# Low Power Mode in Cloud Storage Systems

Danny Harnik, Dalit Naor and Itai Segall
IBM Haifa Research Labs, Haifa, Israel
{dannyh, dalit, itais}@il.ibm.com.

## Abstract

We consider large scale, distributed storage systems with a redundancy mechanism; cloud storage being a prime example. We investigate how such systems can reduce their power consumption during low-utilization time intervals by operating in a low-power mode. In a low power mode, a subset of the disks or nodes are powered down, yet we ask that each data item remains accessible in the system; this is called *full coverage*. The objective is to incorporate this option into an existing system rather than redesign the system. When doing so, it is crucial that the low power option should not affect the performance or other important characteristics of the system during full-power (normal) operation. This work is a comprehensive study of what can or cannot be achieved with respect to full coverage low power modes.

The paper addresses this question for generic distributed storage systems (where the key component under investigation is the *placement function* of the system) as well as for specific popular system designs in the realm of storing data in the cloud. Our observations and techniques are instrumental for a wide spectrum of systems, ranging from distributed storage systems for the enterprise to cloud data services. In the cloud environment where low cost is imperative, the effects of such savings are magnified by the large scale.

## 1 Introduction

### 1.1 Motivation and Background

Energy savings in storage systems has become a key aspect in the design and use of future systems. IT systems today consume about 1-2% of the total energy in the world [1] and this will double in the next 4 years (according to [2], in 2006 US datacenters consumed 1.5% percent of total U.S. electricity consumption). It has been shown that storage systems are the second most important component in the IT after the computing resources, and in some cases (e.g. in large data centers) may consume up to 40% of the total energy [3, 2]. This ratio may grow due to two facts: (i) the power consumption of compute resources has been getting a lot of attention and as a result will become much more efficiently utilized; and (ii) the data deluge (expected 10-fold Growth in 5 Years [4]) will increase the relative contributions of storage systems to the overall power consumption. Thus reducing power consumption has become a central goal when managing data centers. This is ever more crucial in the cloud environment due to the large scale and the critical importance of low operational costs.

Methods for treating power reduction in storage systems that are based on powering off storage devices have been suggested. Some are centered around placing the data wisely (or migrating it dynamically) to allow powering off unused devices - most notable, MAID [8], Popular Data Concentration (PDC [20]), Hibernator [29] and Diverted Access (DIV [21]). Others, (e.g. [19] and Pergamum [23]) employ techniques to delay the access to some of the storage devices. A comprehensive summary of existing methods is given in section 1.4).

In this paper we focus on high scale and distributed storage systems (e.g. [12, 26, 17, 10, 5, 6]). Such systems are architected with built-in redundancy to address requirements like availability, performance and reliability We investigate how such systems can reduce their power consumption during low-utilization time intervals by operating in a *low-power mode*, similarly to the setting studied in [21] (see Section 1.4). We focus on power savings that can be attained while *not compromising* other system characteristics such as high performance and reliability during full-power (normal) operation. The question to address is whether periods of low utilization (nights, weekends etc) can be used to significantly reduce power consumption, and if so - how.

### 1.2 The Model and Problem

Our study is aimed at implementing a low power mode within real world (existing) systems with minimal disruptiveness to the system, utilizing existing mechanisms (i.e. the placement function or the redundancy scheme) whenever possible. The general approach powers down a subset of the disks or nodes in the system during low utilization, yet we require that each data item remains accessible at all times. In low utilization periods, availability and performance may be compromised, while reliability remains essentially the same since the redundant copy(ies) can be recovered.

In our model, the system is comprised of individual *storage nodes* (a storage device like disk or controller, a rack or a even portion of a datacenter) that can be turned on and off independently. Data is divided into *redundancy units* (block, segment, object or file) and each unit's data and redundancy are distributed across a number of different storage nodes according to the system's *placement function*. The other main characteristic is that the storage system has active and non-active utilization periods. The periods of low utilization that we consider are relatively long, for example nights or 5pm-9am, targeting scenarios whereby the spun-down disks are typically not turned on during this period. Although it is known that there are limitations on the number of disk spin up/down cycles, in our model disks are unlikely to get close to this limit (e.g. 50,000). Same applies to the break-even interval[1] - we assume intervals that far exceed it.

This work focuses on studying **existing systems** while minimizing changes to the system and specifically to the **placement function** of the data. Typical designs invest much effort in choosing a placement function that optimizes various aspects of the system such as performance, load balancing, availability, efficient failure recovery and location awareness. These properties are designed to be optimally utilized at peak workloads. During non-active periods we are willing to trade off performance for power saving. However, it is imperative that the option of a low power mode does not compromise the performance of any of the above mentioned properties at full power mode. Hence we strive to keep the placement function unmodified whenever possible. We note that if one is allowed to freely modify the placement function (while ignoring other properties) then there are simple designs which can achieve optimal low power mode as described in [21].

In addition, our goal is to obtain at all times, and in particular during low power mode, a **full coverage** of the data so that access to each data item is always available. This is in contrast to other approaches which attempt to only maximize coverage and are therefore susceptible to unexpected power-ups of disks. For example, models advocated by [8, 20, 29, 25] arrange data according to its access patterns to maximize the coverage of popular data. This has several drawbacks including the need for analysis of the access patterns to the data and the unpredictability of certain access patterns (see further discussion in Section 1.4). In contrast, the full coverage model can handle *any* type of data access pattern, and therefore can be applied to a generic storage system. Moreover, the full coverage approach can support applications like business intelligence and search that require access to every data item. Consider for example, a system that performs most of its writes during the day, while during the nights it performs analytics over the data (e.g. an HDFS [5] or GFS [12] cluster running MapReduce jobs at night). Such analytics would typically require full access for reads but hardly any writes and hence fit the specification of a full coverage low power mode.

[1]The break even interval is the minimal time for which it is beneficial (in terms of power consumption) to spin a disk down.

Finally, the bulk of the paper considers the redundancy scheme to be that of **replication** (each data item has $d$ replicas on $d$ different nodes). However, our results generalize nicely to most instances of erasure codes, (see Section 3.4). That being said, the potential power saving (and the actual savings obtained) are typically much lower than the case of replication, simply because in order to have full coverage, one can only power down redundant data, and the amount of redundancy in erasure codes is typically much lower than in replication (much of the appeal of codes is in their smaller footprint). For example, a RAID 6 mechanism only adds a $50\%$ factor to the data's length, while replication the additive blowup is by at least a $100\%$ (for $d = 2$). Note that the choice of redundancy is treated as a given property of the underlying storage system, and by no mean do we advocate using replication rather than error correction mechanisms. Neither do we encourage using higher replication, since this only increases power consumption. Replication has been a popular choice in designs for storage in the cloud (e.g. in Google's GFS [12], Amazon's Dynamo [10], Hadoop's HDFS [5], Facebook's Cassandra [6] and others) mainly because of the easier handling and managing overheads and better accessibility of such schemes. Therefore, most of the positive results in the paper are more relevant for the setting of cloud storage.

