyond a threshold where the lower number of disks is insufficient, we decide to restore the transient disks back into normal operation. At this point, we propagate the updates to those foreign blocks back to their original locations.

We use this simulation to extract per-request latencies, and then compute the average of these per-request latencies at time window granularity of 100 ms each. Since our model does not yet take disk buffering into account (we plan to address this in the future), we also turned off buffering in DiskSim.

DiskSim is basically a performance model. We extracted fine-grained timing measurements like total seek time, total access time, etc., from the simulation and used the disk power values obtained from detailed disk specification, for those fine-grained operations to get the total energy consumed by

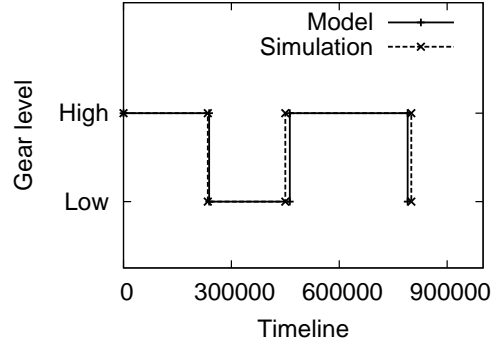


Fig. 7. Gear-Shifting Accuracy of the Model

the system.

B. Comparison with the model

We now compare the performance and energy values from our simulation in DiskSim with the values obtained from our analytical models for both RAID and PARaid systems. The goal of this section is to validate the representation of RAID and PARaid systems derived from our modeling framework against fine-grained, request-level simulation. One of the important parameters we extract from the trace is the run-length vector. We extract the run-length vector at various time intervals of 100ms each, and model the average latency within each interval. While extracting run-lengths within a 100ms interval, we account for disk scheduling effects by sorting the block numbers and collating I/O to blocks which are not necessarily contiguous in the trace, but occur in quick succession that the disk scheduler would have interpreted them as sequential. We then feed these run-lengths to the formulas specified in the section III to come up with latency estimates. Note that the process of extracting these run-lengths and other parameters in the model has to be done only once per trace. After the trace is "fingerprinted" once with this technique, that fingerprint can be used repeatedly to evaluate it for various layout strategies by simply changing the model to depict that strategy.

We first compare a regular RAID system of 8 disks in both the simulated setup and in our model. Figures 2 and 3 show the comparison of sorted average latencies from both the simulation and our model. The y-axis shows average latency values over a time window of 100ms. The x-axis is the time window sorted with respect to their average latency values. More than 85% of the time windows have zero latencies as they don't have any requests issued in those periods. Therefore, for better readability we start the x-axis nearer to the point where the average latency values are non-zero. Our model does a very good job of modeling the latency profile for both the traces. Not only are the shapes of the curves similar, they match almost identically in terms of their amplitude. This shows that despite the simplicity of the

model, it can match the accuracy of a full-blown simulation which accounts for numerous dynamic effects such as head movement, disk scheduling, etc.

Next, we validate our model for PARAID by comparing its results to the simulated PARAID system. During the run of the traces in PARAID, there were gear up-shifts and down-shifts that happened based on utilization, thus resulting in extra I/O being generated due to the write propagations. We plot the gear-shifting events as captured by the simulation and our model for "usr0" workload. Even in that setup, our model does a pretty good job of predicting the latency profile ; again the sorted latencies for the two traces match quite closely for both the simulated and the modeled version of PARAID as shown in 4 and 5. The model calculates the utilization values from the bandwidth and based on a threshold value for utilization, the gears are shifted in the next time window. From the graph 7 it can be seen that our model performs well in estimating gear-shift events.

