Using Replication for Energy Conservation in RAID Systems

Jinoh Kim and Doron Rotem
Lawrence Berkeley National Laboratory
University of California
Berkeley, CA 94720, USA
{jinohkim,d_rotem}@lbl.gov

Abstract—Energy efficiency has become a major concern in data centers as several reports predict that the anticipated energy costs over a three year period will exceed hardware acquisition. In particular, several reports indicate that storage devices (and cooling them off) may contribute over 25 percent of the total energy consumed in a data center. In this paper, we present a novel approach for energy conservation, called iRGS (inter-RAID Gear-Shift), which utilizes data replication as a tool for extending the idleness period of a large fraction of the disks. iRGS adapts to the workload observed by the system, thus allowing energy saving subject to required service level. In particular, iRGS can manage power in large data centers with multiple RAID groups unlike previous work which deals with individual disks on one RAID group. To enable this, iRGS provides (1) a new replication algorithm that allows gradual adaption to the workload by gear shifting, (2) a mapping technique to service requests under power saving modes, and (3) a write consistency mechanism for new writes and existing file updates under replicated environments. Simulation with real life trace data (Cello99) and synthetic data shows that our method saves up to 60% of energy, and outperforms existing power management algorithms, while providing better response time.

Keywords: Power Management, RAIDs, Replication

1. Introduction

Power optimization in data center environments has recently gained a lot of interest because of the costs involved in power delivery and system cooling. In a recent report to congress [12], EPA stated that many data centers have already reached their power capacity limit and more than 10% of data centers will be out of power capacity by the end of this year, while 68% expect to be at their limit within the next three years.

Among the many components in the data center, it is currently estimated that disk storage systems consume about 25–35 percent of the total power [4]. This percentage of power consumption by disk storage systems will only continue to increase, as data intensive applications demand fast and reliable access to on-line data resources. This in turn requires the deployment of power hungry faster (high RPM) and larger capacity disks. Most of the larger data centers use some type of RAID disk configuration to provide I/O parallelism. While this parallelism benefits performance, it increases the number of spinning disks and energy consumption. There are many suggested techniques for power savings in the research literature, including:

(a) Dynamic power management (DPM) algorithms [5]: These make decisions in real time when disks should be transitioned to a lower power dissipation state while experiencing an idle period. The length of the idle period before a spin down is triggered is called idleness threshold. Analytical solutions to this online problem have been evaluated and it was shown that the optimal idleness threshold period should be set to \(\frac{\beta}{P}\tau\) where \(\beta\) is the energy penalty (in Joules) for having to serve a request while the disk is in standby mode, (i.e., spinning the disk down and then spinning it up in order to serve a request) and \(P\tau\) is the rate of energy consumption of the disk (in Watts) in the idle mode. However, it is difficult for this kind of fixed threshold-based techniques to adapt to workload characteristics, which may vary significantly over time.

(b) Using Solid State Devices (SSDs) instead of disks [6]: While SSD technology may help to alleviate some of the energy problems, they are currently an order of magnitude more expensive than HDDs in terms of dollars per gigabyte [9].

(c) MAID (massive array of idle disks) technology [2]: In a MAID architecture, the number of drives that can spin at any one time is limited. This allows extremely dense packaging, often impossible in conventional architectures. An example is a single COPAN\(^1\) frame that can support up to 896 drives.

\(^1\)http://www.sgi.com/products/storage/maid/
where at most 25% of the drives are spinning at any given period. Disks are spun down whenever they experience long idle periods. In case data is needed from a non-spinning disk, severe response time penalties may be incurred. This technology was successfully utilized for archival type data, but in most typical data center environments, the disks do not experience long enough idle periods, unless some request redirection is available.

(d) Disks with dynamic RPM [16], [10], [14]: This technology may potentially allow to move disks to a state that consumes less energy while still being able to actively serve requests although at a slower transfer rate. This technology is unfortunately not yet available on a large commercial scale.

Our approach in this work is to use data replication as a tool for extending the idleness period of a large fraction of the disks, thus allowing them to be spun down. Our algorithms adapt to the workload observed by the system (e.g., disk utilization, I/O request arrival rate) thus allowing energy saving subject to required service level. Our key contributions are as follows:

- We present the design of a new replication algorithm that allows gradual adaptation to the workload by gear shifting for massive RAID systems.
- We develop a mapping technique to service requests under power saving modes and a write consistency mechanism for new writes and existing file updates in replicated environments.
- We present experimental results with real life trace data (Cello99 [1]) and synthetic data. The results show that our method outperforms PARAID and DPM algorithms in energy saving, while providing a better response time.