Our analysis focuses mainly on the handling of READ operations. The handling of WRITE operations follows the same mechanisms suggested in previous solutions such as DIV [21] and write offloading [19].

## 1.3 Main Contributions

This paper presents an in-depth study of a range of solutions for low power modes that leverage the existing replication in distributed storage systems. We first consider methods which *do not modify the placement function* of the system and analyze the overhead this may introduce to the system. We then address possibilities of modifications to the placement in order to achieve better power savings.

We introduce a method that obtains full coverage for *any* placement function by using auxiliary nodes. We provide tradeoffs between the replication factor, the amount of auxiliary nodes required and the overall power saving that is attainable. For example, on the negative side we demonstrate that for some systems this approach achieves very little power savings with replication of 2 (mirroring). As the replication factor grows, substantially better results can be obtained; specifically for three copies savings of over 45% is attainable with only 5% additional storage. We analyze this approach when applied to some commonly used systems.

Among our key conclusions are:

- For *every* placement function with replication $d$ and every fraction $p$ there is a subset of size $p$ that when powered down leaves a coverage of all but $p^d$ of the data items. There are placements for which this is the best possible.

- For a generic placement, finding a subset of nodes that yields the best possible data coverage is computationally difficult. Heuristics such as greedy algorithms can be employed; they typically achieve good enough approximations for power saving purposes.

- For random placement mechanisms we show that additional auxiliary-nodes are always required to achieve meaningful savings. Analyzing the random behavior is important since many systems employ placements that are either random or pseudo-random (in order to achieve full balancing and performance), We show a tradeoff between the replication factor and the additional amount of required auxiliary-nodes.

- We analyze one of the most widely used placement functions, the consistent hashing, which has been deployed in systems like CFS [9], Oceanstore [17], Dynamo [10] and Cassandra [6]. We show that the consistent hashing (with or without virtual nodes) naturally accommodates a low power mode and, specifically, is better suited for such power savings than the random placement function.

- In a hierarchical system (e.g. a system consisting of a cluster of nodes with multiple disks within them, a cluster of virtualizers or a cluster of datacenters) it is sufficient to modify only the lower level placement in order to achieve *optimal* power savings.

The results in this paper can be used to suggest whether a real system can be augmented to accommodate low power modes in low utilization periods, and if so how and to what level.

## 1.4 Related Work

This idea of a tradeoff between performance and power consumption has been suggested and deployed in the past. We list works that have studied this:

**Idle disks spin-down:** This is a paradigm used in most power saving models. The idea is simply to spin down a disk once it has been in idle mode for a long time.

There are several methods for helping disks stay idle longer in order to allow their spin-down. Massive array of idle disks (MAID) [8] employ cache disks with recently read data. Popular Data Concentration (PDC) [20] concentrates popular data together. Hibernator [29] combines variable disk speeds and data concentration according to popularity to save power. Several drawbacks of theses approaches are: (i) they are beneficial mainly for systems that contain some rarely accessed data, such as archival systems; (ii) they require some knowledge on the data access pattern; (iii) concentrating the popular data, rather than evenly distributing it, has critical effect on the systems performance; and (iv) if the statistics in the system only accumulate over a long period then power saving might only kick in gradually and will require lots of shifting of data among the storage units.

The Write off-loading technique employed in [19] demonstrates that in some enterprise systems, where the I/O workload is WRITE intensive, handling writes without the need for spin-ups is achievable and does not degrade performance significantly; this allows to increase the idle times for disks and MAID will work better. The Pergamum system [23] adds NVRAMs to each storage node to allow longer spin-down periods for disks in archival systems.

**Exploiting redundancy:** The Diverted Access (DIV) methodology [21] considers replication and as such is the closest in nature to our work. It advocates segregating all the primary copies of the data on a partial set of the disks to allow powering down the rest of the disks and demonstrate the potential power gains by such a modification. We note that this modification of the placement function can potentially hurt the performance of the system in full power mode (see more in Section 5). In contrast, our study focuses on achieving similar gains while minimizing the disruption and ramification on existing system designs. We study solutions that either do not change the placement at all or make only acceptable changes to the data placement.

Other works that exploit redundancy include EERAID [18], RIMAC [28] and PARAID [25]; they study the possibility of power saving within a RAID system. Their solutions apply to settings where the coding is limited to small groups of disks rather than a wider distribution of the redundancy over a whole system. Greenan et al. [13] exploit the redundancy in specific families of erasure codes, a solution that they term power aware coding.

**Notation and Parameters:** Throughout the paper we use the terms ’node’ or ’storage unit’ interchangeably to denote an entity storing a subset of data items that can be shut down independently of the other units. A data item refers to the basic replication unit of data, and in practice can be a block, an object, a stripe or a file. Throughout the paper the term ’data item’ is used regardless of the actual unit in question.

The replication factor of the system is the number of copies stored for each data and is denoted by $d$. $M$ is the number of nodes (or storage units) and $N$ is the number of data items in the system. $p$ refers to the fraction of the nodes that are turned off, and $\alpha_{low}$ is the fraction of time that the low power mode accounts for. We say that a low power mode is *optimal* if one can spin-down all but $1/d$ of the nodes while maintaining at least one live copy of each data unit.

**paper organization:** Section 2 describes the full coverage objective as a graph cover problem; it provides upper and lower bounds on the size of the cover and on the size of the corresponding auxiliary nodes needed for arbitrary placement functions. Section 3 provides the general framework solution; it discusses how to choose a good designated subset of nodes to spin-down and what are the potential power savings. Section 4.1 analyzes the special case of a random placement function and Section 4.2 analyzes the consistent hashing function. In Section 5 we discuss augmenting the placement function while taking low power mode into consideration. In the Appendix we add notes regarding handling heterogenous systems and regarding possible modifications of the placement function.

## 2 The Coverage Problem

In principle, the idea is to find a maximal subset of nodes that is designated to be powered down. The criterion is that the remainder of the nodes should "cover" as many of the data items as possible (hold at least one copy of these data items). The problem at hand can be described naturally as a the following graph problem. The input is a bi-partite graph with $M$ vertices on the left hand side (representing the nodes in the system) and $N$ vertices on the right hand side (representing the data items). Each data item on the right has $d$ edges connected to $d$ distinct nodes on the left (representing the locations on which the $d$ copies of the item reside). For a given fraction $0 < p < 1$ the goal is to find a subset of the nodes of size $Mp$ such that the remainder of the nodes cover a maximal number of data items. A vertex is covered by a subset if it has an edge connecting to a node in the subset. Our ultimate goal is given a graph to find a subset that maximizes $p$ (the fraction of nodes to be powered down) and minimizes the fraction of uncovered data items.

A solution achieves *full coverage* if all data items are covered. However, one can quite simply design a placement function for which there is no full coverage, even for very small $p$. The random placement function is one such example (See Section 4.1). That being said, we show that all placement functions have at least a decent coverage, where decent is a factor of the replication parameter $d$ (The higher $d$ is the better the coverage is).