Finally, we compare the energy consumption values arrived by both the simulation and our energy model. Both in the case of simulation and modeling, we use disk manufacturer's specification of energy values for each of the micro-operations like seek, spin-up, idle, busy, etc., In the case of simulation, DiskSim gives accurate profile of seeks, rotation, spin-up and spin-downs where we substitute the energy values. In the case of model, we substitute those energy values in the model parameters that represents each of these micro operations. Fig 6 shows how close our model performs with respect to simulation values. We have energy values from the simulation as a baseline value(100%) and show the error rate of our model. The error rate is the ratio between the total energy consumption value obtained from the simulation and our model. RAID model performs very well when compared to simulation with less than 3% error. The PARAID model is also reasonably accurate with at the most 7% error rate. For both workloads for PARAID, our model overshoots the energy value because we have assumed that all seeks to remote location in disk would incur a seek distance of $\text{max_distance}/2$, while in practise, the seek distance may be lesser than that. One of our future work on this area is to develop a better model for representing the seeks to remote location in disk.

V. E-PARAID : INTERLEAVED LAYOUT

The performance model for PARAID makes it clearly visible that one of the parameters that significantly affects low-gear performance of PARAID is the extra seek incurred when accessing data from remote location on disk. Based on insights provided by our performance model, we propose an enhancement to PARAID system that improves the performance of low-gear operation of PARAID while conserving the same amount of energy. We call our enhanced version of PARAID as e-PARAID.

A. Motivation

The PARAID system conserves energy by dynamically reducing the number of active disks at any instance based on the load intensity. It conserves energy when running in a lower gear mode with just a subset of the total number of disks. While in this mode, the requests that would have normally been mapped to one of the standby disks are re-mapped to the already active disks. This is possible because, during gear-shifting process, the contents of the disks that are about to be shut down are copied to the unused areas in the active disks. In the original PARAID design, this reserve space is found at the end of the disk so that the re-mapping function is straight forward and the reclamation of the reserve space is done with ease as and when needed. For the ease of explanation let us consider the native content of the disk as *native data* and the contents of the shut down disks that are copied to the reserve space as *foreign data*. One of the significant issues in having such a design is that, the disks incur increased amount of seeks when servicing a request whose content spans the native data and the foreign data. When a sequential request arrives at the array, the disks that are currently active need to seek back and forth to gather the native data and the foreign data that is located in a remote location on disk. Even with efficient disks scheduling, the disk head needs to move over a larger area on disk. Instead of allocating a separate chunk of reserve space for foreign data, seek distance can be reduced by co-locating the reserve space with the primary data space. More complex layout strategy needs to be employed to implement this.

B. Design

We propose a new layout methodology to place the foreign data such that the overall seek overhead due to the movement of disk head between the native and the foreign data is reduced. Instead of allocating one full chunk of space in the end of disk to place the foreign data, we introduce a concept of using *buckets* that are spread across the entire disk. A bucket is a small chunk of free space in the disk that are placed equidistant to each other. The space between two consecutive buckets are used to store the native data and the buckets are used to store the foreign data. The size of the bucket and the space between any two consecutive buckets must be carefully chosen to avoid decrease in performance of accessing the native data during a high-gear operation. The space between two consecutive buckets should be big enough to avoid a sequential request accessing a native data from seeking across a bucket when the disk is in high-gear mode.

It may be perceived that spreading out the effective space for foreground requests by introducing buckets would reduce the performance of the foreground requests when the system operates in high gear. However, as there is sufficient chunk of space between the buckets, the foreground sequential performance is not affected. One of the important advantage

