Examining Energy Use in Heterogeneous Archival Storage Systems

Ian F. Adams*, Ethan L. Miller*, Mark W. Storer†
*University of California, Santa Cruz †NetApp

***Abstract*—Controlling energy usage in data centers, and storage in particular, continues to rise in importance. Many systems and models have examined energy efficiency through intelligent spin-down of disks and novel data layouts, yet little work has been done to examine how power usage over the course of months to years is impacted by the characteristics of the storage devices chosen for use. Long-term power usage is particularly important for archival storage systems, since it is a large contributor to overall system cost.**

In this work, we begin exploring the impact that broad policies (e.g. utilize high-bandwidth devices first) have upon the power efficiency of a disk based archival storage system of heterogeneous devices over the course of a year. Using a discrete event simulator, we found that even simple heuristic policies for allocating space can have significant impact on the power usage of a system. We show that our system growth policies can cause power usage to vary from 10% higher to 18% lower than a naive random data allocation scheme. We also found that under low read rates power is dominated by that used in standby modes. Most interestingly, we found cases where concentrating data on fewer devices yielded *increased* power usage.

I. Introduction

Data center energy usage doubled between 2000 and 2006 to nearly 61 billion kilowatt-hours, and has been forecast to double again by 2011 [1]. Storage is one of the primary power users in data centers, and is estimated to consume upwards of 35% the total energy used [2]. As systems grow to encompass thousands of storage devices and petabytes of data, the costs become staggering. Tens of thousands of dollars are being spent to power and cool storage devices, and energy costs are no longer the only issues—data center architects must now consider the associated "carbon footprint" of their systems as well.

Although there has been much work recently in the field of power efficient storage—such as intelligent disk spin-down techniques [3], and energy-aware data placement [4–6]—there has been little in the way of examining how broad, long-term management policies impact energy usage in large storage systems. This is a significant hole in our knowledge that has a large impact on storage as a whole, and in particular storage archives.

Archives are unique in the low density of accesses they receive, often spending significant portions of their time idle, where it may be acceptable to spin-down disk based storage for significantly reduced power consumption. Furthermore, archives are often monotonically increasing in the amount of data stored, and must store data for decades or longer, far exceeding the typical 3-5 year hardware life-cycle. This leads to large systems grown incrementally over many years, composed of many heterogeneous devices of varying capability and characteristics.

In this work, we explore the impact that a variety of broad policies (which devices are prioritized for use first) have upon the energy usage of a large archive of heterogeneous storage devices over the course of a year. Using a basic management framework within a discrete event simulator we found several notable results summarized here. First, with a basic energy-aware policy it is possible to reduce power consumption by nearly 20% over a naive policy when integrating devices in a growing heterogeneous system. Second, at low read rates, idle periods and the associated standby modes dominate power usage, reducing the impact of policy guided growth. Third, we found cases where a policy utilizing significantly *more* devices yielded comparable or lower power usage than other policies that concentrated data on fewer devices.

The rest of the paper is structured as follows: In Section 2, we discuss relevant work in the fields of storage system management, energy aware systems, and power modeling. Section 3 provides an overview of the architecture of our storage system. In Section 4 we provide details on our discrete event simulator. Section 5 covers experiments and results. Section 6 discusses future work, and Section 7 concludes.

II. Related Work

There are many storage system solutions and tools that aid in providing optimization and tuning towards organizational goals. However, archival storage solutions must focus on long-term growth and power-efficiency,

rather than short-term load-balancing and performance tuning. In particular, approaches that require frequent migration and balancing are inappropriate due to the energy cost and overhead to relocate the data.

Hippodrome [7] aims to maximize performance and minimize resource usage by providing automatic configuration of storage systems using details of available resources and workload. By analyzing previous I/O patterns and disk layout, it designs a system that is neither over nor under-provisioned for the predicted tasks, and reconfigures the current system to meet the new design. Over time, Hippodrome continues to analyze and reconfigure the system over time, eventually converging at an optimal design. Though the authors do not examine it, Hippodrome should be able to integrate power-awareness.

Strunk *et al.* [8] take the approach of automating the design and initial provisioning of a storage system based on its utility, *i.e.* the perceived profit or loss for a given system design. Similarly, Minerva [9] takes as input an application's needs and available storage device capabilities, and returns an optimal provisioning and layout for a storage system. While these systems are useful in initial design and provisioning of a storage solution, they design a static system, as opposed to one that will be undergoing continuous growth and evolution as is likely in a large scale archival environment.

Systems such as MAID [10], Pergamum [11] and PARAID [5] provide power efficiency by keeping as many disks as possible in lower power modes (spun down, or rotating at a lower speed) to minimize electricity usage. MAID and PARAID still aim to have relatively high performance by utilizing various data-layout techniques, while Pergamum forgoes performance for lower power usage with its use of independent low-power NAS devices. FAWN [12] is a clustered key-value storage system built from many low power devices; this design has similarities to Pergamum with its use of independent coordinated devices with a focus on power efficiency.

