Abstract—This paper deals with the problem of scheduling requests on disks for minimizing energy consumption. We first analyze several versions of the energy-aware disk scheduling problem based on assumptions on the arrival pattern of the requests. We show that the corresponding optimization problems are NP-complete by reduction to the set cover or the independent set problem. Then both optimal and heuristic scheduling algorithms are proposed to maximize the energy saving of a storage system. Our evaluation results using two realistic traces show that our approach significantly reduces energy consumption up to 55% and achieves fewer disk spin-up/down operations and shorter request response time as compared to other approaches.

I. INTRODUCTION

Large scale datacenters need to maintain thousands of rotating disks for fast online access of stored datasets. As disks are getting cheaper in terms of dollars per gigabyte, the prediction is that the energy costs for operating and cooling these rotating disks will eventually outstrip the cost of the disks and the associated hardware needed to control them. Currently it is estimated that disk storage systems consume about 35 percent of the total power used in data centers [15]. This percentage of power consumption by disk storage systems will only continue to increase, as data intensive applications demand fast and reliable access to on-line data resources. This in turn requires the deployment of power hungry faster (high RPM) and larger capacity disks.

Several energy saving techniques for disk based storage systems have been introduced in the literature [21], [32], [19], [10], [27], [26]. Most of these techniques revolve around the idea of spinning down the disks from their usual high energy mode (idle mode) into a lower energy mode (standby mode) after they experience a period of inactivity whose length exceeds a certain break-even time (also called idleness threshold). The reason for this is that typical disks consume about one tenth of the power in standby mode as compared with their power consumption in idle mode (See Fig 5). It is well known [17] that the optimal deterministic power management policy is to set idleness threshold to $T_B = \frac{E_{up} + E_{down}}{P_I}$ seconds where $E_{up}$ and $E_{down}$ are the amount of energy (in joules) needed to spin the disk up (from standby to idle mode) and down (from idle to standby mode) and $P_I$ is the power (in watts) required to keep the disk spinning in idle mode. We call this power management policy 2CPM (2-competitive power management scheme) as it was proved that it results in energy consumption not exceeding twice that of the best possible algorithm that has a priori knowledge of all request arrival times. There are several problems associated with this spin-down technique:

- **Energy and Response Time Penalty**-Disks can only service requests while they are in idle mode, in case a request arrives when the disk is in standby mode there is a response time penalty of several seconds (5-15 seconds) before the request can be serviced. In addition, as mentioned above, the energy cost of $E_{up} + E_{down}$ joules may in some cases exceed the energy saved by transitioning the disk to standby mode.
- **Expected length of inactivity periods**-Under many typical workloads found in scientific and other applications, disks do not experience long enough periods of inactivity (longer than the idleness threshold ) thus limiting the opportunities to save energy by transitioning to standby mode.
- **Number of spin-up/down operations**-For typical disks, an excessive number of such operations may shorten the lifetime of the disk and must also be considered as a factor in the energy saving strategy. As an evidence of the importance of this factor it is included as an attribute of S.M.A.R.T., a popular monitoring system for predicting hard disk failures.

The research literature attempts to solve the above problems using several techniques. The idea is to re-shape the workload so that I/O requests are directed to a small subset of the disks allowing the other disks to enjoy relatively long periods of inactivity. Several of these techniques are listed below.

- **Data Placement** [26], [27]: These techniques use intelligent data placement on disks where the most popular files are packed into the fewest possible disks subject to response time constraints. This placement is based on some knowledge of data access patterns. A second approach tries to achieve the same goal by using dynamic data migration based on observed workloads. This approach does not require an a priori knowledge of the access patterns but results in a larger time-scale to adapt.
- **Write off-loading** [23]: These techniques are directed towards minimizing energy consumed due to write requests. Newly written data is diverted to disks which are currently spinning (anywhere in the data center) thus enabling disks in standby mode to remain in their low-energy mode for longer periods.