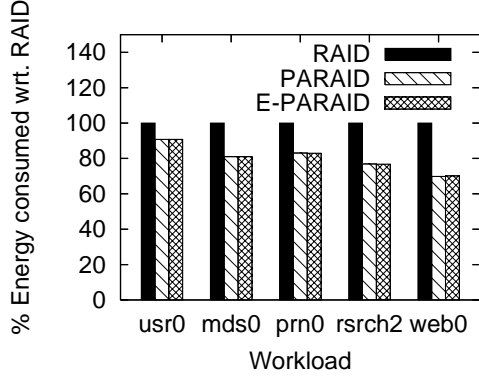


Fig. 8. Energy Consumption Overhead in e-PARAID

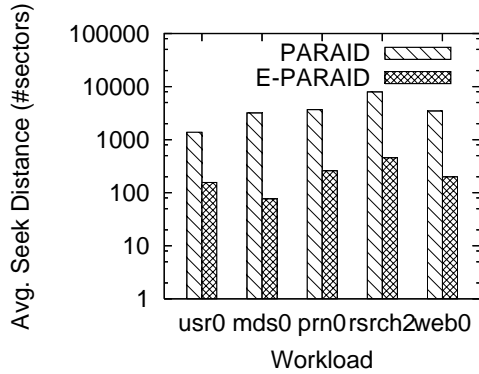


Fig. 9. Reduction in Seek distance in e-PARAID compared to PARAID

of using this approach is, the foreign data and the native data near which it is placed are often correlated. Consider a sequential request that was written to a 5-disk RAID 0 array when it operates in the high-gear mode and assume that the request occupied a few stripes in the original RAID. When the system shifts to a lower gear by shutting down the disk number 5, the data in the 5th disk is laid over the other 4 disks in such a way that native and the foreign data that corresponded to a same stripe while in the higher gear are located close to each other. When mapping the foreign data from the disk 5 to the other disks, the buckets are chosen in such a manner that the correlated data are co-located.

C. Modeling e-PARAID

As we interleave the reserve space among the primary data region on the disk, the extra delays arising from seeking to far off location on disk is avoided in the case of e-PARAID. Our modeling framework that we introduced in section III can capture this enhancement without much change in parameters. In the case of original PARAID model, equation (8) has a parameter called sd'_τ that accounts for increased seeks in lower gear by applying a weighted average of computed seek values and extra seeks due to re-mapping. For e-PARAID, we change that parameter to just sd_τ which is the seek delay computed for time window τ as a function of seek distance by

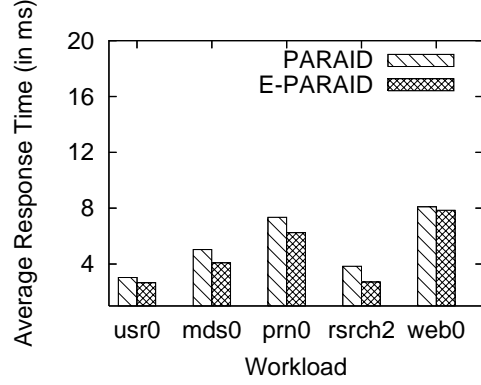


Fig. 10. e-PARAID improvement in average response time

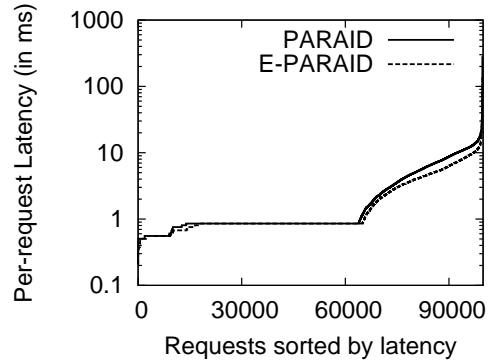


Fig. 11. CDF of per-request latency

using Ruemmler's [21] seek model, since both the original requests and re-mapped requests are co-located. Therefore, the elapsed time for requests issued in time window τ is given by,

$$et_\tau = \sum_{len'_\tau=1}^{max_len'_\tau} runcount'[len'_\tau]_\tau \left(\frac{len'_\tau / N_{active} \tau}{Seq} + sd_\tau + rd \right) \quad (10)$$

where $runcount'[len'_\tau]_\tau$ represents run-lengths from both trace requests and background data copy requests (during gear shift operation) in a time window τ .

The ability to represent a new layout strategy with the help of our modeling framework shows the generality of the framework and the ease of comparing different energy efficiency schemes in a macro perspective.