**Lemma 2.1** *For* every *placement function with replication* $d$ *and every* $0 < p < 1$ *such that* $Mp \in \mathbb{Z}$ *there exists a subset of size* $Mp$ *such that the remaining nodes achieve coverage of at least a* $(1-q(p,M,d))$ *fraction of the data items, where* $q(p,M,d) = \frac{Mp(Mp-1)\ldots(Mp-d+1)}{M(M-1)\ldots(M-d+1)}$.

Note that $q(p,M,d) < p^d$ and tends to this number as $M$ grows.

**Proof:** Fix the placement function and consider a randomly chosen subset of the nodes. Denote by $p_i$ for $i \in [N]$ the probability that all $d$ copies of the $i^{th}$ data item reside in the chose set. Since the set is chosen randomly, the probability is $p_i = \binom{M-d}{Mp-d}/\binom{M}{Mp} = \frac{Mp(Mp-1)\ldots(Mp-d+1)}{M(M-1)\ldots(M-d+1)} = q(p,M,d)$ (the subset must include the $d$ copies and has freedom to choose the remaining $Mp-d$ nodes). This is true for each $i \in [N]$ independently, though the probabilities may be highly correlated. We ask what is $E_{cover}$, the expected number of data items covered by this random set. By the linearity of expectations, $E_{cover} = N - \sum_{i\in[N]} p_i > N - Nq(p,M,d)$. Since this is the expectancy over all possible subsets, then there exists at least one subsets that covers $N - Nq(p,M,d)$ items (otherwise the expectancy would be smaller). $\square$

We note that the proof of Lemma 2.1 is non-constructive in the sense that it gives no hint of which subset gives a good cover, only that one such subset exists. The parameters achieved in Lemma 2.1 are also tight. This is seen by the following Claim:

**Claim 2.2** *There exists placement functions for which every subset of size* $Mp$ *of the nodes yields at least a* $q(p,M,d)$ *fraction of uncovered data items.*

**Proof:** The proof is by the following simple example. Consider a system with $M$ nodes and $N = \binom{M}{d}$ data items (for small $M$ and $d$ this is a reasonable parameterization). Define a one to one mapping between the data items and subsets of size $d$ of the nodes (a replica of the data item is stored in each of the nodes in the corresponding subset). Now for each such subset there is exactly one data item. Any designated subset of size $Mp$ leaves exactly $\binom{Mp}{d}$ uncovered data items. Finally observe that $\frac{\binom{Mp}{d}}{N} = \frac{Mp(Mp-1)\ldots(Mp-d)}{M(M-1)\ldots(M-d)} = q(p,M,d)$ $\square$

**Consequences:** The above statements can be viewed as both positive and negative results. On the one hand, it shows that it is possible to achieve a coverage of all but a $p^d$ fraction of the data items by shutting down a $p$ fraction of the nodes, regardless of the placement function. For some practical parameters this achieves a decent saving in power as will be demonstrated in Section 3.1. On the other hand, this also forms the limit on possible savings for some placement schemes. Therefore, in some settings, such as a random placement function with replication $d = 2$, the approach of not changing the placement function seems futile.

**Finding the optimum is hard:** We next argue that finding the best coverage in a generic graph is a computationally hard problem. This is seen by yet another graph representation of the problem, this time as the problem of *vertex-cover in a hypergraph*. Consider a hypergraph that has $M$ vertices (each representing a node in the system). There are $N$ hyperedges in the graph, each connecting $d$ vertices (the edges represent data items and the hyperedges connect all the nodes on which they reside). Solving the coverage problem is equivalent to finding a small as possible subset of vertices that touches as many as possible of the edges. In the special case of $d = 2$ this translates to the classical vertex cover problem (one of Karp's original 21 NP-hard problems [16]). The general problem with constant $d$ was further shown to be hard to approximate, (see [11] and references within). We conclude that finding the maximal cover is a computationally hard problem and therefore we shall resort to heuristics and approximations in order to find a good subset (see Section 3.1).

**The cover problem in a dynamic system:** Note that the ability (or inability) to find the optimum is not crucial in our specific scenario. The reason is that the systems we deal with may be very dynamic, and the underlying graph for the cover problem varies as time goes (data items may come and go and new nodes are added and removed). These variations change the optimal solution, so the best solution today may not be as good tomorrow. On the other hand, since the changes are mostly local, they typically don't change the situation by much. That is, a good solution will likely remain a good one for quite a while. Therefore, our objective is to find a good solution to the cover problem (not necessarily the best one) and use it for a while.

## 3 A General Framework for Low Power Mode with Existing Placement

We have established that for some placement functions one cannot find a meaningful subset with full coverage. Namely, when powering down a subset of nodes, we are bound to leave some data items uncovered. To avoid this, we introduce the notion of *auxiliary nodes* to the system as a means of achieving a meaningful low power mode yet having full coverage of the data at all times (thus avoiding undesired spin-up delays). The point in introducing auxiliary nodes is that by Lemma 2.1, we know that the number of auxiliary nodes can be significantly smaller than the number of nodes to be powered down ($\frac{p^d}{d}$ compared to $p$). Thus we can substitute a large number of nodes with a small number of auxiliary nodes and still attain full coverage.

We next give our general framework at a high level, leaving specific details to be filled in for specific implementations and placement functions being deployed.

1. **Find a good designated subset:** The first step is to find a designated subset that gives a good solution to the cover problem. This step can be done using numerous heuristics (see Section 3.1). Choosing the appropriate $p$ depends on various parameters of the system as discussed in Section 3.2.

2. **Construct the auxiliary nodes:** Once a designated subset has been chosen, we construct the auxiliary nodes. This pool of extra nodes contains an additional copy of data items that are bound to be uncovered in low power mode (thus the system holds $d + 1$ copies of these data items). More precisely, a data item has a copy in the auxiliary nodes iff all of its $d$ copies reside in the designated subset.

3. **Low power mode:** To go into this mode one should simply shut down all of the nodes in the designated subset.

   - **Read operations:** On a read operation access a live copy in the regular nodes, if it does not exist then redirect to the auxiliary nodes.
   - **Write operations:** Writes are performed regularly to all live nodes (this includes writes to the auxiliary nodes). In addition, all write operations that involve the nodes in the designated set are recorded in a log, in order to be completed during power up. In case the log fills up, we can selectively power-up nodes in the designated subset in order to relieve the write log. See further discussion on writes in Section 3.3.

4. **System wake up:** Apply write log. At the end of this process all of the data items should have their $d$ copies in place (according to the placement function).

5. **Full power mode:** The system runs in usual operation with the one exception that the auxiliary nodes are maintained in an online fashion. That is, each write to a data item all of whose copies are in the designated subset is actually performed $d + 1$ times. $d$ times for the regular copy and an extra write in the auxiliary nodes.