Write off-loading [4] is a technique whereby writes targeting a spun-down disk are redirected to persistent storage elsewhere, and opportunistically migrated back to the intended disk later, thus leaving more drives spun down. Popular Data Concentration (PDC) [6] migrates frequently accessed data to a subset of devices on a disk array, skewing the load such that more of the drives in the array can be transitioned to lower power modes. Similarly, Hibernator [13] utilizes *Adaptive Layout* to strike a balance between power-efficiency and performance in a storage system by dynamically migrating data blocks between devices. Otoo *et al.* [3] take an approach similar to the above systems, concentrating popular data on fewer devices. Like Hibernator, they maintain awareness of the performance impact of such migrations. Though these solutions consider the power and performance impact, none account for the cost of migration, and all are relatively reactive and short-term based upon characteristics of the workload they are under. This is important for archival storage, as it is better to have a merely adequate location for a long period of time, than more frequent migration to the optimal one, due to the cost of migration.

Allalouf *et al.* provide a framework for detailed power estimation in storage [14]. Their system can take a workload and details of device capabilities and provide accurate power use estimations. It is not clear how large a system may be simulated and their results, though accurate, are over quite short time periods—less than 8 hours—and only encompass at most a single RAID enclosure.

III. Architecture

In this section we provide an overview of the storage system architecture we use, and provide details on how it grows and maintains itself over long periods of time using high level policies to dictate which devices should be used first. Our architecture was explicitly designed for coarse-grained, long-term provisioning and evolution. This is in contrast to the management and provisioning architectures we discussed earlier that focus on shorter-term provisioning, load-balancing and performance optimization. Our system is not a replacement for management systems such as Hippodrome, but is rather a substrate for long-term, evolving storage systems where such fine grained control may be unnecessary or infeasible due to things such as lacking a defined workload or administrative overhead.

A. System Overview

At a high level, our system is comprised of 4 entities: storage devices (nodes), clients, an administrator, and a coordinator. Nodes provide usable space to the system and can coordinate with each other to provide redundancy, like that of the Pergamum system [11]. A client is anything that writes or reads data to the storage system. The administrator dictates policies that the coordinator uses to grow and optimize the system towards a desired goal, such as power-efficiency. Finally, the coordinator is a process that monitors and tracks the storage system, growing and maintaining the system based on the administrator's policies. The coordinator does not dictate where client reads and writes go within

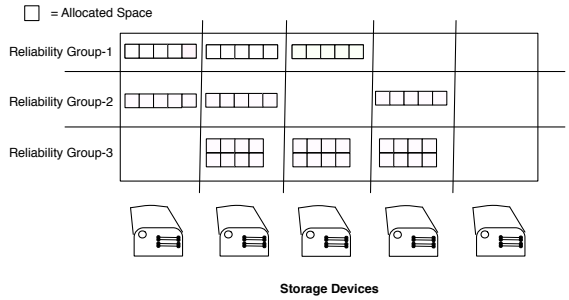


Fig. 1: An example system layout with three reliability groups composed of three nodes. The squares represent allocated space within each reliability group. Note that some devices are members of multiple RGs, while others have yet to be integrated into the system.

the system, rather it is only in charge of integrating devices into the active storage system, and choosing when and where to migrate data between devices.

Groups of devices are arranged into *reliability groups* (RGs); RGs are effectively distributed RAID groups [15]. RGs provide data survivability in the form of replication or error correcting codes so that device failures can be tolerated, which is of vital importance in a long-term archive. A device may be a part of multiple reliability groups. A single device's contribution to a reliability group is referred to as a *stripe element*. The *thickness* of a stripe is the amount of space each device contributes to a reliability group. The overall system's RGs and their respective details—stripe thickness, what devices are used—are referred to as the system *layout*. Figure 1 shows an example system layout.

The system has a single coordinator that is responsible for the creation and maintenance of RGs. In order to make intelligent decisions regarding the system layout, the coordinator maintains current information about the state of storage devices within the system, using distributed information gathering techniques [16, 17]. This information includes data such as: active and standby power usage, bandwidth, available space, space allocated to RGs, age, and RG membership. The coordinator uses this information to manipulate a system's layout through *management tasks*.

B. Management Tasks

There are two basic types of management tasks. The first, *scale-out*, chooses which devices will be used to create new RGs to increase the space available to clients. The second, *migration*, is responsible for migrating stripe elements between devices to improve efficiency or prepare a device for decommissioning.

Each management task runs on the coordinator and follows the same basic flow: trigger, candidate discovery, proposal generation, and implementation. First, a trigger dictates when a task should be started. Second, candidate discovery creates a list of devices appropriate for use in the current task. Third, proposal generation uses the list of devices and a *policy algorithm* to create a proposed change to the system, such as creating a new RG. Fourth, Implementation contacts the relevant nodes and issues commands to implement the proposal. We now describe each step in more detail, and how administrators dictate policies for when and how to change the system.

A trigger is an administratively defined policy threshold stating when a management task should take place, such as "create a new RG whenever space is 90% utilized". Once the trigger has been tripped, candidate discovery begins. In this step, the coordinator discovers a list of devices that should be considered for use within the current management task. The coordinator then runs this list through a *candidate device filter* that removes devices that do not meet administratively defined criteria, such as devices older than two years. The devices that remain after being run through the filter are known as a *candidate list*.