D. Evaluation

We now show two set of experiments to evaluate our new layout methodology. We first present the simulation results comparing the performance and energy consumption of the enhanced version of PARAID with the original version of PARAID. We then evaluate the accuracy of our model in estimating the performance of e-PARAID by comparing it with simulation results. We have used five different traces

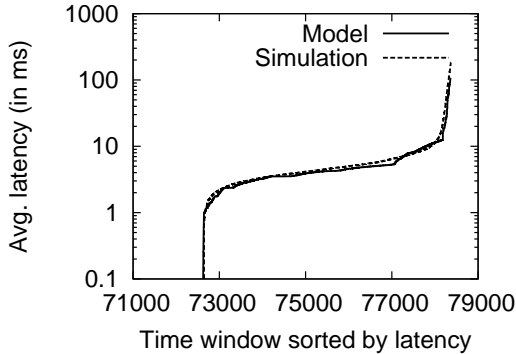


Fig. 12. Workload: 'usr0', System: e-PARAID

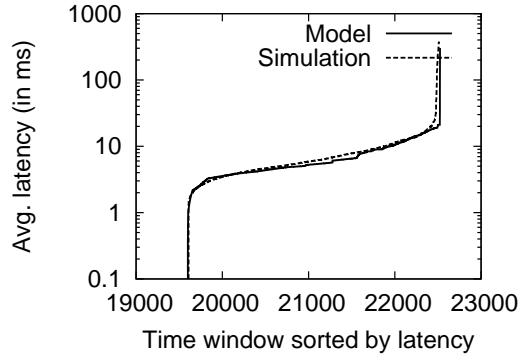


Fig. 13. Workload: 'prn0', System: e-PARAID

collected from the servers of Microsoft Research, Cambridge [17]. We have already discussed about the details of the trace *usr0* and *prn0* in section IV. We use three more traces collected from MSR Cambridge servers. *rsrch2*, *mds0* and *web0* are traces collected from servers containing project files of MSR researchers, a media server and a web server respectively.

We simulated our new layout methodology by modifying the RAID controller part of the DiskSim simulator. We intersperse every mapping made by the RAID controller and re-map it according to our layout. We fixed the chunk size between buckets as 32 stripe-units, which is large enough to not affect significantly performance in top-gear, but small enough to improve locality in lower gears. The bucket size is computed as $chunksize \times (m/N_{active})$. Figure 8 is a graph showing relative energy consumed by three types of systems namely unmodified RAID, unmodified PARAID and our enhanced PARAID. We have shown the % energy consumed by PARAID and e-PARAID compared to the baseline RAID system. This shows our performance enhancement routine did not adversely affect the energy conservation of the original PARAID. We then measure the performance of the e-PARAID system by using the five different workloads. In order to visualize the direct benefit from our enhancement, which is reduction of the seek distance when the system operates in lower gear, we plotted a graph (Fig: 9) that shows the average seek distance of our system when compared to the original PARAID system. We captured the seek distance when both the systems were in low-gear mode. In the simulation of the original PARAID, we allocated the reserve space near middle region of the logical address space while the native data of the disks stayed in the first 20% of the logical address space. We also measured the CDF of per-request latency for one of the representative traces namely *rsrch2*, which was extracted from a server from MSR Cambridge that hosted projects of researchers at MSR. We cropped the trace for a 10-hour period when the load intensity was lesser and as the result, the system operated in lower gear for most of the time. Figure 11 shows that e-PARAID

is faster than PARAID in servicing the requests in lower gear mode. Figure 10 is a graph plotted with average per-request response times of PARAID and e-PARAID. This shows that our enhancement consistently performs very well in low-gear mode when compared to traditional PARAID.

The second phase of our evaluation is validation of our model of e-PARAID. We compared the results of e-PARAID performance model with its simulation. Figures 12 and 13 shows sorted average latency values given by our model and detailed simulation for the traces namely *usr0* and *prn0*. Our model gives a reasonable estimation on the performance of the e-PARAID system with just altering the seek distance parameter to account for co-located data placement.

Therefore, the enhancement proposed to PARAID conserves the same amount of energy as the original PARAID system, while providing better performance in low-gear operation.