In addition to the above, a typical system would require a periodical refresh to the designated subset. This may be invoked due to notable changes, such as a noticeable data distribution change or addition/substraction of nodes. One may also want to change the subset due to power cycling considerations. Since spin-ups/spin-downs of disks shorten the disks life time, it may be beneficial to modify the set of disks being spun down and thus achieve a longer mean time to disk failure (MTDF).

Finally, in case of node failure during low power mode, the system must wake up all the nodes that are required for the recovery of the lost data. Depending on the severity of the failure and the placement mechanism, this may result in powering-up anywhere between one node to the whole system.

## 3.1 How to Choose the Subset

There are various methods one can use for choosing a good designated subset. In some cases, the placement function dictates a simple deterministic choice that achieves very good coverage (see example in Section 4.2), while for others we need to use various search mechanisms. In general, any good approximation heuristic for the cover problem may be used here, where the effectiveness of the techniques may vary greatly depending on the underlying placement mechanism. Other than the quality of the cover being achieved, one should take into consideration also the efficiency of the technique, with an emphasis on the ability to run it in a distributed system. We consider two main approaches here.

- **Choosing random subsets.** This method is the simplest to implement and will typically provide a solution that is close to the expected value (namely, approximately $p^d$ uncovered data items). A simple to implement improvement is to sample a number of random subsets and to take the one which gives the best coverage. With very high probability this yields a result that is better than the expected value.

- **A greedy algorithm.** In a nutshell the greedy technique iterates adding single nodes to the subset, in each step adding the best current candidate to the subset. For example, one approach is to choose nodes for the non-designated (power-up) subset. At each step we choose the single additional node that covers the most data items that were not covered thus far. Greedy algorithms have proved quite a successful heuristic in covering problems such as ours. In fact, Sümer [24] (Corollary 3.9) shows that in our setting (of vertex-cover on hypergraphs) the greedy algorithm approximates the best solution to within a constant factor.

  While the greedy algorithm is quite feasible for almost any setting, its efficiency may vary depending on the system architecture. If conducted in a centralized manner, it requires enumerating all available nodes and finding the best choice in each iteration. Alternatively, the nodes themselves can compute their current cover status, and report this to a central component that is then required to simply find the maximum. This requires the central component to broadcast the list of currently chosen nodes to all of the remaining nodes, at each iteration.

## 3.2 Choosing the fraction $p$

A central question in choosing the designated subset is to decide what its size should be. We give an analysis of the best choice for the general case, in which the uncovered set is of size $Np^d$. This approach can be modified accordingly for various placement functions. To obtain maximal saving we need to model the power savings gained from the suggested low power mode. In our basic simplified model we assume that all nodes consume the same average power during operation mode, and do not consume power when turned off.[2] The actual turning on and off has some additional cost in terms of power, but since we assume that the low power mode is for significantly long time period (at least several hours), we can discard this factor in the calculation. Suppose that the low power mode accounts for an $\alpha_{low}$ fraction of the time. Viable options for this parameter can be $\alpha_{low} = \frac{1}{3}$ (for 8 hours of daily nighttime), or $\alpha_{low} = \frac{128}{168} = \frac{16}{21}$ (accounting for all but 40 of weekly work hours – $9$ to $5$ on weekdays). We ask what is the power saving as a function of the chosen $p$. There are two factors to consider:

1. The spun down disks – this accounts for $\alpha_{low}p$ of the overall power consumption. In actual computations we should estimate $\alpha_{low}$ as a little smaller than what we actually expect it to be (e.g., take $\alpha_{low} = 0.3$ rather than $\frac{1}{3}$). This substraction comes to compensate for additional costs, such as the power required for spinning-up/down disks or unplanned power-ups due to overload of writes or disk failures.

2. The additional auxiliary disks – these disks run at all times and should cover a $p^d$ fraction of the data. This amounts to $\frac{M}{d}p^d$ nodes, which consume a fraction of $\frac{p^d}{d}$ of the overall power.

The total power saving is thus $Sav(p) = \alpha_{low}p - \frac{p^d}{d}$. To find the maximum value we derive this function and find its zero value. That is $\alpha_{low} - p_{max}^{d-1} = 0$ and therefore $p_{max} = \alpha_{low}^{\frac{1}{d-1}}$. Table 1 shows some selected values that come up from this choice.

As seen in Table 1, the maximal saving may require quite a large number of auxiliary disks (e.g., for $\alpha_{low} = \frac{16}{21}$, $d = 3$ and $M = 1000$, the best saving requires approximately $216$ auxiliary nodes). Alternatively, one can set the number $M_{aux}$ of available auxiliary nodes, and derive the best attainable $p$ given this $M_{aux}$ according to $p = \left(\frac{dM_{aux}}{M}\right)^{\frac{1}{d}}$. For example, with $M_{aux} = 10$, $M = 1000$, $d = 3$ and $\alpha_{low} = \frac{16}{21}$, the power down fraction can be $p \approx 0.31$. This

[2]This assumption can be partly justified by the fact that the actual spinning of the disks accounts for majority of the power consumption of the storage node (see, e.g. [8]). Thus the overall power consumption during various utilizations does not vary greatly.

| $\alpha_{low}$ | $d$ | $p_{max}$ | $\frac{M_{aux}}{M}$ | Saving % | Saving % low time |
|---|---|---|---|---|---|
| 1/3 | 2 | 0.3 | 0.045 | 5 % | 15 % |
| 1/3 | 3 | 0.547 | 0.054 | 11 % | 37 % |
| 1/3 | 4 | 0.669 | 0.050 | 15 % | 50 % |
| 1/3 | 5 | 0.740 | 0.044 | 18 % | 59 % |
| 16/21 | 2 | 0.750 | 0.281 | 28 % | 38 % |
| 16/21 | 3 | 0.866 | 0.216 | 43 % | 58 % |
| 16/21 | 4 | 0.908 | 0.169 | 51 % | 68 % |
| 16/21 | 5 | 0.930 | 0.139 | 56 % | 74 % |

Table 1: For two possible workload scenarios $\alpha_{low} = 1/3$ (8 hours nightly low power mode) and $\alpha_{low} = 16/21$ (low power mode outside working hours), and various options of the replication constant $d$ we compute $p_{max}$ that maximizes the power saving. For this value we compute the ratio between the number of required auxiliary nodes $M_{aux}$ and the number of regular nodes $M$ and estimated total power saving percentage and power saving percentage for each hour of low power mode. We see that the saving is quite low for $d = 2$ and improves drastically as $d$ grows.