The paper is organized as follows. First, we provide related studies, particularly utilizing replications for energy conservation in section 2. Our novel replication and mapping techniques for flexible gear shifting will be presented in section 3. We provide our evaluation results in section 4 with synthetic and Cello99 workloads. Finally, we provide conclusions and future work directions in section 5.

2. Related Work

There has been a great deal of work on energy conservation for large-scale storage systems, based on caching [15], [17], data placement [14], data migration [10], [16], and data replication [11], [8], [13], [7]. These techniques try to prolong disk idle times, so as to make it possible to place idle disks in low-power state, thus saving energy. Among these techniques, exploiting data replication is an attractive practical option, since it is widely employed in server clusters for diverse purposes, including data availability, durability, load balancing, etc. Although replication requires additional storage capacity, it usually comes at a very low cost, since it is well known that storage resources in data centers are often considerably under-utilized at around 1/3 of total available capacity [4], [7], [8]. In addition, one critical problem of non-redundancy-based techniques is the possibility of extremely long response times (greater than tens of seconds) for any data accesses against powered-down disks as these disks need to be spun up before they can serve any requests. Given that data access can be bursty, more than a single request may experience such unexpected, huge latencies, violating system level agreements (SLAs). In this work, we focus on techniques for taking advantage of data replication for energy conservation.

Diverted Accesses [11] and eRAID [8] utilize existing replication for energy saving by using redirection of requests under light load. For example, eRAID relies on RAID-1 mirroring. By exploiting replication, it is possible to spin down mirrored disks thus saving energy. Since the spinning disks contain all data in the system, there would be no data access misses, and this could extend disk sleeping time, unless system load surges up.

While the above studies simply exploit replicated data, some studies [13], [7] provide replication strategies for energy management. Lang et al. [7] suggested a replication technique based on Chained Declustering. The basic idea is to create a full replication to one of the adjacent nodes in a virtual ring topology. Thus, up to half of the nodes can be spun down for energy management. The authors also showed how they can achieve load balancing in such a ring-based replication setting.

PARAID [13] suggests replication in a RAID group. Instead of running all disks in a RAID group, PARAID determines an appropriate gear level, i.e., how many disks in a RAID group are spinning vs. disks in sleep mode. This decision is based on the current system load (derived from disk utilization). Based on gear level, disks in the RAID system are spun down (when the gear goes down) or spun up (when the gear goes up). PARAID provides skewed data replication, so that it can continue service using only a subset of the disks in each RAID group. Our work is inspired by PARAID in terms of shifting gear levels depending on current workload. One big difference is that our approach, called iRGS (Inter-RAID Gear-Shift), shifts gear levels be-
between RAID groups (or RDG) rather than within a single RAID group. Hence, there is no sacrifice in parallelism that PARAID may suffer in low-level gears. In addition, iRGS can be more suitable for large-scale settings, since data centers include multiple, possibly hundreds of, RAID groups. In this case, power management based on RAID groups (instead of individual disks) would be simpler and more efficient. In the next section, we discuss details of our iRGS architecture and operations.

3. Inter-RAID Gear-Shift

The basic idea of iRGS is to maintain replicated data between multiple RDGs. This redundancy allows the system to serve requests from a replica that resides on a spinning disk allowing a large fraction of the disks to remain in the sleep state resulting in significant energy conservation. Based on system workload, iRGS shifts the gear, which results in changes of power state of RDGs. Again, our approach shifts power modes not in a single RAID system, but between multiple RAID groups. Thus, any gear-shift makes entire disks in a RAID group either spun down or up. When the load gets light below a certain boundary, iRGS attempts one-level gear-down shift sending an RDG to low-power state. In such a low-gear mode, replicated data is accessed instead of the original data. Conversely, iRGS shifts the gear up whenever necessary, e.g., when observing performance degradation.