VI. DISCUSSION

The goal of our model is to get as close as possible (without compromising on simplicity) to an accurate simulation of the system being modeled, thus allowing system designers the ability to quickly reason about specific design choices before actually implementing them. While we believe we have gotten close to this goal in many respects, there are some issues that are still not addressed in our model. The first such issue is storage controller level caching. Note that the input to our model is a block-level trace, the requests of which are already misses from file system level caches. In many cases, the storage cache is much smaller when compared to host level caches and therefore modeling storage-level cache is not as important as modeling the file system cache in a full-system model. Given the dynamic nature of caching, it would be hard for any detailed analytical model to accurately model fine-grained cache behavior. Incorporating a coarse-grained notion of caching would be feasible in our model provided some high-level parameters about cache-hit rate for the workload, etc., By providing such parameters as input to our model, we can avoid the complexities of integrating

an accurate cache model in our framework. Finding an input parameter like cache-hit rate can be done off-line by a simple cache simulator, given a workload and system parameters. As we focus more on internal disk-level interactions arising because of power-conservation schemes, we did not incorporate disk-level caching into our model.

The other missing-feature in our model is that we do not model RAID configurations with parity; the update behaviors on parity disks would be different from the data disks. Modeling parity disks within our model is fairly straight-forward; we would need to incorporate the notion of partial stripe writes into our run-length computation. The current model is meant to be illustrative and is easily extensible to support these refinements. Overall, we believe that our modeling framework strikes a good balance between simplicity and accuracy by expressiveness.

VII. RELATED WORK

We present the related work on two major categories namely, (i) the different types of energy conservation policies and (ii) performance models that were proposed for RAID based storage systems.

One of the common energy conservation algorithms is to switch the disks to standby mode after it is idle for a particular time period. Many research projects [7], [8], [13], [11], [18], [24], [33] have used and analyzed this methodology often called as the Fixed threshold algorithm. More recent research has been towards the direction of multi-speed disks. Gurumurthi et.al. [22] was one of the first to propose the multiple speed disk model and later many systems [32], [10], [19] used it to develop sophisticated energy conservation policies. These techniques perform very well, but the downside is that multiple speed disks are still not commercially available in a large scale. Our analytical framework can be used to model multi-speed disks by manipulating with the effective seek delay values. PARAID [29] was one of the first systems that proposed an energy conservation policy for RAID arrays constructed over legacy disks. One of the most recent project on enterprise-level power conservation was proposed by Narayanan et.al. [17] where the idle times in one server can be extended by offloading its writes to another server temporarily.

There is a large body of work on analytical performance models of RAID based storage systems. Varki et.al. [28] developed an analytical model that accounts for caching effects and disk parallelism along with the effects of controller-level optimizations for performance of the disk array. Prior to their work, Uysal et.al [27] presented a model that predicts the throughput of the disk array taking into account the mean queue length of the disk, implementation of the controller optimizations etc., They have a parameter called run-count which is the number of contiguous requests with consecutive block numbers. This parameter represents the degree of sequentiality in their workload. Our notion of $runcount[*len*]$

values is similar to theirs; the the main difference is that they assume an average request size, with run-count just reflecting the number of requests that exhibit sequential locality. Bitton and Gray [3] modeled seek times in mirrored disks arrays. Chen and Towsley [6] proposed an analytical queuing model for the response time of disk arrays in several of their papers. Bachmat [2] modeled the reconstruction performance of RAID 1 arrays. While these work mainly were focused on RAID 1/0 arrays, Matt and Menon [14] modeled the response time of RAID 5 arrays and later, Kuratti and Sanders [12] and Thomasian [26] modeled the normal, rebuild and degraded mode of the RAID 5 arrays. Merchant and Yu [15], [30] presented many queuing models for response time of RAID 1 and RAID 5 disk arrays. While all these works model the performance of disk arrays to a reasonable accuracy, none of these models are tailored for power aware storage systems. Our model is one of the first analytical models, to the best of our knowledge to model the performance of power-aware storage.