In proposal generation, the coordinator uses the candidate list to create a *proposal*. A proposal is a mapping of stripe elements to devices, either for determining where to migrate a stripe element, or which device should be used for a particular element of a new RG. Each task's proposal is created by a policy algorithm, which is the mechanism that system administrators use to dictate the metrics for which a proposal should be optimized towards, *i.e.* what devices should be prioritized for use first. So long as a valid proposal is returned from the algorithm, any technique and metric for optimization may be used for a policy. Figure 2 shows an example of two iterations of a policy algorithm for scale-out that is searching for a new power-efficient RG proposal.

After a valid proposal has been generated, the coordinator proceeds to implement it by contacting the relevant devices and issuing commands to act on the proposal. The combination of triggers, filters, and task layout algorithms are collectively referred to as a *system policy*. The system policy is how administrators tell the coordinator when and how the system should be grown, maintained, and optimized.

We now describe the specifics of each task's operations and what a valid proposal looks like for each in turn.

1) Scale-Out: A scale-out task is one that increases the usable space to the system through the creation of a new RG. A scale-out proposal has the form of a bipartite graph, with a one-to-one mapping of stripe elements to devices. A device may be mapped to one, and only one,

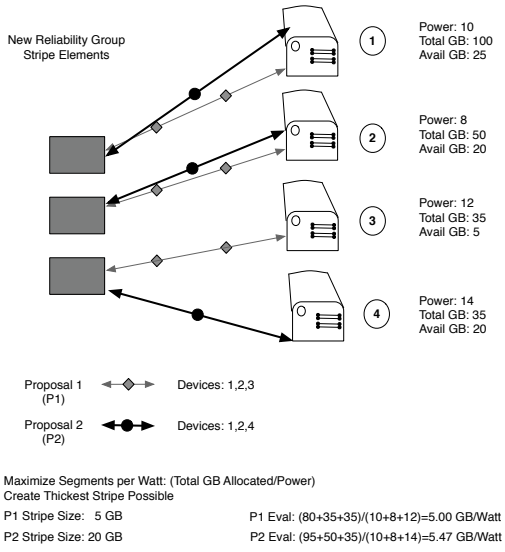


Fig. 2: An example of an RG policy algorithm searching for a power efficient RG of 3 devices. It randomly selects 3 devices and tries to create the thickest reliability stripe, evaluating the total contributed space/watt of each RG it finds. After a set number of iterations—here we see two—it returns the best mapping, in terms of space/watt, it finds.

stripe element from any single RG because it would violate reliability requirements to have multiple stripe elements from a single RG on one device.

2) Migration: Migration consists of moving stripe elements between devices within the system to improve the system state or prepare nodes for removal from the system. The process for migration includes an additional *culling* step to create a *move-list* before proceeding to candidate discovery. Culling consists of identifying which devices need their stripe elements migrated, which are then put in the move-list. Culling is much like candidate discovery: it can be defined to meet certain specifications, such as only selecting devices older than four years for stripe element migration.

A layout proposal for migration has slightly different rules from that of scale-out. In migration, multiple stripe elements *may* map to a single device, provided that the device in question does not already contain elements of the same RG, and that it has enough space to accommodate the stripe element being moved. For example, if there are two stripe elements from different RGs, both elements may go to a single device provided it has the available space, and the device is not already contributing to the RGs that the elements in question are from. However, if two stripe elements are from the *same* RG, they are not allowed to both map to the same device.

IV. Simulation

In this section we describe our discrete event simulator in more detail and the workload generators we use in our experiments.

To begin with, each simulated device is treated as a single element, and has a few basic characteristics: standby power draw, active power draw, bandwidth, and available and total space. Each device has a queue where read and write actions are appended. At each tick, if a device has any pending actions, it processes them until the bandwidth for that tick is consumed or the queue is empty. If all the bandwidth is used, any pending actions continue during the next tick. The amount of data that may be read or written to a device during a tick is the device bandwidth in megabytes per second (MBPS) adjusted for the length of the tick, which in our simulations is one minute.

A device consumes power at its active rate if has serviced a read or write within 5 time ticks. At each time tick, a device is either on at full power or in standby mode for the entire tick. The length of time a device waits before a transition to standby is its *idle threshold*. Though there are a variety of methods for calculating the optimal idle time [3, 18], we chose a fixed time for simplicity of implementation. A conservative spin-up delay of 15 seconds is simulated by consuming 1/4 of the available bandwidth for that tick.

Data is not cached within the system; all reads and writes go directly to the storage devices. We do this for two reasons: First, accesses to archives are generally assumed to be random and with little locality [19], reducing the effectiveness of caching. Secondly, with no caching we are provided a worst case scenario for power usage, as a device must always be powered up for a read or write if it is not already active.

Layout changes are reflected immediately within the system when they occur, and the coordinator has an omniscient view of the system. This was done to remain focused on the impact of various polices on power efficiency, not on simulating fine-grained implementation details. We leave examining the energy costs of migration to future work as it is a one-time cost, while in this work we focus on the impact of data location over time.

Parity updates and writes are handled in a coarse fashion. When a write is given to a data device in a RG, its corresponding parity devices receive a write of the same size as the original file. In a real system, the parity calculations may allow many writes to be coalesced into one. The end result is that the simulator may overestimate the amount of time parity writes and updates take, but will not underestimate, helping keep

our results conservative.