3.1 Replication

We first discuss the replication strategy in iRGS. Figure 1 illustrates iRGS replication and gear shifts. In the highest gear, all RDGs are spinning, whereas at the lowest gear only RDG1 is spinning. For each gear shift, one RDG is either activated (spun up) or deactivated (spun down). As can be seen in the figure, RDGs have different sets of replicas to facilitate gear-shifts for power management. We denote by “super-RDGs”, RDGs that are kept spinning even in the lowest gear level, while “ordinary-RDGs” can move to low-power state to conserve energy. In the example in Figure 1, RDG1 is the only super-RDG.

iRGS employs replication not only for power management but also for fault tolerance. Each data is maintained in 2 copies. Table 1 shows an example of replication in iRGS with 6 RDGs. iRGS replicates all the ordinary-RDGs data to super-RDGs, where each super-RDG gets an equal fraction of the replicated data. Here, \( X_{a/b} \) stands for the fraction of the data of \( X \). Since there are 2 super-RDGs in this iRGS setting, they each keep half of the ordinary-RDGs’ data. Ordinary-RDGs also maintain a part of other ordinary-RDGs replicas based on the gear-shift principle. By assigning a part of replicas to ordinary-RDGs, iRGS can distribute the request load in a low-gear mode. In order to maintain fault tolerance requirements, super-RDG blocks are also evenly replicated to ordinary-RDGs as shown in the table, even though super-RDGs are always kept spinning. It is easy to prove that with this replication strategy, any block of data is replicated at least 2 times and at most 3 times.

One question that arises is: Which blocks in an ordinary-RDG are replicated on other ordinary-RDGs? Let us consider an example with the above replication in Table 1. The entire set of blocks in D is copied to super-RDGs (RDG1 and RDG2). Additionally, 1/3 of the blocks in D is replicated to RDG3. A question is that which blocks of D will be assigned to RDG3? This is important in order to balance the load at gear level 2 (notice that both RDG1 and RDG2 are running at gear level 1). If the copied blocks of D are only a subset of the blocks on RDG1’s, RDG2 should endure 1/2 of load for D at gear 2. Currently, we randomly assign replicas to ordinary RDGs. Our future work will include explorations of other possible assignments.

Now, let us discuss storage requirements for replicas.
for each RDG. Suppose \( N \) RDGs and \( M \) Super-RDGs \((M < N)\). Let \( V_i \) be the volume of original data, and \( S_i \) be the size of replica storage for RDG-\( i \). For ease of exposition, we assume \( M \) RDGs with the lowest identifiers (i.e., RDG 1 to \( M \)) have the role of Super-RDGs. Thus, the rest of RDGs (i.e., RDG \( M+1 \) to \( N \)) will work as ordinary-RDGs. In addition, we assume that power management spins down an RDG with the greatest identifier among current working RDGs. Storage requirements for replication in this setting are as follows:

\[
S_i = \begin{cases} 
\sum_{k=M+1}^{N} V_k / M & \text{for Super-RDGs;} \\
\sum_{k=1}^{M} V_k / (N - M) & \text{for RDG-} N; \\
S_N + S_{i+1} + V_{i+1} / i & \text{otherwise (for RDG-} i\).
\end{cases}
\]

### 3.2 Write Consistency

Although our focus is more on read-dominant environments, iRGS also provides functionality for write consistency. There can be two kinds of write requests:

1) **New writes**: New files are created.
2) **Updates**: Existing files are updated.

The mechanism we use for write consistency is straightforward. The main principle is that we redirect all writes to super-RDGs first and perform a reorganization phase later. For new writes (case 1), one of the super-RDGs is selected to accommodate new files. For this, there may be some optimizations, such as first-fit, best-fit, and worst-fit, based on the new file size. For updates (case 2), iRGS simply updates the corresponding super-RDG copy, and marks the original copy as “stale”, so that it will not be accessed for any subsequent requests. Regardless of gear level, after that, only the super-RDG copies are accessed for any read and write requests, until a reorganization process is executed.

Whenever a gear goes up to the highest level, reorganization is scheduled. Shifting to the highest level means that the current system load is too heavy to handle. Thus, we do not want to impose additional overhead at that time. After a while, if conditions allow iRGS to down gear-shift, reorganization is activated, deferring gear-shifts during reorganization. In this phase, modified blocks in super-RDGs are flushed to RDGs storing original blocks and stale information is reset (The details of this are explained in section 3.3.). New files are also moved to one of ordinary RDGs (based on the predefined policies) and replication takes place as discussed in section 3.1. Since super-RDGs cache new files, they simply throw away a part of the files unnecessary to keep. The gear is then shifted down after reorganization completes. In the course of reorganization, any condition to gear-up interrupts this process, and the system devotes all its power for user requests.