There are also a few power models proposed in the past. Mantis [9] system generates power model by correlating a few key system/workload parameters with the actual power measurements. It models the power consumption of the entire system and focuses more on understanding the power consumption behavior of each physical components of the system. However, our model focusses only on the storage layer and aimed at easier comparison of various storage energy conservation schemes. Dempsey [31] on the other hand provides accurate power and performance measurements in the granularity of individual seeks and rotations for every request that is serviced by the disk. While Dempsey provides accurate power consumption values, it is a fine grained simulator than an analytical model. In order to compare different energy efficiency schemes with Dempsey, one needs to implement the technique in the simulator, which is often a difficult job. STAMP [1] is a recent modeling technique where front-end request characteristics are fed as input to the model and it is converted to fine-grained back-end disk activities like seeks. However, they do not take into account the layout mechanism when translating front-end requests to back-end activities. Also, the front-end request characteristics fed to the model are more course grained than our technique.

VIII. CONCLUSION

In this paper we have proposed a new model framework to easy derive representations for performance and energy behavior of energy efficient storage systems. We have validated the model with the simulation results for each of the modeled systems and we show that our new model performs a good job in estimating performance and energy behavior of RAID-based storage systems.

We have also proposed a new layout methodology for an existing energy conservation scheme called PARAID and we have shown that our enhancement performs better than

the original PARAD system in terms of performance, while saving the same amount of energy as the original system. This also shows the usefulness of our model in gaining insights that lead to improvements to existing systems and its ability to model new layout strategies without much complexity.

ACKNOWLEDGEMENTS

The first and second authors are partially supported by grants from NSF CISE NetSE program, CyberTrust program, an IBM SUR grant, an IBM faculty award, and a grant from Intel Research Council. We thank Eric Kruus, Anirudha Bohra, Akshat Aranya, Salil Gokhale and Sean Noonan for their valuable contributions. We would like to thank the anonymous reviewers and our shepherd Parthasarathy Ranganathan for their comments which helped improve the quality of this paper.