A. Files and Workloads

Our record-keeping for files is simple. Each file has a size, location—what device they are on—and a tag marking them as available for reading or not. When a file is inserted into the system, it is unavailable for reading until its corresponding write action completes. We do not stripe files across multiple devices as this requires spinning up every device in an RG for all reads and writes, which is contrary to the goal of a power efficient system. When a file read or write is put into a device queue, the device services them in a FIFO order.

Archival workloads are generally considered to be either write dominated [11] or read dominated [19], but with unpredictable read access patterns in either case. However, the most recent studies are from more than a decade ago [20]. Because of this, we chose to create a single generic write generator and three different read generators to examine power usage across several different access patterns.

The file-write generator periodically creates files and inserts them as writes to the system. The number of files created is contingent upon the size of the files and how much data is to be generated. The time at which a given file is written or read follows a rough diurnal pattern. Each file has a 90% chance of being accessed during the middle twelve hours of the day, from 6 AM to 6 PM.

We created 3 read generators for the simulation: random, 8020, and time-correlated. The random-read generator chooses files uniformly for reading, *i.e.* all files are equally likely to be chosen; this represents a worst case scenario for the system as any device is equally likely to be turned on when a read is issued. In the 8020 generator, 80% of the reads go to 20% of the files. When a file is written, it has a 20% chance of being marked as a member of the 20% group; this represents having a subset of the files be significantly more popular. The time-correlated read generator works as follows: when a read is issued, a file is randomly (uniformly) chosen. Of the prior and next 45 files—our simulator tracks files in a list ordered by their write times—between 5 and 35 are chosen. We then insert read requests for all such files within a 20 minute time-span to represent a client accessing a group of temporally correlated files . Like the writes, reads may be inserted in either a uniform or diurnal pattern, and the number of reads depends upon how much data is to be read.

V. Results

Our experiments fall into two broad categories to examine the power efficiency impact maintained under the policies we describe below. The first set of experiments show the impact of the policy-guided scale-out on the power-efficiency of the system, while the second examines migration tasks.

A. Experiment Parameters

In our system, we simulate a purely disk based approach. We chose disk based systems rather than tape for a number of reasons. First, tape suffers from very poor random access times. Second, tape scales very poorly, with long access times and very high overhead when migrating between systems. Third, though individual tapes take no power, tape drives and the associated hardware and robotics take significant power to run [11]. The choice of our device parameters was based upon the characteristics of commercially available hard drives and single disk network-attached storage drives available at the present time, summarized in Table I.

We create RGs of 14 data devices and 2 parity devices because our primary experiments are simulating a long-term archival style system that should withstand more than a single device failure simultaneously [11]. We do not simulate device failure or stripe element rebuild as we are not modeling a system's overall mean time to data loss. Additionally recovery is similar to migration in its long-term impact on the system, as it is effectively relocating a stripe element.

Reliability groups are created as needed. The trigger is for this is simple, when an RG becomes full, a new one is created via a scale-out task. All subsequent writes are assigned to the newly created RG. We iteratively write to each device until its stripe element is filled. Once all stripe elements are filled, the RG is not written to or updated again. In other words, we have an append-only storage system.

In our workloads, files uniformly range in size from 5 to 250 megabytes. We chose this range as archival systems, such as the Internet Archive [21], often group files together into large compressed files. Large file dominated systems can also be found in other arenas, such as video service and scientific computing. Table II has an overview of the characteristics we use in each of our read and write generators.

1) policy algorithms: We implement and use five policy algorithms for generating scale-out and migration proposals. The policies we chose are simple and along one or two dimensions to ease implementation, and aid in analysis of their impacts. We wanted a clear

delineation between the heuristics to better understand the influence various device prioritizations had upon the the system. We describe each briefly below, along with the motivation behind their use.

Random: This algorithm randomly picks nodes until a valid layout is found. For scale-out, it creates the thickest RG stripe possible out of the chosen devices. In other words, the stripe thickness is equal to the amount of space available on the device with the least left, thus fully utilizing that device. For migration, each stripe element is mapped to the first device that has sufficient space for the stripe element. We use this algorithm as our experimental control as it represents the most naive approach to growing and managing the system with no bias towards any device characteristic.

Greedy by Capacity per Watt (SW): This algorithm sorts the candidate list by each device's best potential space-per-watt. For example, a device with a 10 watt active power draw, and 500 GB of space would have a rating of 50 GB/watt. For scale-out, the algorithm chooses the first valid device for each stripe element and creates the thickest stripe possible. For migration the algorithm checks first to see if the stripe element under consideration will fit in the device, if so, it maps it there, else it checks the next best device. This policy is intended to create the most power efficient RG possible in scale-out, and similarly migrate stripe elements to power efficient nodes.

Greedy by Bandwidth (BW): This algorithm sorts the candidate list by device bandwidth. This algorithm is intended as a comparison for SW, and to show optimization over other metrics, in this case bandwidth. This algorithms migration and scale out tasks proceed the same as the SW algorithm.

Greedy by Power (Power): This algorithm sorts by candidate devices active power draw, with the *highest* power devices coming first. Its mappings of migration and scale out proceed the same as the above greedy algorithms

Greedy by SW$\times$BW (Combo): This algorithm sorts by the unit space per watt multiplied by the device bandwidth. This policy is intended to create RGs that prioritize power efficiency as well higher bandwidth in its device choices.