---

`http://www.linuxjournal.com/article/6983`
techniques rely on either reconfiguring or migrating data. The effective techniques among existing approaches, these storage system could become unreliable or unavailable because of data consistency and synchronization issues. Expensive data transfer overhead also occurs in terms of network bandwidth usage and request response time. Finally, new data transfer protocols and control systems must be deployed to facilitate data movement. On the other hand, although data reconfiguration does not introduce performance degradation, it forces data to be organized in a specific structure or location (e.g. [28] that segregates the original data from the redundant data placing them on different subsets of disks; [22]. Therefore, storage systems cannot manage data without cooperating with these techniques, resulting in the introduction of more limitations and complexity.

While data placement and replication are the more practical and effective techniques among existing approaches, these techniques rely on either reconfiguring or migrating data. The interference of data placement could cause several performance and implementation concerns. During data migration, a data placement and replication are the more practical and effective techniques among existing approaches, these techniques rely on either reconfiguring or migrating data. The interference of data placement could cause several performance and implementation concerns. During data migration, a storage system could become unreliable or unavailable because of data consistency and synchronization issues. Expensive data transfer overhead also occurs in terms of network bandwidth usage and request response time. Finally, new data transfer protocols and control systems must be deployed to facilitate data movement. On the other hand, although data reconfiguration does not introduce performance degradation, it forces data to be organized in a specific structure or location (e.g. [28] that segregates the original data from the redundant data placing them on different subsets of disks; [22]. Therefore, storage systems cannot manage data without cooperating with these techniques, resulting in the introduction of more limitations and complexity.

In this paper, we argue that intelligent scheduling can be used to save energy using a given data placement for the replicas. We develop a scheduling algorithm that takes a stream of requests as input, and determines the disk location for serving each request, to minimize energy consumption. To the best of our knowledge, it is the first time a scheduling algorithm is used in the context of energy saving for storage systems.

Our approach has several advantages over previous solutions. First, it has no interference with any data placement scheme. Second, it is designed to work with any power management scheme in which a disk simply gets spun down after experiencing a fixed idleness threshold. Such schemes are used commercially in systems such as Copan-400 by SGI [11] and AutoMAID by NexSan [2]. Last but not least, it is complementary and compatible with many other power-saving techniques [22], [20], [10] that could use scheduling to further concentrate requests on fewer active disks. Our main contributions are:

- We propose a novel energy-aware scheduling approach to reduce energy consumption of storage systems without interfering with data placement or introducing additional performance and deployment overhead.

- We formulate energy-aware scheduling as an optimization problem and propose both optimal and heuristic algorithms for offline, batch and online cases.

- We analyze the complexity of the energy-aware scheduling problem and show several variations of it are NP-complete using reductions from the weighted set cover problem in the batch case and the maximum independent set problem in the offline case. In both cases we solve the scheduling problem sub-optimally using efficient polynomial approximations.

- We provide extensive evaluations with realistic traces (Cello [6], Financial1 [30]), disk model and power manager. The experiments show a reduction in energy consumption up to 55%, while minimizing the number of disk spin-up/down operations and request response time.

The rest of the paper is organized as follows. Related work on the set cover and independent set problem is given in Section II. Section III gives an overview of the energy-aware scheduling problem. Our scheduling algorithms for the three scheduling models are proposed in Section IV. The experimental setup and results are presented in Section V, and Section VI. Finally, conclusions and future work are presented in Section VII. Detailed correctness proofs of our algorithms are given in the Appendix.