REFERENCES

- [1] Miriam Allalouf, Yuriy Arbitman, Michael Factor, Ronen I. Kat, Kalman Meth, and Dalit Naor. Storage modeling for power estimation. In *SYSTOR '09: Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, pages 1–10, New York, NY, USA, 2009.
- [2] Eitan Bachmat and Jiri Schindler. Analysis of methods for scheduling low priority disk drive tasks. *SIGMETRICS Perform. Eval. Rev.*, 30(1):55–65, 2002.
- [3] D. Bitton and J. Gray. Disk shadowing. In *Proceedings of the International Conference on Very Large Databases*, pages 331–338, Los Angeles, CA, August 1988.
- [4] John S. Bucy and Gregory R. Ganger. The disksim simulation environment version 3.0 reference manual. Technical report, CMU, 2003.
- [5] Jeffrey S. Chase and Ronald P. Doyle. Balance of power: Energy management for server clusters. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, 2001.
- [6] Shenze Chen and Don Towsley. A performance evaluation of raid architectures. *IEEE Trans. Comput.*, 45(10):1116–1130, 1996.
- [7] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–11, 2002.
- [8] Fred Douglass, P. Krishnan, and Brian N. Bershad. Adaptive disk spin-down policies for mobile computers. In *MLICS '95: Proceedings of the 2nd Symposium on Mobile and Location-Independent Computing*, pages 121–137, Berkeley, CA, USA, 1995. USENIX Association.
- [9] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Parthasarathy Ranganathan. Full-system power analysis and modeling for server environments. In *Workshop on Modeling Benchmarking and Simulation (MOBS)*, 2006.
- [10] Enrique Carrera Eduardo, Eduardo Pinheiro, and Ricardo Bianchini. Conserving disk energy in network servers. In *Proceedings of the 17th International Conference on Supercomputing*, pages 86–97, 2003.
- [11] Richard Golding, Peter Bosch, Carl Staelin, Tim Sullivan, and John Wilkes. Idleness is not sloth. In *TCOON'95: Proceedings of the USENIX 1995 Technical Conference Proceedings on USENIX 1995 Technical Conference Proceedings*, pages 17–17, Berkeley, CA, USA, 1995.
- [12] Anand Kuratti, , and William H. S. Performance analysis of the raid 5 disk array, 1995.
- [13] Chris Member-Gniady, Ali R. Member-Butt, Y. Charlie Member-Hu, and Yung-Hsiang Member-Lu. Program counter-based prediction techniques for dynamic power management. *IEEE Trans. Comput.*, 55(6):641–658, 2006.
- [14] J. Menon and J. Kasson. Methods for improved update performance of disk arrays. In *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on*, volume i, pages 74–83 vol.1, Jan 1992.
- [15] Arif Merchant and Philip S. Yu. An analytical model of reconstruction time in mirrored disks. *Perform. Eval.*, 20(1-3):115–129, 1994.
- [16] Arif Merchant and Philip S. Yu. Analytic modeling of clustered raid with mapping based on nearly random permutation. *IEEE Trans. Comput.*, 45(3):367–373, 1996.
- [17] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: practical power management for enterprise storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST 2008)*, 2008.
- [18] Athanasios E. Papathanasiou and Michael L. Scott. Energy efficient prefetching and caching. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 22–22, Berkeley, CA, USA, 2004.
- [19] E. Pinheiro and R. Bianchini. Energy Conservation Techniques for Disk Array-Based Servers. In *Proceedings of the 18th International Conference on Supercomputing (ICS 2004)*, pages 68–78, 2004.
- [20] Eduardo Pinheiro and Ricardo Bianchini. Energy conservation techniques for disk array-based servers. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, pages 68–78, New York, NY, USA, 2004. ACM.
- [21] Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27:17–28, 1994.
- [22] S. Gurumurthi and A. Sivasubramaniam and M. Kandemir and H. Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In *Proceedings of the 30th annual international symposium on Computer architecture*, pages 169–181, 2003.
- [23] Elizabeth Shriver, Arif Merchant, and John Wilkes. An analytic behavior model for disk drives with readahead caches and request reordering. In *SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 182–191, New York, NY, USA, 1998. ACM.
- [24] S. W. Son, G. Chen, and M. Kandemir. Disk layout optimization for reducing energy consumption. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 274–283, New York, NY, USA, 2005. ACM.
- [25] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, and Kaladhar Voruganti. Pergamum: replacing tape with energy efficient, reliable, disk-based archival storage. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 1–16, Berkeley, CA, USA, 2008.
- [26] Alexander Thomasian and Jai Menon. Raid-5 performance with distributed sparing. *IEEE Trans. Parallel Distrib. Syst.*, 8(6):640–657, 1997.
- [27] Mustafa Uysal, Guillermo A. Alvarez, and Arif Merchant. A modular, analytical throughput model for modern disk arrays. In *MASCOTS '01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 183, Washington, DC, USA, 2001.
- [28] Elizabeth Varki, Arif Merchant, Jianzhang Xu, and Xiaozhou Qiu. Issues and challenges in the performance analysis of real disk arrays. *IEEE Trans. Parallel Distrib. Syst.*, 15(6):559–574, 2004.
- [29] C. Weddle, M. Oldham, J. Qian, A. A. Wang, P. Reiher, and G. Kuenning. PARAD: A gear-shifting power-aware RAID. In *Proceedings of the Fifth USENIX Conference on File and Storage Technologies (FAST '07)*, pages 245–260, San Jose, CA, February 2007. USENIX Association.
- [30] Philip S. Yu and Arif Merchant. Analytic modeling and comparisons of striping strategies for replicated disk arrays. *IEEE Trans. Comput.*, 44(3):419–433, 1995.
- [31] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modelling hard-disk power consumption. In *the 2nd USENIX Conference on File and Storage Technologies (FAST 2003)*, pages 217–230, 2003.
- [32] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: Helping disk arrays sleep through the winter. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, pages 177–190, Brighton, UK, October 2005. ACM Press.
- [33] Qingbo Zhu and Yuanyan Zhou. Power-aware storage cache management. *IEEE Trans. Comput.*, 54(5):587–602, 2005.