Note that we create the thickest RG stripe possible across all scale out policies . Though stripe thickness is an important facet of storage systems, and can impact issues such as rebuild and migration times, an examination of their effects is beyond the scope of this paper.

Our filter functions for scale-out remove devices that are already fully utilized from the candidate list, as they have no storage space to offer.

Space	500-1500 GB
Bandwidth	25-125 MBPS
Active Power Draw	5-18 watts
Standby Power Draw	1-4 watts
Idle Time	5 Minutes

TABLE I: Device characteristics.

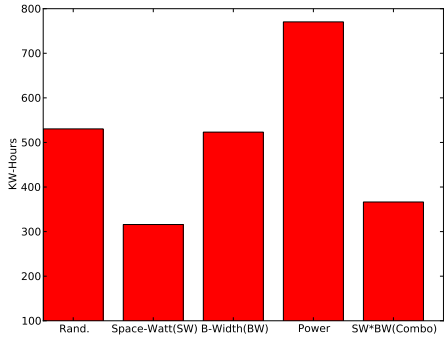
Amount Written	5000 GB Per Week
Amount Read	500 GB Per Week
File Size Range	5-250 MB
Read-Write Insert Pattern	Diurnal

TABLE II: Workload characteristics.

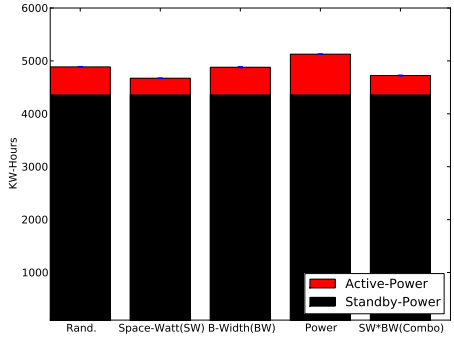
B. Scale-Out Experiments

In the first set of experiments we use 1000 heterogeneous devices, each generated randomly with characteristics detailed in Table I. We subjected the system to our workloads, tracking the overall power usage and number of devices powered on over the course of a simulated year. Each workload was run 5 to 10 times under a different device set for each scale-out (RG creation) policy. These experiments were designed to examine the power draw of a system grown under each of the various policy algorithms.

Under low rates of uniform reads, each policy algorithm has a significant impact on the amount of power used by active devices over the course of a year. The space-watt (SW) greedy algorithm performed the best, having 40% better power-efficiency than the control (random) policy, and 10% better than the next nearest policy algorithm (combo) as show in Figure 3(a). However, when we look at the system over time, and include the power used in standby modes, the impact of the various policies is minimal. We see no more than 2% difference in power consumption across all the workloads, as shown in Figure 4, because most devices spend the majority of their time in standby due to the low rate of reads. To illustrate this, in Figure 3(b) we show a comparison of the amount of power used between standby and active power usages at the end of one year. Note that the top subfigure shows only power used while *active* and the bottom shows the total power usage including both active and standby modes. In the figure we use a device standby power usage of 1/2 watt, as opposed to the 1-4 we originally simulated. This was done to show that even with very low standby power, the power use is dominated by devices in standby because of the low rate of device activations. This highlights a situation in storage where Amdahl's law [22] is analogously applicable, as we are optimizing over the devices power usage while active, which is a small fraction of the total time, reducing the impact of our optimizations.

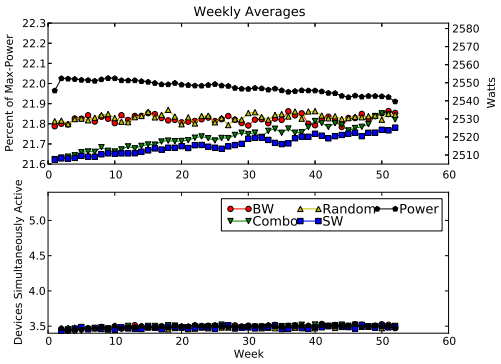


(a) One Year Power Draw, Active Devices Only

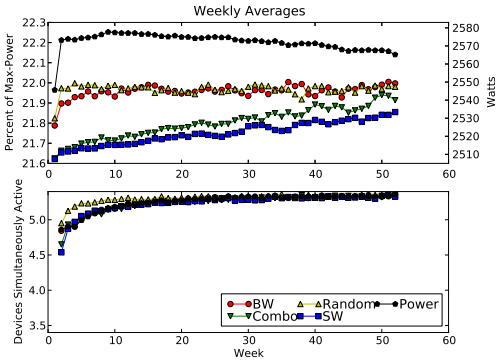


(b) One Year Power Draw, Including Standby Power

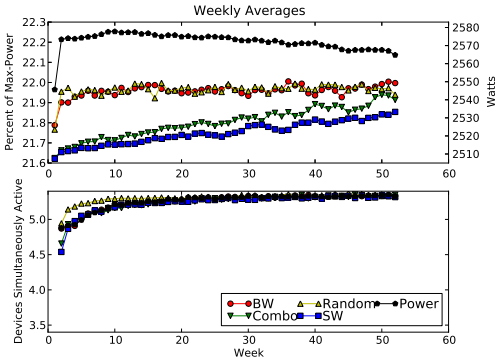
Fig. 3: These figures show the year's total power usage under each of the scale-out policy algorithms. Figure 3(a) shows only the amount of power when devices are active, *i.e.* the light portion of the bars in the bottom chart. Figure 3(b) shows the total power with the active power usage plus .5 watts per device. Results are averaged across 10 runs under an 80/20 workload.