| $\frac{M_{aux}}{M}$ | $\alpha_{low}$ | $d$ | $p$ | Saving % | Saving % low time |
|---|---|---|---|---|---|
| 0.01 | 1/3 | 2 | 0.75 | 3 % | 11 % |
| 0.01 | 1/3 | 3 | 0.311 | 8 % | 28 % |
| 0.01 | 1/3 | 4 | 0.447 | 12 % | 41 % |
| 0.01 | 16/21 | 2 | 0.75 | 10 % | 13 % |
| 0.01 | 16/21 | 3 | 0.311 | 22 % | 30 % |
| 0.01 | 16/21 | 4 | 0.447 | 33 % | 43 % |
| 0.05 | 1/3 | 3 | 0.531 | 11 % | 36 % |
| 0.05 | 1/3 | 4 | 0.669 | 15 % | 50 % |
| 0.05 | 16/21 | 2 | 0.316 | 19 % | 25 % |
| 0.05 | 16/21 | 3 | 0.531 | 35 % | 46 % |

Table 2: We fix the fraction of auxiliary nodes in the system $\frac{M_{aux}}{M}$ to two values ($0.01$ and $0.05$) and study for various replication constants $d$ and workload scenarios $\alpha_{low}$ what is the fraction $p$ of the power down set and estimate the total power saving percentage and saving percentage during low power mode. We see that some options yield only minor saving (such as with $d = 2$), but for instance for $d = 3$, $\alpha_{low} = 16/21$ and $5\%$ of additional auxiliary nodes, a saving of $46\%$ during down time is achieved. This improves as $d$ grows.

amounts to a saving of approximately $0.3$ of the power consumption during the low power mode (as opposed to $0.58$ with $p_{max} = 0.547$). While not the best possible, this may yield decent savings at a much lower price in terms of auxiliary nodes. In Table 2 we list some possible parameters using this approach.

## 3.3 Discussions

**Auxiliary nodes vs. partial coverage:** We make a case for using auxiliary nodes to cover the remaining uncovered data (in a sense, creating a $d+1$ copy for these data items). The alternative could be to leave a fraction of data uncovered and to risk unexpected spin-ups on reads from this fraction of the data. We have several justifications for our approach: (i) In order to cover a fraction $q$ of the uncovered data, one only needs to use a fraction of $\frac{q}{d}$ of the nodes. This factor of $d$ maybe quite substantial. For instance, with $d = 3$ and $p = 0.53$, the use of auxiliary nodes amounts to a $5\%$ increase in the capacity, whereas the alternative is to risk $15\%$ of the data being uncovered. (ii) It is tempting to hope that the $q$ fraction of uncovered data can be made to contain mostly unpopular data. However, in the cases we studied (i.e., random placement and consistent hashing) we see that this fraction is randomly distributed across all data items, which effectively means that the uncovered data will contain an equal percent of popular data as the whole system.

Another alternative is to modify the placement scheme just for the fraction of uncovered data. The complexity of such a solution can be small as long as this fraction $q$ remains small. We note however that in this solution one does not have the $d$ factor gained by using auxiliary nodes (i.e., a $q/d$ fraction of the nodes to cover a $q$ fraction of the data).

**Handling write operations:** Unlike read operations, write operations need to access *all* of the replicas of a data item. This will not be possible in low power mode and the system must adjust accordingly. Our solutions do not deviate from those of [21] and [19]. All writes to live nodes are executed, including writes to the auxiliary nodes. However, unlike with read operations, most writes will have at least one replica in the powered down set. These writes are placed in a log file and flushed during power up (the auxiliary nodes may serve for this purpose as well). Because of this, such a low power mode is not designed to handle a high volume of write operations in low power mode. Regarding data reliability: because of the full coverage property, every write operation updates at least one copy and an additional copy is placed in the log mechanism. Thus at least two replicas are saved during the low power mode and this is complemented to $d$ at power up. Assuming the log mechanism is more reliable than average (e.g. use NVRAM for this purpose) this may suffice in terms of reliability until full power mode is restored.

## 3.4 The Case of Erasure Codes and Varied Availability

The use of replication, while simpler to manage, is quite wasteful in terms of capacity. Similar or even greater reliability against failures can be achieved by using erasure correction mechanisms. In a typical instantiation, the data object is translated into $d$ segments, each stored on a different node or failure domain, and any $k$ of these segments suffices to reconstruct the original data. Such a system can therefore withstand failure to $d - k$ nodes without losing any data. Thus, in this case, full coverage would require that of any $d$ segments of a single item, at least $k$ would be powered up – we call this a $(k, d)$-low power mode. This model is relevant to most coding schemes being deployed.[3] For example, a scheme based on the RAID 6 encoding has full coverage in a $(4, 6)$-low power mode. Note that the potential power saving in such a system is limited to $\frac{k}{d}$. That is, an optimal solution of a $(4, 6)$-low power mode can power down at most $\frac{1}{3}$ of the nodes without losing coverage.

A different justification for this problem comes from systems that do use replication but a single available copy is not sufficient even in low power mode, due to availability requirements. Rather, the system is willing to have a reduced number of copies during low power mode. For example, a system that stores 8 copies and will make only 3 available copies during low utilization periods.

The results discussed for the case of replication can be generalized for the $(k, d)$ model as well. The change is in the number of required auxiliary nodes as a function of $d$ (recall that previously this was $q(p, M, d)/d = \frac{p^d}{d}$). The new probability $q'(p, M, d, k)$ for uncovered data is as follows:

$$q'(p, M, d, k) = \sum_{i=0}^{k-1} (k - i) \binom{d}{i} \frac{\binom{M-d}{Mp-d+i}}{\binom{M}{Mp}}$$

Note that when $M$ is large, then $\frac{\binom{M-d}{Mp-d+i}}{\binom{M}{Mp}}$ tends to $p^{d-i}(1 - p)^i$. The number of required auxiliary nodes is then a $q'(p, M, d, k)/d$ fraction of the regular nodes.

In Table 3 we show the numbers for some choices of $k$ and $d$. As expected, the power saving potential when using error correction codes is quite low, and specifically for RAID 6 seems hardly worthwhile. The situation improves as the ratio between $d$ and $k$ grows.

## 4 Analysis of Specific Placement Functions

We turn to a more in depth analysis of the situation for specific placement functions. Following are studies of two prominent strategies, a random placement function and the consistent hashing approach.

[3]There exist schemes in which the number of segments required for reconstruction varies according to which segments are obtained. Such schemes to not work well with a general $(k, d)$-low power mode design.

| Redundancy type | $\frac{M_{aux}}{M}$ | $\alpha_{low}$ | $p$ | Saving % low time |
|---|---|---|---|---|
| (4,6) | 0.01 | 1/3 | 0.16 | 13 % |
| (4,6) | 0.05 | 1/3 | 0.28 | 13 % |
| (2,5) | 0.01 | 1/3 | 0.34 | 31 % |
| (2,5) | 0.05 | 1/3 | 0.52 | 37 % |
| (2,8) | 0.01 | 1/3 | 0.56 | 53 % |
| (2,8) | 0.05 | 1/3 | 0.73 | 58 % |
| (3,8) | 0.01 | 1/3 | 0.51 | 48 % |
| (3,8) | 0.05 | 1/3 | 0.66 | 51 % |

Table 3: The table presents the power saving potential in $(k, d)$-low power modes for various choices of $k$ and $d$. We see that for ratios typical of erasure codes such as $k = 4$ and $d = 6$ the power savings are quite low and hardly worth the effort involved. Things improve as the ratio of $d/k$ grows.