II. RELATED WORK ON ALGORITHMS

Our algorithms rely on approximations to two well known NP-complete problems. Some background about these problems is given below. Weighted Set Cover (WSC) and Maximum Weighted Independent Set (MWIS) are two classic NP-complete problems. In the WSC problem, \( WSC(S, U) \), we are given a set \( U \) of \( n \) elements and a collection \( S \) of subsets of \( U \) each associated with a positive weight. A set cover of \( U \) is a collection of subsets, \( S' \), of \( S \) where the union of the subsets in \( S' \) is \( U \). The weight of a cover \( S' \) is the sum of the weights of the subsets in it. The WSC problem is to find a set cover with minimum weight. The greedy set cover algorithm works as follows. It first selects the most cost-effective subset, i.e., the subset whose cost per element is smallest, and adds that subset to the solution while removing the covered elements and the subset from further consideration. This process is repeated on the remaining subsets and elements until all elements are covered. This simple heuristic was proven to produce a set cover with cost at most a factor of \( H_n = O(\log n) \) is the \( n^{th} \) harmonic number equal to \( 1 + \frac{1}{2} + \cdots + \frac{1}{n} \), and \( n \) is the cardinality of \( U \). The input to the MWIS problem is a graph \( G(V, E) \) with weighted vertices. An independent set \( I \) of \( G \) is a subset of \( V \) such that there is no edge in \( E \) between any pair of elements in \( I \). The MWIS problem is to find an independent set of maximum weight. One of the greedy algorithms for the non-weighted MWIS problem that has been investigated in the literature selects a vertex of minimum degree, removes it and its neighbors from the graph, and iterates this process on the remaining graph.
does not admit an $O(\log n)$ factor approximation algorithm for general graphs [4], [16]. Exact algorithms for MWIS have been developed and gradually improved [18], [13]. But, approximation algorithms with additional constraints and assumptions have been more commonly discussed [8], [14]. Recently, branch-and-bound approximation methods [31] have also been proposed for general graphs. In our experiments, we applied the greedy set cover and the GWMIN algorithms for our scheduling algorithms.

### III. ENERGY-AWARE SCHEDULING

**A. Scheduling Architecture and Models**

Figure 1 represents a typical architecture for a modern storage system with three main components. Data placement manager determines the number of data replicas and data location based on a replication scheme. Disk power manager decides when to spin down disks by implementing a power management policy. Finally, Metadata server keeps all the necessary information for system operations, such as the mapping between file blocks and disk location, the power status and utilization statistics of disks, etc.

In general, users submit requests to perform disk I/O on data, and each request is associated with an arrival time and a single data block. After receiving requests, the scheduler dispatches requests to disks according to a scheduling algorithm. The release time of a request is the time when it is dispatched to the disk for processing by a scheduler. Our goal is to develop energy-aware scheduling algorithms that can minimize the energy consumption for a given workload and storage system configuration. Specifically, we discuss our algorithms in the following three scheduling models:

**Offline** assumes a scheduler has *a-priori* knowledge of the arrival times of requests before making its scheduling decisions. It represents an ideal scheduling scenario and used mainly as a benchmark for theoretical analysis.

**Batch** queues requests and dispatches them all together to disks periodically at a scheduling interval. This model is commonly used for high-performance and scientific computing [24].

**Online** represents a basic model where a scheduler immediately dispatches requests to disks upon their arrival.

**B. Assumptions**

Below is a summary of the underlying assumptions used in the rest of the paper:

- **Data availability:** Fault tolerance issues such as recovery from disk failures and guaranteeing data availability during such failures is handled by the storage system replication scheme and is not affected by our scheduling algorithm.
- **Data placement:** Data placement information is given and available to the scheduler through a metadata server. Our technique is designed to handle arbitrary placement implemented by any replication scheme. The scheduling algorithm does not pre-determine or change data placement.
- **Power management:** Power management is similar to the 2CPM scheme in which a disk is spun down after experiencing a fixed predetermined idleness threshold. This scheme is the most common one used in commercial systems [11], [2].
- **Request arrival time on disk:** Our technique schedules the disk location for servicing requests and dispatches requests in the same order as their arrival time. In the offline and online model, the arrival time of a request is the same as its release time. In the batch model, requests are queued and are dispatched together at the end of a scheduling interval. The length of the scheduling interval is a parameter controlled by system administrators and not the scheduler.
- **Write requests:** In our scheduling problem, we do not distinguish between read and write requests because by using the write techniques discussed in the next subsection they both only need to perform an I/O on one of the disks holding their data replicas.

