Low Power Mode in Cloud Storage Systems

Danny Harnik, Dalit Naor and Itai Segall
IBM Haifa Research Labs
Haifa, Israel
Email: {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 [18]), Hibernator [27] and Diverted Access (DIV [19]). Others, (e.g. [17] and Pergamum [21]) 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. [11], [24], [15], [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 [19] (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.

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

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

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, e.g. 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 [19].

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], [18], [27], [23] 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 [11] 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.

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, as shown in the full version of this paper (omitted due to lack of space). 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 [11], 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 [19] and write offloading [17].

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:

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

- 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 [15], 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) [18] concentrates popular data together. Hibernator [27] 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 [17] 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 [21] adds NVRAMs to each storage node to allow longer spin-down periods for disks in archival systems.

Exploiting redundancy: The Diverted Access (DIV) methodology [19] 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 [16], RIMAC [26] and PARAID [23]; 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. [12] 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' to denote an entity storing a subset of data items A data item refers to the basic replication unit of data.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 and N is the number of data items in the system. p refers to the fraction of the nodes that are turned off, and α_{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.

Note: The full version of this paper includes all proofs as well as sections that broaden the scope of the study to include also erasure codes and the case of heterogenous systems.

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 $q(p, M, d) = \frac{Mp(Mp-1)...(Mp-d+1)}{M(M-1)...(M-d+1)}$.
Note that $q(p, M, d) < p^d$ and tends to this number as M grows. The proof of Lemma 2.1 follows the probabilistic method. We note that the proof 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 (proof omitted):

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.

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, yet not essential: We next argue that finding the best coverage in a generic graph (or even finding a good approximation) is a computationally hard problem. This is seen by reduction to well known graph problems (specifically the problem of *vertex-cover in a hypergraph*). Therefore we shall resort to heuristics in order to find a good subset (see Section 3.1). Yet this inability of find the optimum is not crucial in our specific scenario. The reason is that the systems we deal with may be very dynamic, since 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). 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. The general framework is as follows:[2]

1) **Find a good designated subset:** (good in terms of the underlying cover problem). This step can be done using numerous heuristics (see Section 3.1).
2) **Construct the auxiliary nodes:** For a chosen designated subset the pool of extra nodes contains an additional copy of data items that are bound to be uncovered in low power mode.
3) **Low power mode:** To go into this mode one should 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

2. We leave specific details to be filled in for specific implementations and placement functions being deployed.

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.

4) **System wake up:** Apply write log.
5) **Full power mode:** The system runs in usual operation with the one exception that the auxiliary nodes are maintained in an online fashion.

Typically, the system would require a periodical refresh to the designated subset. This may be invoked due to noticeable data distribution change, addition/substraction of nodes or due to power cycling considerations.[3]

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.

3.1. How to Choose the 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. The efficiency of this heuristic should also be taken into consideration.

Choosing random subsets. This method is the simplest to implement and will likely provide a solution that is close to the expected value ($\sim p^d$ uncovered data items). One can improve by sampling a number of random subsets and taking the one with 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. Greedy algorithms have proved quite a successful heuristic in covering problems such as ours. In fact, Sümer [22] (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. Greedy algorithms will be feasible for most settings, and can be made more efficient by distributing their work across the nodes in the system.

3.2. Choosing the fraction p

We give an analysis of the best choice of the fraction p assuming that the uncovered set is indeed of size Np^d. This analysis can be modified accordingly for various placement functions. We model the power savings gained from the suggested low power mode. Assume that all nodes consume the same average power during operation mode, and do not consume power when turned off,[4] and discard the power consumed by the actual turning on and off of the disks Suppose that the low power mode accounts for an α_{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: (i) The spun down disks – this accounts for $\alpha_{low}p$ of the overall power consumption. In actual computations we should estimate α_{low} as a little smaller to compensate for various low order factors; and (ii) 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 for this choice.

α_{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 the two possible workload scenarios α_{low} discussed, and replication constant d we compute p_{max} that maximizes the power saving. We compute the ratio between the number of required auxiliary nodes M_{aux} and regular nodes M and estimated total power saving percentage and power saving percentage for each hour of low power mode.

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 amounts to a saving of approximately 0.3 of the power consumption during the low power mode (as opposed to 0.58 with $p_{max} = 0.547$, but with far less auxiliary nodes). In Table 2 we list some possible parameters using this approach.

3. 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).

4. 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.

$\frac{M_{aux}}{M}$	α_{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. For $\frac{M_{aux}}{M}$ fixed to values 0.01 and 0.05, we compute the fraction p of the power down set and estimate the total power saving percentage and saving percentage during low power mode.

3.3. Discussions

Auxiliary nodes vs. partial coverage: Instead of full coverage, it is possible 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. (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.

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 [19] and [17]. That is, write to all live replicas and additionally to a log (for writing during wake up). Such a system is not designed to handle a high volume of write operations in low power mode. 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.

4. Analysis of Specific Placement Functions

4.1. The Random Placement Function

This scheme refers to a random (or pseudorandom) allocation of locations for each of the replicas. Such a placement functions carries very good properties (such as load balancing, quick recovery from failure, etc...) and are therefore advocated by several systems. For example GFS [11], FARSITE [7] and RUSH/CRUSH [13], [25] 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.[5] Recall that this amounts to approximately a p^d fraction of the data items. The remaining question is how much does such a subset deviate from the expectancy. 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 is around the expected value $q(p, M, d)$.