### 4.1 The Random Placement Function

This scheme refers to a random (or pseudorandom) allocation of locations for each of the replicas. More precisely, each data item is stored in $d$ distinct disks that are chosen according to the random distribution. In practice, some modifications are typically required to avoid too much of a disparity between the number of data items allocated at a single disk. E.g., if one disk is filled up, we should avoid allocating new replicas to this disk. These modifications form very weak correlations between the placements of different data items, typically only when disks become full. For our purposes, we can consider the choices as fully independent.

The importance of random placement functions is that they have very good properties (such as load balancing, quick recovery from failure, etc...) and are therefore advocated by several systems. For example GFS [12], FARSITE [7] and RUSH/CRUSH [14, 27] try to achieve random or pseudorandom placement.

For a random placement function we show that for *every designated subset* of size $Mp$, the expected number of uncovered data items is $Nq(p, M, d)$, where expectancy is over the random choice of the placement.[4] Recall that this amounts to approximately a $p^d$ fraction of the data items. To see this, fix the designated subset and for each data item compute the probability that all $d$ copies are inside this set. The probability is $\binom{Mp}{d}/\binom{M}{d}$ which is exactly $q(p, M, d)$.

The remaining question is how much does such a subset deviate from the expectancy, or in other words, how good or bad can the best coverage be for a single instantiation of the placement function. In Figure 1 we exhibit the situation for one instantiation of the random placement, which shows a nice normal looking distribution around the mean. That is, taking a random subset is likely to result in a coverage that

[4]In Lemma 2.1 we show that there exists at least one subset that achieves this coverage, while here we show that this is the case on the average.

is around the expected value $q(p, M, d)$.

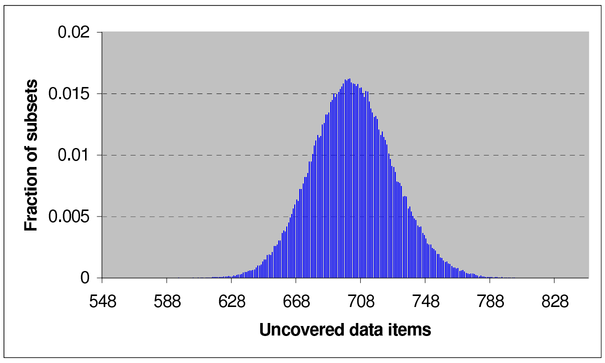

Figure 1: This is a system with random placement function, $M = 20$ nodes, replication $d = 3$ and $N = 6666$ data items. The fraction of the power down set is $p = 1/2$. The figure plots for each number of uncovered data items, what is the fraction of subsets of size $Mp$ that yield this coverage. The calculated expectancy for uncovered items is $Nq(p, M, d) \approx 702$ which fits the peak nicely.

We ran experiments to evaluate the success of the various heuristics for choosing a good designated subset. Results (appear in full version) exhibit that the "best of 10 random subsets" method achieves results that are marginally better than the average, similar to what one would expect from a random normal distribution. The greedy tests that we conducted achieved an improvement of a constant factor over the expected. This factor was more significant for lower $p$.

We conclude that the random placement function does not accommodate a low power mode readily, and requires the addition of auxiliary nodes in order to achieve full coverage. When using auxiliary nodes, taking a random subset is a good option as it does not deviate by much from the optimal solution. Using a greedy algorithm is helpful in reducing the number of necessary auxiliary nodes to be closer to the optimum.

## 4.2 Consistent Hashing

This placement scheme, introduced in [15] has proved useful in Peer-to-Peer networks (the Chord system [22]) and is the underlying placement function for a number of distributed storage systems (e.g., CFS [9], Oceanstore [17], Dynamo[10]).

**The basic consistent hashing placement:** The placement algorithm operates on a fixed size range, e.g. the range of 128 bit strings (that is $2^{128}$ values). This range is viewed as a cyclic ring (where the value $2^{128} - 1$ is followed by the value 0). Each of the $M$ nodes in the system is mapped (pseudo-)randomly onto this ring, typically using a hash function (like SHA1). According to this mapping, the ring is divided into $M$ regions of consecutive values on the ring. Each region consist of consecutive values between two neighboring nodes on the ring and is attributed to the node at the beginning of the region (we say that a region is "owned" by the corresponding node). In order to place a data item, its name (or contents) are also hashed onto a point on the ring. The data item is stored in the node owning the region which it hit. Additional replicas are placed on the next $d - 1$ consecutive regions (or nodes owning the next regions).

The consistent hashing scheme defined above has nice and desired properties, but is suspect to problems regarding load balancing (some nodes are bound to be very light) and failure from recovery (limited in its ability to parallelize). The standard technique [15] to overcome this is to employ a method where each node owns a collection of **virtual nodes** rather than one node. Now the load of a node is averaged over its various virtual nodes. When the number of virtual nodes per actual node is on the order of $\log M$, then the load balancing becomes acceptable.[5]

**Initial observation: perfect solution with no virtual nodes.** The consistent hashing placement is ideal for a low power mode when *no* virtual nodes are used. This is simply because one can power down $d - 1$ out of every $d$ consecutive nodes on the ring and still guarantee that one copy is always active. If the number of nodes is a multiple of $d$ then this achieves the optimal energy saving with full coverage. That is, exactly $1/d$ of the nodes need to stay alive (one copy of each data block). The solution is close to optimal if $M$ is not a multiple of $d$.

**Solutions with virtual nodes.** We turn to the more challenging yet more realistic case in which virtual nodes are used. The perfect solution described above no longer works, since powering down an actual node corresponds to removing a number of virtual nodes, whose locations on the ring are random, and in particular, do not behave nicely when looking at the ordering modulo $d$. Now it is unclear that for a high fraction $p$, there exists a full coverage at all. Still, for a random choice of subset of size $Mp$, the probability that a single data item is left uncovered is again $q(p, M, d) \approx p^d$. Our experiments show that as the number of virtual nodes grows, the coverage behaves closer and closer to the coverage in a random placement.

We now focus on systems where the number of virtual nodes is approximately $\log M$, the amount advocated by [15]. The question is at what value of $p$ can the system still achieve full coverage, and moreover, how does the coverage behave beyond this $p$. In Figure 2, we learn that this fraction is surprisingly high and grows as the replication parameter $d$. For instance, with $d = 3$ it is possible to shut down $35\%$ of the nodes and still yield full coverage (with no auxiliary

[5]The use of virtual node also comes in handy when considering nodes of different capacities. A strong node can own a larger number of virtual nodes than a weak one.

nodes at all).