As may be noticed, iRGS relies on no additional components such as extra caches for update consistency. Nonetheless, adding disks or non-volatile memory for caching would be helpful for dealing with write requests, as considered by other studies [13], [8]. Our current work is focused on taking advantage of existing infrastructures, but we plan to employ large non-volatile caches in the future, as the price of SSD rapidly decreases [6].

### 3.3 Request Mapping

iRGS shifts gear based on system load. In other words, RDGs (ordinary-RDGs) change their power state over time. In the highest gear, load is balanced by allowing access of “original data” rather than “replicated data”, unless original data is not stale (for read requests). When the gear shifts down, the rest of RDGs evenly share the load for low-powered RDGs. To enable this, iRGS maintains a mapping table, as illustrated in Figure 2. When a request arrives and the original data block is located in a sleeping RDG, iRGS first refers to the mapping table. For each data block, the associated mapping table entry contains replica addresses in both ordinary-RDGs and super-RDGs, in addition to a flag indicating update history (stale or new creation). The associated ordinary-RDG address can be null, but super-RDG address should not be null.

Algorithm 1 illustrates how the request is serviced in the system. If the request is for a new write, iRGS allocates a block from one of super-RDGs, and a new entry is created in the mapping table. In the case of read, the stale bit is checked first, based on which the request is either directed to the original address or redirected to mapped address. The request for update is handled almost similarly to read, but only super-RDG copies are updated and the stale bit is set to indicate not to access the other associated copies.

We consider the stripe size for the block size in the mapping table. If the system is configured with 146 GB disks, 128 KB for stripe size, and 24 RDGs (which is our experimental configuration with 120 disks in section 4), the required amount of storage is 230 MB for the mapping table.\(^3\) Since server clusters typically

\(^3\)We compute this with 8-byte mapping entry size: ORA (31 bits), SRA (31 bits), flag (2 bit), and RDG# (7 bits) and Block# (24 bits) in the address format.
Input: Request $r$

```java
switch $type$ do
  case new writes:
    Get one of super-RDGs and block address;
    Write the block to the address;
    Create a mapping entry with $flag = CREATE$;
  endsw
  case update:
    Entry $e = MappingTable.get(r.address);
    Write block to $e.SRA$;
    $e.flag = STALE$;
  endsw
  case read:
    Entry $e = MappingTable.get(r.address);
    if $e.ORA$ is not null and $e.flag != STALE$ then
      Read block from $e.ORA$;
    else
      Read block from $e.SRA$;
    end
  endsw
Algorithm 1: Request service
```

have very large memory (as large as tens of GBs), the mapping table can be accommodated in the main memory.

### 3.4 Discussion

Scalability is an essential component for large-scale systems. If there are any large number of RDGs (e.g., several tens of RDGs) in the system, some procedures, such as replication, can be much more complicated. To handle this, we consider “partitioning” of the system: each partition comprises of super-RDGs and ordinary-RDGs as discussed, and replication and redirection only applies within a partition.

Exploring gear management is one of the big challenges in this project. Gear-shift can be reactive, as PARAID does based on disk utilization [13], or proactive by incorporating load prediction models. In our initial evaluation presented in the next section, we use a reactive technique similar to PARAID’s based on disk utilization. One main difference is that iRGS uses super-RDG disk utilizations for gear shifts, while PARAID uses RAID disk utilization. More specifically, iRGS computes disk utilizations for each super-RDG at every time frame (32 seconds used in PARAID), and compares the maximum utilization ($U$) and its associated standard deviation ($S$) to threshold values for upshift ($T_u$) and downshift ($T_d$). If $U + S > T_u$, iRGS shifts the gear up; otherwise, if $U + S < T_d$, iRGS shifts the gear down. For our evaluation, we used $T_u = 0.9$ and $T_d = 0.5$, but we plan to design more sophisticated gear-shift techniques with prediction models to maximize energy benefits.

### 4. Evaluation

In this section, we present our initial experimental results with synthetic and real workloads.

#### 4.1 Experimental Setup

For evaluation, we augmented Disksim [3], widely used for studying storage systems. We consider Seagate Cheetah 15K.5 enterprise disks. For this disk model, however, some power information, such as standby power and spin up/down power, is missing in the associated documents. For this reason, we alternatively chose power parameters from Seagate Barracuda specification. Since the main purpose of our experiments here is to see applicability of iRGS in terms of both performance and power, comparing iRGS with existing techniques with the identical power parameters would be acceptable. The power model we used in this paper is shown in Figure 3.