(a) Time Correlated Diurnal Workload



(b) Random Diurnal Workload



(c) 8020 Diurnal Workload

Fig. 4: These figures show the weekly averages of weekly average power draw as a function of the total possible power usage, and the average number of devices simultaneously on. The data points are average across 5 device sets. Note little variation in power usage across workloads.

As mentioned above, we found there was relatively little overall difference between the workloads, shown by the overall weekly averages in Figure 4. This was because each read workload—8020, time-correlated, and random—was ultimately distributed across the system in a relatively uniform fashion, minimizing the impact of workload variation on the test. The impact of the 8020 and random workloads are nearly identical; although there is some locality in the 8020 workload, the popular files are also uniformly distributed, effectively mimicking the random workload. The time correlated workload had slightly lower power usage due to its higher locality and hence lower simultaneous device accesses.

Note that all the workloads' power draws appear to be converging in Figure 4. This is because, as the system grows in size, more devices must be used and the rate of device activations stays the same. This means reads are spread out across more devices, reducing the benefit that the early use of more power-efficient devices conferred.

As a contrast, we ran a test with the same rate of writes, but with the number of reads each week being a function of the size of the archive—15% each week—

under a random diurnal pattern. In other words, the rate of reads grew linearly with the amount of data stored. As shown in Figure 5, increased reads led to very different behavior over the course of the year. The average number of devices on at any time also increased linearly as the growing amount of reads were uniformly distributed throughout the system. This in turn led to a linear increase in the average power usage of the system across all task layout algorithms. At the end of a year, the SW policy based system was nearly 20% more power efficient than the control.

A counterintuitive result we found was despite the random scale out algorithm integrating over twice as many devices on average for each experiment—approximately 700 compared to 300 for all other algorithms as shown in Figure 6—its power draw was still *less* than that of the power-based scale-out policy and comparable to the bandwidth based one, as shown in Figure 3(a). This was due to two factors. First, the average device's active wattage for the random scale-out policy was 25 to 30% lower than that of the power-based policy. Second, because of the larger number of devices, reads and writes were further spread out. Individual devices spent *less* time active: approximately 2000 minutes per year, versus 4000 to 5000 for systems grown under the other algorithms, shown in Figure 7(b). Though fewer devices were utilized with the other (non-random) policies, their higher power usage, combined with the very low rate of reads, counteracted the potential impact of multiple reads or writes being serviced within a single spin-up. This is at odds with the general rule of thumb that utilizing fewer devices for storage results in better power-efficiency. The implication on future storage system design is that it may be desirable to have a significantly larger number of lower power devices for storage, than fewer higher power and performance devices, particularly for low-density, low-locality workloads.

We found that the random policy also had significantly fewer spin-ups per device on average, shown in Figure 7(a). This is significant for two reasons. First, frequent spin-ups and spin-downs may physically impact the reliability of hardware when done at high frequency, but at relatively low rates such as that shown (effectively 2-3 times a day for most of our policies) its impact is minimal [23]. Second, spin-up is itself an energy intensive operation. This serves to highlight the compromise that must be done in regards to disk spin-down as a technique for energy savings. On the one hand, it allows significant power savings to be realized, on the other hand, it incurs access delays, extra-energy costs, and hardware wearing. For example. when we looked

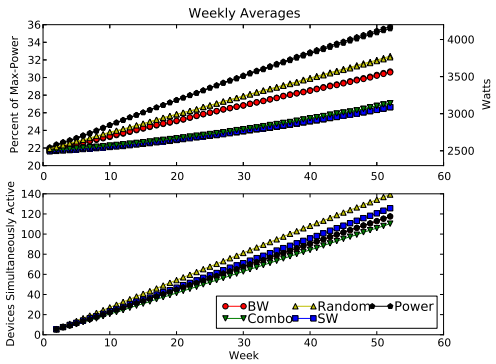


Fig. 5: The averages for system with random diurnal reads upscaled to read 15% of the archive each week. This is averaged across 10 runs of a system under a random diurnal workload. Note the linear growth and increased power usage over that shown with the constant rate of reads in Figure 4

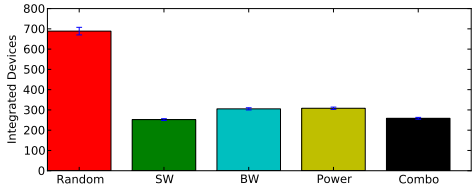


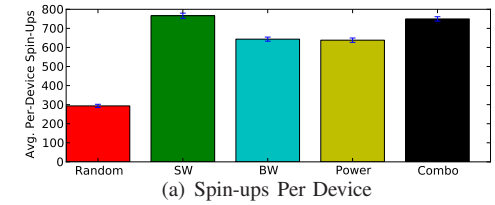
Fig. 6: This figure shows the average number of devices integrated into the system. The random policy integrates over twice as many devices as the other policies, as it does not prioritize any device type.

at the spin-up numbers for our upscaled workload (15% of the archive read per week) the number of spin-ups increased by nearly an order of magnitude, a situation where hardware wearing and extra energy costs could become a serious issue.