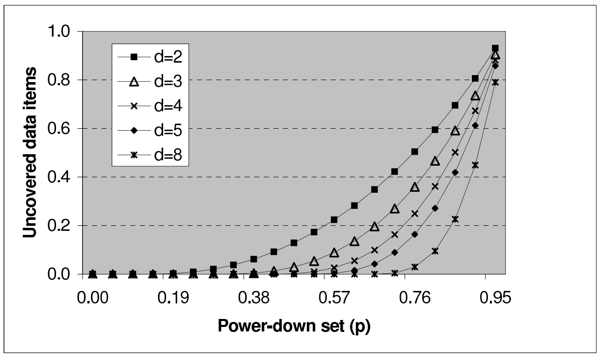

Figure 2: We evaluate the success of the greedy algorithm in a consistent hashing system as a function of the fraction of the power-down subset $p$. We test this on a system with $M = 4096$ nodes $\log M = 12$ virtual nodes. The experiment shows that we can shutdown a high percentage of nodes and still obtain full coverage, with this percentage growing substantially with $d$.

| $\frac{M_{aux}}{M}$ | 0 | | 0.01 | | 0.05 | |
|---|---|---|---|---|---|---|
| | p | Save | p | Save | p | Save |
| $d = 2$ | 0.17 | 17 % | 0.32 | 29 % | 0.52 | 37 % |
| $d = 3$ | 0.35 | 35 % | 0.5 | 47 % | 0.65 | 50 % |
| $d = 4$ | 0.46 | 46 % | 0.6 | 57 % | 0.73 | 58 % |
| $d = 5$ | 0.54 | 54 % | 0.66 | 63 % | 0.77 | 62 % |

Table 4: For various values of the replication constant $d$ we provide the fraction $p$ of the powered down disks that can be covered without, with $1\%$ or with $5\%$ of auxiliary nodes in a consistent hashing system. We also provide the power saving percentage during the low power mode interval. The savings are estimated according to night time low power mode ($\alpha_{low} = 1/3$). The data is according to tests run with $M = 1024$ and $M = 4096$ nodes with $\log M$ virtual nodes. We learn that the savings with consistent hashing are much more significant than in the general case. This is especially evident for $d = 2$ (see Table 2 for comparison).

We next ask what is the effect of using auxiliary nodes in the system. In Table 4 we extract some key numbers to demonstrate this behavior. We see that the introduction of a small number of auxiliary nodes is beneficial in terms of power savings. This effect is highest for the low replication constants ($d = 2$ and $d = 3$). We observe that for every $d$, there exists a fraction beyond which adding more auxiliary nodes is no longer beneficial in terms of power saving. It is interesting to note that for the high replication values this break even point is quite low.

We conclude that the consistent hashing placement is better suited for low power mode than the random placement function. This can be explained as a result of the strong correlations in placement to nodes produced by this scheme. We also learn that for this scheme, the greedy algorithm is significantly more successful than the choosing random subsets.

## 5 On Augmenting the Placement Function for Low Power Mode

The results presented in Sections 2,3 and 4 illuminate the limitations of working with a given placement function that does not readily accommodate a low power mode. The alternative is to modify the placement function so that it fits the power saving model. This approach was pioneered in [21] with the Diverted-access (DIV) idea. They suggest to simply segregate the primary copies of the data items on a specific set of disks and so the rest of the disks may be powered down without damaging the full coverage of the system. This achieves an optimal full power mode in the sense that one can power down $\frac{d-1}{d}$ of the nodes, leaving exactly one copy of each data item alive.

The challenging question is how to design a placement function that has a multitude of desired properties, rather than just a good power saving design. Note that if power saving is the only goal then a simple system in which each disk is mirrored $d$ times (as a whole disk), accommodates an optimal low power mode. The wider goal is to combine the possibility of optimal low power mode with other capabilities.

The requirement for optimal low power mode is summarized by the need that a $\frac{1}{d}$ fraction of the nodes will be designated for the first replicas of each data item. This may entail a compromise in some of the subtle properties of the system and raise new considerations that should be addressed. For example, there may be considerations such as location awareness that dictate the placement function. Another example is the ability for parallel recovery from failure. The required modification to the placement function limits the possible degree of parallelism when recovering from a node failure; e.g. for $d = 2$ parallelizing the recovery is limited to $\frac{1}{2}$ of that of a fully random system. There may also be considerations such as location awareness that dictate the placement function that should not be toyed with.

In the following we discuss a setting (that of hierarchical systems) in which minimal modification suffices for optimal low power mode.

### 5.1 Low Power Mode in Hierarchical Systems

Many common storage systems have a hierarchical structure, typically of two levels (and sometimes more). We look at systems which have a top level that divides the lower level nodes into correlated failure domains. For example, disks connected to the same controller or nodes on the same rack, cabinet, or even data center. As such, the placement specification requires that the $d$ copies of data item be placed on different failure domains (nodes of the top level). For this discussion we call nodes at the upper level simply as "nodes" and lower level nodes as "disks" (although these may actually consist of anywhere from a disk to a whole system).

The observation is that in such systems we can have a low power mode in which partial nodes are spun-down (i.e., part of the disks in every node are spun down). In order to achieve an optimal low power mode we need the number of disks per node to be a multiple of the replication parameter $d$.

The basic idea is to form a correlation between the locations of the different replicas within their respective nodes. In the most basic setting, for each data item we designate a single replica to be the "always-up" replica. This choice is made by a random coin flip where each replica is given equal probability. Within each node, we divide the disks into two groups, where $\frac{1}{d}$ of the disks are designate for the always-up replicas and the rest are for the other replicas. Now within each node the always-up replicas are all located on the designated disks. According to the random choice, this should account for $\frac{1}{d}$ of the blocks stored on the node and should therefore fit in the designated disks. During low power mode, all but the designated disks are spun-down. The main benefit of this method is that the placement function across nodes (which replica resides on what node) can be left unmodified, and only the placement within a node is touched. Even more so, the distribution within each single node remains the same, and the correlations are only observed when looking at the placement within two nodes or more.

We view this as a real option for power saving with limited modification of the overall placement function. Note that hierarchical systems are more common than not, and this approach gives optimal saving and is effective also for low replication such as $d = 2$. When taken to its extreme, one can think a network of numerous data-centers. By correlating the placement within different data-centers (yet keeping the placement of replicas across data-centers) we can power down a $\frac{d-1}{d}$ fraction of all data centers in the network and still maintain full coverage.

## 6 Conclusions and Future Work

This paper studies what can be achieved by and the limitations of a family of methods that attempt to reduce power consumption during low power modes by shutting down some of its storage units.

Several directions can be investigated further. One natural direction is to extend this study to other prominent systems and placement functions. It will also be interesting to understand the consequences of adding data popularity aspects to the placement, an approach discussed in [21]. More precisely, can popularity based power saving methods benefit from the existing redundancy and also work with a given placement scheme.

**Acknowledgments:** We thank Michael Factor for helpful discussions.

## References

[1] Green IT: A New Industry Shock Wave, Gartner Symposium/ITxpo, October 2007, .