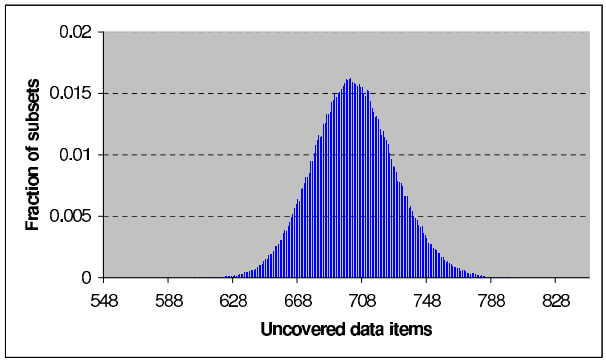

Figure 1. This is a system with random placement function, $M = 20$ nodes, replication $d = 3$ and $N = 6666$ data items. The fraction of the power down set is $p = 1/2$. The figure plots for each number of uncovered data items, what is the fraction of subsets of size Mp that yield this coverage. The calculated expectancy 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 exhibit that the "best of 10 random subsets" method achieves results that are marginally better than the average. The greedy tests achieved an improvement of a constant factor over the expected (the factor was larger for lower p).

4.2. Consistent Hashing

This placement scheme, introduced in [14] has proved useful in Peer-to-Peer networks (the Chord system [20]) and is the underlying placement function for a number of distributed storage systems (e.g., [9], [15], [10], [6]). In a nutshell, consistent hashing assigns to each node a segment of the range of data items (all data items are mapped onto a cyclic range of, say, 2^{128} values). Each nodes then stores the data items in its corresponding segment as well as the next $d - 1$ consecutive segments. This scheme has nice and desired properties, but is suspect to problems regarding load balancing (some nodes are bound to be very light) and

5. 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.

failure from recovery (limited in its ability to parallelize). The standard technique [14] to overcome this is to employ assign to each node a collection of **virtual nodes** rather than just one. When the number of virtual nodes per actual node is on the order of $\log M$, then the load balancing becomes acceptable.[6] For further details see [20].

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 and still guarantee that one copy is always active.

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 randomly placed virtual nodes. 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 focus on systems where the number of virtual nodes is approximately $\log M$, the amount advocated by [14]. 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 nodes at all).

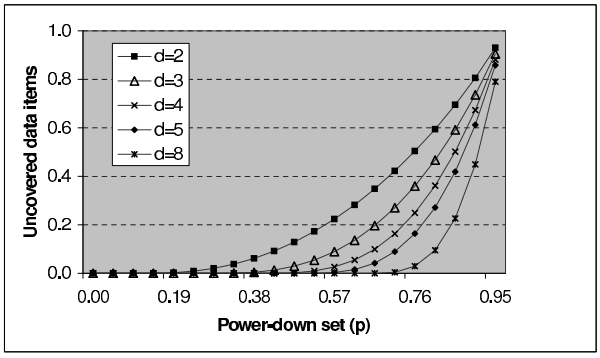

Figure 2. We evaluate the success of the greedy algorithm in a consistent hashing system as a function of the fraction of the power-down subset p. We test this on a system with $M = 4096$ nodes $\log M = 12$ virtual nodes.

We next ask what is the effect of using auxiliary nodes in the system. In Table 3 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.

6. Virtual nodes are also handy for handling nodes of different capacities (by varying the number of virtual nodes per node).

$\frac{M_{aux}}{M}$	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 3. We provide the fraction p of the powered down disks and power saving during low power (with $\alpha_{low} = 1/3$) that can be covered without, with 1% or with 5% of auxiliary nodes in a consistent hashing system. Tests were run with $M = 4096$ nodes and $\log M$ virtual nodes.

We conclude that the consistent hashing placement is better suited for low power mode than the random placement function (an artifact of the strong correlations produced by this scheme). Also, the greedy algorithm proves very useful in this scheme.

5. On Augmenting the Placement Function for Low Power Mode

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 to fit in [19] with the Diverted-access (DIV) idea, suggesting to segregate the primary copies of the data items on specific set of disks and so the rest of the disks may be powered down without damaging full coverage. This achieves an optimal full power mode, leaving exactly one copy of each data item alive. Designing a placement function that has a multitude of desired properties, rather than just a good power saving design, may prove quite challenging since accommodating an optimal low power mode may compromise some of the properties of the system (e.g. location awareness or the the ability for parallel recovery from failure).

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". Such a structure benefits from the

ability to spin-down partial nodes (i.e., part of the disks in every node are spun down). We suggest an augmentation of the placement to allow an optimal low power mode in such systems. All one needs to assure is that each data item has exactly one of its replicas placed on a disk that is designated to remain powered up (the choice of which replica to keep alive is done randomly). The main benefit of this method is that the placement at the upper level remains *unmodified*. Even more so, the distribution within each single node (at the lower level) remains the same and the only change is the addition of inter-node correlations. We view this as a real option for power saving in many common systems that is beneficial also for low replication such as $d = 2$. When taken to its extreme, one can think a network of numerous data-centers where one can power down half of each data center and still maintain full coverage.

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

[12] 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.

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

[14] 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.

[15] 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.

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

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

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

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

[20] 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.

[21] 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.

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

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.