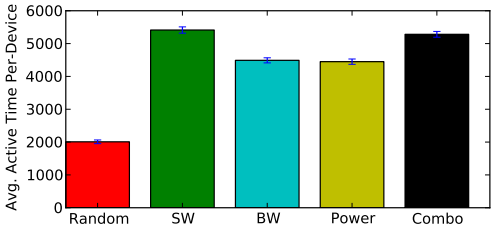
If we examine the average bandwidth of the devices integrated into the system (see Figure 7(c)) we can see that the combination metric chose devices with higher average bandwidth, and as shown in Figures 3 and 5 it still provides good power-efficiency as well. This demonstrates that even simple greedy policies like the ones we are using, can not only optimize a system towards a power efficiency, but incorporate performance as well. Despite the increased performance however, the increased bandwidth had minimal impact on the time active, as the reads were infrequent enough that servicing multiple reads quickly offered little benefit in terms of a device being able to spin-down sooner.

C. Migration Experiments

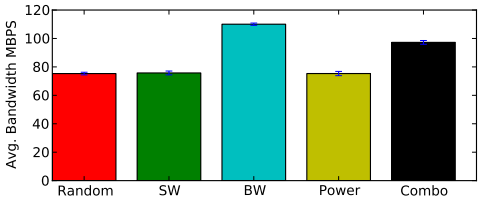
To examine the impact of migration, our experiments had two phases. First, 500 devices were populated with files until 125 TBs of data was inserted. RGs were



(a) Spin-ups Per Device



(b) Active Time Per Device



(c) Avg. Bandwidth

Fig. 7: These figures show averages for spin-ups, active time, bandwidth of devices over a year under an 8020 workload. Note the random policy has significantly less active time and spin-ups, and devices under the BW policy remain active for similar amounts of time despite higher transfer rates.

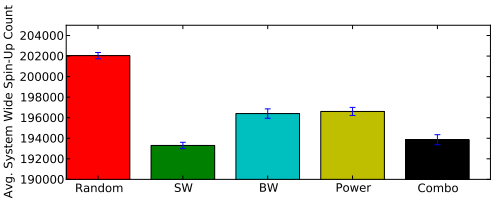
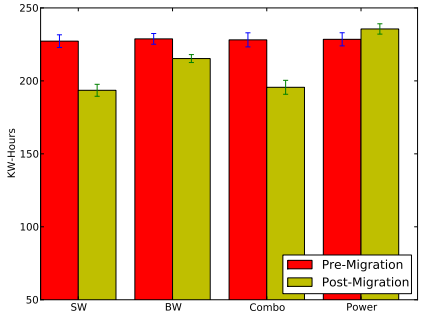


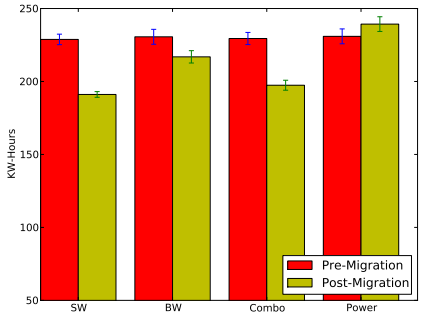
Fig. 8: This figure shows the average number of device spin-ups across the system. Despite different policies and in particular the random policy's fewer integrated devices (Figure 6) the number spin-ups across all policies are within 5% of one another.

created as needed using the random RG creation algorithm. We ran a read workload of each type—8020, time-correlated, and random—at a rate of 500 GB per week under a diurnal pattern, and tracked the power usage.

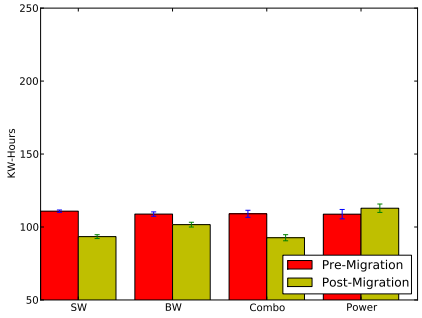
For the second phase, the coordinator identified the 20% least power-efficient devices—those with the worst total space-to-watt ratio—that had been integrated into one or more RGs. It then migrated the least efficient devices' stripe elements to other devices in the system using each of the heuristic algorithms described earlier.



(a) Random Diurnal



(b) 8020 Diurnal



(c) Time Correlated Diurnal

Fig. 9: Impact of migration policies. Note impact is consistent across all workloads, though the time-correlated uses less power due to its high-locality and lower device active time.

After migration we ran the same workload again and tracked power usage. Each migration layout algorithm was run on the same 500 device set, with five iterations of each workload and the cumulative results averaged.

Figure 9 shows that running the migration policies on the 500 random device set affected the overall power usage of the system in each case. The SW algorithm improved the most across the workloads, between 14 and 16%. The power based policy increased the amount of

power usage slightly. Though this is within the standard deviations, each of the algorithms concentrates stripe elements on fewer devices, while random spreads it out. This is much the same as the earlier counter-intuitive result where concentrating data manages to increase power usage. Despite concentrating data on fewer devices, power usage increased measurably, adding more weight to our earlier result showing that spreading data out may yield power savings when there is a choice between using few high power devices and greater numbers of low power ones.