[2] EPA Report on Server and Data Center Energy Efficiency, Public Law 109-431, U.S. Environmental Protection Agency, ENERGY STAR Program `http://www.energystar.gov`.

[3] StorageIO, Greg Sculz, `http://www.storageio.com`.

[4] The Diverse and Exploding Digital Universe, An IDC White Paper - sponsored by EMC, `www.emc.com/collateral/analyst-reports/`.

[5] The Hadoop Distributed File System: Architecture and Design, `http://hadoop.apache.org/`.

[6] A. Lakshman, P. Malik, and K. Ranganathan. Cassandra: A Structured Storage System on a P2P Network, product presentation at SIGMOD 2008.

[7] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *OSDI*, 2002.

[8] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *SC*, pages 1–11, 2002.

[9] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *SOSP*, pages 202–215, 2001.

[10] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, pages 205–220, 2007.

[11] I. Dinur, V. Guruswami, S. Khot, and O. Regev. A new multilayered pcp and the hardness of hypergraph vertex cover. In *35th ACM STOC*, pages 595–601, 2003.

[12] S. Ghemawat, H. Gobioff, and S. Leung. The google file system. In *SOSP*, pages 29–43, 2003.

[13] K. Greenan, D. Long, E. Miller, T. Schwarz, and J. Wylie. Spin-up saved is energy earned: Achieving power-efficient, erasure-coded storage. In *HotDep08*, 2008.

[14] R. Honicky and E. Miller. Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. In *IPDPS*, 2004.

[15] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *STOC*, pages 654–663, 1997.

[16] R. Karp. Reducibility among combinatorial problems. *In* **Complexity of Computer Computations**, *edited by R. Miller and J. Thatcher, New York: Plenum Press*, pages 85–103, 1972.

[17] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *ASPLOS*, pages 190–201, 2000.

[18] D. Li and J. Wang. Eeraid: energy efficient redundant and inexpensive disk array. In *ACM SIGOPS European Workshop*, page 29, 2004.

[19] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. In *FAST*, pages 253–267, 2008.

[20] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *ICS*, pages 68–78, 2004.

[21] E. Pinheiro, R. Bianchini, and C. Dubnicki. Exploiting redundancy to conserve energy in storage systems. In *SIGMETRICS/Performance*, pages 15–26, 2006.

[22] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

[23] M. Storer, K. Greenan, E. Miller, and K. Voruganti. Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage. In *FAST*, pages 1–16, 2008.

[24] Ö. Sümer. Partial covering of hypergraphs. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 572–581, 2005.

[25] C. Weddle, M.Oldham, J. Qian, A. Wang, P. Reiher, and G. Kuenning. Paraid: A gear-shifting power-aware raid. *TOS*, 3(3), 2007.

[26] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *OSDI*, pages 307–320, 2006.

[27] S. Weil, S. Brandt, E. Miller, and C. Maltzahn. Grid resource management - crush: controlled, scalable, decentralized placement of replicated data. In *SC*, page 122, 2006.

[28] X. Yao and J. Wang. Rimac: a novel redundancy-based hierarchical cache architecture for energy efficient, high performance storage systems. In *EuroSys*, pages 249–262, 2006.

[29] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: helping disk arrays sleep through the winter. In *SOSP*, pages 177–190, 2005.

## A On Heterogenous Systems

Sections 3 and 4 focussed on systems of homogenous structure. Namely, systems that contain identical nodes, so that powering down one node has the same effect as powering down another. This is quite a reasonable assumption for small to medium sized storage systems. However, it is not necessarily the case for large scale systems (e.g. [17, 26]). Nodes may consist of various machines with various capacities, capabilities and power consumption. This may have a real effect on the power savings that a low power mode can introduce. Specifically, the imbalance in node sizes needs to be taken into account when choosing a spin-down subset. We describe a greedy algorithm that takes into account the properties of the various nodes, trying to optimize the solution in such a system.

Each node $j \in [M]$ is associated with the parameter $P_j$ which corresponds to the average power consumption of this node in a light workload setting (night time). The greedy algorithm is as follows:

1. Initialize the live set $L = \emptyset$
2. Enumerate for $(M - Mp)$ rounds (the total number of live nodes).
   - For every node $j \in [M]\backslash L$ calculate $C_j^L$, the number of data items covered by $j$ and not covered by $L$.
   - Add to $L$ a node $j$ with minimal $\frac{P_j}{C_j}$.

Note that $\frac{P_j}{C_j}$ is the power loss per data item that results from keeping node $j$ alive.

As a stoping criteria, rather than setting $p$ in advance, one can choose the number of available auxiliary nodes $M_{aux}$ and continue the process until the fraction of data items not covered by $L$ is at most $q = \frac{M_{aux}}{dM}$.

## B Issues Related to Augmenting the Placement Function for Low Power Mode

In the following we mention some issues and design points in modifying the placement function.

**Treating primary replicas:** In some systems, the primary copy of a data segment is also the most frequently accessed copy. If all of the primary copies are segregated together on the same partition of nodes, then this leads to congestion on these nodes. To overcome this, it is suggested to decouple the replica-number and its placement. For each data item there should be a random and independent choice of which replica should remained powered-up. This allows the primary replicas to remain distributed evenly across all nodes. The access should be as follows: during full power mode the primary copy is accessed; in low power mode the 'live copy' is accessed, where the live need not be the primary copy.

**Power cycling – When and How:** A drawback in solutions that spin disks up and down is that this action increases the wear and tear on the disk and shortens its life time. In fact, some disks carry a limit on the number of spin-ups/spin-downs that they can handle in a life time (e.g., 50,000). Power cycling refers to alternating between the disks being spun-down in order to avoid extensive spin-downs and spin-ups of a limited set of disks. This, in turn, will provide a longer mean time to failure of a disk in the system. In order to accommodate this option, one needs to further partition the nodes in the system to $d$ different sets, and to maintain a copy of each data block on each of the partitions ($d$ copies to $d$ sets).[6] In this manner, each set can be left as the lone live set of disks and still provide a live copy of every data block. Note that for $d = 2$ this is automatically the case, but for

[6]This is as opposed to partitioning the disks into two uneven sets, one containing $1/d$ of all disks for the primary copies, and the other for the rest of the replicas.

$d \geq 3$ requires special care. Every time the system switches to low power mode, a different set is left alive, keeping all sets at the same rate of spin-downs. This method amounts to a longer mean time to disk failure (MTDF), by a factor of $(1 + \frac{1}{d-1})$ (when compared to always spinning-down the same set). For example, power cycling with $d = 2$ yield an MTDF that is twice as high, and if $d = 3$ then the MTDF grows by a factor of $1.5$. On the other hand, we note that the partition into $d$ sets has its price in performance of several tasks, such as parallel recovery which becomes more and more limited as $d$ grows.

This power cycling approach should be employed only when the number of spin-ups/spin-downs actually dominates the disks lifetime, and only when $d$ is small (because then both the power cycling makes a noticeable difference and the effect on performance is minor).