In our experiments, we consider a large-scale data center consisting of 120 disks. Although our model has no dependency on RAID organization, we simply assume RAID-5 structure with 4 data disks and 1 parity disk. Thus, there are 24 individual RAID groups in

---

the system (i.e., 24 RDGs). We partitioned the system into four partitions each with 6 RDGs. Each partition consists of 2 super-RDGs and 4 ordinary ones.

We employed two workloads, a synthetic workload close to the Internet data access model and a real trace from Cello99 [1]. Cello99 is a trace collection gathered by the HP Storage Research Lab in 1999 throughout the year (except for a couple of weeks in January). We used 3-day traces between May 1st and May 3rd. Table 2 describes characteristics of the workloads in our experiments. As seen in the table, the synthetic trace is read-dominant, while the Cello99 trace is write-intensive.

We compared 4 different systems: NPS is a base system for comparison without any energy management functions; FTH is a system employing a fixed idleness threshold, $T_\tau = \frac{\beta}{P\tau}$, mentioned in section 1; PARAID($n, m$) is a PARAID configuration with $n$ disks with gear shifting down to $m$; iRGS($x, y$) is an iRGS configuration with $x$ RDGs and $y$ super-RDGs in a partition. We set up two PARAID systems (PARAID(5,3) and PARAID(5,2)) and one iRGS system (iRGS(6,2)). By definition, PARAID systems can spin down or up to 40% (PARAID(5,3)) and 60% (PARAID(5,2)) disks, while iRGS(6,2) can spin down 67% disks at max.

### 4.2 Synthetic Workload

We begin by presenting experimental results with the synthetic workload. Figure 4 presents normalized power saving compared to NPS. As shown in the figure, PARAID systems saved 40%–54% power compared to NPS. iRGS further saves energy up to 60%. Interestingly, FTH yielded no power saving. This is due to a fairly high rate of requests with 10 ms inter-arrival time that can create 4–5 requests for each disk within the idleness threshold time. Even with the skewness for disk access pattern ($\alpha = 1.0$), there is a high probability that each disk can see at least a single request within the idleness time period. For this reason, in this setting, FTH cannot see many chances to spin down disks.

We observed no significant performance differences in terms of response time distribution. The mean response times are NPS=3.9 ms, FTH=4.0 ms, PARAID(5,3)=4.9 ms, PARAID(5,2)=4.4 ms, and iRGS(6,2)=4.8 ms. 99% response times for all the techniques are located within 10 ms.

### 4.3 Cello99 Workload

We next report our simulation results with the 3-day Cello99 trace. Figure 5 shows power saving results with this real workload. Unlike the synthetic workload, FTH could save energy with this workload, suggesting power management with a fixed threshold can be highly sensitive to workload characteristics. In contrast, PARAID and iRGS consistently conserve energy in both workloads. The saved powers by those systems in this workload are almost equal to the ones in the synthetic workload (37%–60%). Unlike the synthetic workload, FTH dramatically saved energy over 70% than NPS. This is because FTH could see many more chances to spin down disks with a lower degree of request arrival rate than the synthetic case. However, it needs to pay severe performance penalties with respect to response time for power saving. Table 3 summarizes response times for each technique on average and in percentile. The average response time of FTH is over 3 times of NPS's, and the percentile numbers suggest a heavy tail in distribution. PARAID systems show better performance than FTH, but still pay considerable penalties: PARAID(5,2) shows almost a factor of 2 compared to NPS, while PARAID(5,3) shows around 1.5 times of NPS in terms of mean response time. With the real trace, iRGS shows a relatively small performance loss (17%) with only 40% of energy consumption.

### 5. Conclusion

In this work, we developed a system called iRGS (inter-RAID Gear-Shift), which utilizes data replication as a tool for extending the idleness period of a large fraction of the disks. iRGS adapts to the workload
observed by the system, thus allowing energy saving subject to required service level.

For evaluation, we used two workloads: read-intensive synthetic workload and real life Cello99 workload that is write-intensive. The simulation results show that iRGS saves up to 60% of energy with a relatively small performance loss in terms of response time, while existing power management techniques suffer from significantly increased latencies. Future work will include experimenting with a broad set of real life traces in order to study more sophisticated replication algorithms that may provide even better load balancing.

References