VI. Future Work

There is a great deal of investigation left to be done. To begin with, we do not look at the energy impact of migrating and rebuilding data within a large archival system. In systems with upwards of tens of thousands of devices, such activities will be a frequent event. We also need to look at power-aware multidimensional policy algorithms and heuristics. The ones in this work, though effective for exploring power usage impacts, only look at one or two dimensions which is inappropriate for real world use.

Closer examination of the power-performance tradeoff on a large scale is also needed. We focus solely on power as it is generally held that archival systems users will tolerate low performance in order to reduce operating expenses, though this is not appropriate for all systems. Examination of the feasibility and impact of fine-grained control on the energy usage and performance of a system over very long time scales is needed as most work examining the trade-off focuses on only a few devices over the course of minutes to hours, not months to years.

VII. Conclusions

This paper presented an examination of how various policies influence the total power usage of a large distributed archival system over extended periods of time. We found several useful results. First, for low read-rates, power usage is almost completely dominated by that during long idle periods. Second, concentrating data on higher power devices may mitigate the impact of amortizing multiple reads and writes within a spin-up, under low read densities. Third, even naive heuristics can have a significant positive, or negative, impact upon power usage when growing or migrating data within a system. In total, our work provides an initial exploration of long-term power usage, and demonstrates the need for further studies in power-aware storage and archive-specific architectures to efficiently store rarely accessed data.

References

[1] N. Anderson, "Epa: Power usage in data centers could double by 2011," http://arstechnica.com/old/content/2007/08/epa-power-usage-in-data-centers-could-double-by-2011.ars, August 2007.

[2] D. Robb, "Storage turns power hungry," http://www.enterprisestorageforum.com/management/features/article.php/3639286/Storage-Turns-Power-Hungry, October 2006.

[3] E. J. Otoo *et al.* "Analysis of trade-off between power saving and response time in disk storage system," Lawnrence Berkeley National Laboratory, Tech. Rep. LBNL-2184E, 2009.

[4] D. Narayanan *et al.* "Write off-loading: Practical power management for enterprise storage," in *Proceedings of FAST 2008*, Feb. 2008, pp. 253–267.

[5] C. Weddle *et al.* "PARAID : A gear-shifting power-aware RAID," in *Proceedings of FAST 2007*, Feb. 2007.

[6] E. Pinheiro and R. Bianchini, "Energy conservation techniques for disk array-based servers," in *Proceedings of Supercomputing 2004*, Jun. 2004.

[7] E. Anderson *et al.* "Hippodrome: running circles around storage administration," in *Proceedings of FAST 2002*, Monterey, CA, Jan. 2002.

[8] J. D. Strunk *et al.* "Using utility to provision storage systems," in *Proceedings of FAST 2008*, 2008.

[9] *Minerva: An automated resource provisioning tool for large-scale storage systems*, vol. 19, 2001.

[10] D. Colarelli and D. Grunwald, "Massive arrays of idle disks for storage archives," in *Proceedings of Supercomputing 2002*, Nov. 2002.

[11] M. W. Storer *et al.* "Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage," in *Proceedings FAST 2008*, Feb. 2008.

[12] D. G. Andersen *et al.* "FAWN: a fast array of wimpy nodes," in *Proceedings of SOSP '09*, 2009.

[13] Q. Zhu *et al.* "Hibernator: Helping disk arrays sleep through the winter," in *Proceedings of the SOSP 2005*. Brighton, UK: ACM, Oct. 2005.

[14] M. Allalouf *et al.* "Storage modeling for power estimation," in *Proceedings of SYSTOR 2009*, 2009.

[15] M. Stonebraker and G. A. Schloss, "Distributed RAID—a new multiple copy algorithm," in *Proceedings of ICDE 1990*, Feb. 1990, pp. 430–437.

[16] P. Yalagandula and M. Dahlin, "A scalable distributed information management system," in *Proceedings of SIGCOMM 2004*. Portland, OR: ACM Press, Aug. 2004, pp. 379–390.

[17] P. Yalagundula and M. Dahlin, "Shruti: A self-tuning hierarchical aggregation system," in *Proceedings of SASO 2007*, Boston, MA, Jul. 2007, pp. 141–150.

[18] D. P. Helmbold, D. D. E. Long, and B. Sherrod, "A dynamic disk spin-down technique for mobile computing," in *Proceedings of MOBICOM 1996*. Rye, New York: ACM, Nov. 1996, pp. 130–142.

[19] M. Baker, K. Keeton, and S. Martin, "Why traditional systems don't help us save stuff forever," in *Proceedings of HotDep 2005*, Jun. 2005.

[20] E. Miller and R. Katz, "An analysis of file migration in a Unix supercomputing environment," in *Proceedings USENIX 1993* , Jan. 1993, pp. 421–433.

[21] E. Jaffe and S. Kirkpatrick, "Architecture of the Internet Archive," in *Proceedings of SYSTOR 2009*, May 2009.

[22] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS*, vol. 30, 1967.

[23] T. J. E. Schwarz *et al.* "Disk scrubbing in large archival storage systems," in *Proceedings of MASCOTS '04*, Oct. 2004, pp. 409–418.