In the theoretical analysis calculations of the offline model, we make the following two additional assumptions to guarantee the starting and finishing disk I/O time of requests are the same as their release times:

- **Disk spin-up delay:** Equipped with arrival time (and therefore also release time) knowledge, we assume that all requests can start their disk I/O at their release time without suffering disk spin-up delays. This is done by spinning up disks in advance or keeping disks idle before requests arrive according to the scheduling decisions.
- **Disk I/O time:** Since each request is associated with only one file block (normally 512B), the disk I/O time of a request is on the scale of milliseconds and is very small when compared with the time scale of a disk power management operation (e.g. spin-up/down) which is measured in seconds. For that reason, in our analysis we assume the request I/O time is negligible. However in our experimental evaluations we do use the Disksim [12] simulator to take the disk I/O time into account.

**C. Synchronization for Write Requests**

Here, we discuss how write requests are handled by the system as well as in our scheduling algorithm. Write requests either attempt to update an existing data block or create a new one. Writes for new data can be scheduled on any of the active disks, and they do not involve scheduling or cause
requests. On the other hand, writes that update data can be translated into multiple requests, one for each replica of the data. However, such an approach potentially requires multiple disks to be spun up to serve a write request and limits the opportunity for saving energy. Therefore, we suggest to adopt a write offloading technique similar to the one that has been used in [23], [28]. The basic idea is to handle writes by updating the data on only one of its locations. Then the updated data can be propagated to the rest of its locations either periodically, on demand (when a subsequent read or update request for the same block arrives), or proactively when the involved disks become active. The detailed implementation of such write techniques are beyond the scope of this paper and are explained in detail in [23]. By using these techniques, both writes and reads only need to be scheduled on one of their data locations. Therefore, in analyzing our scheduling algorithm, we do not distinguish between read and write requests.

D. Examples

We will now illustrate the advantages and challenges of designing an energy-aware schedule using several examples. For simplicity, we use a dummy disk power configuration which has no time delay or energy cost to spin up or down a disk, and the disks consume exactly 1 unit of energy per second at the active or idle mode and 0 units of energy in standby mode. In the example, we also let the breakeven time (idleness threshold) be 5 seconds, and the initial state of all disks to be standby. Notice, the active state of a disk is not indicated in the following illustration figures because its duration is negligible by our previous assumptions.

1) Batch scheduling example: The first example shown in Figure 2 simulates a batch scheduling problem where all requests access disks concurrently because of the batch queuing delay. In the example, there are six requests, $r_1 \cdots r_6$, and their requested data blocks are $b_1 \cdots b_6$, respectively. The data placement is given and known to the scheduler as shown in the figure where disk $d_1$ has data $b_1, b_2, b_3, b_5$, disk $d_2$ has data $b_2, b_3$, disk $d_3$ has data $b_4, b_6$ and disk $d_4$ has data $b_3, b_4, b_5, b_6$.

Figure 2(a) shows one possible schedule $A$, which assigns requests $r_1, r_5$ to disk $d_1$, requests $r_2, r_3$ to disk $d_2$ and requests $r_4, r_6$ to disk $d_3$. As shown by the disk status at the right side of Figure 2(a), disks $d_1, d_2, d_3$ are spun up and become active at time 0, then they remain idle for a breakeven time 5 seconds before spinning down. Comparing to the always-on disk power scheme, schedule $A$ consumes less energy with 15(=3*5) because it keeps disk $d_4$ standby.

However, as shown in Figure 2(b), there is another schedule $B$ using even less energy with 10(=2*5) by scheduling requests $r_1, r_2, r_3, r_5$ to disk $d_1$ and requests $r_4, r_6$ to disk $d_3$. In fact, schedule $B$ is an optimal schedule for this example, because it uses the minimum number of disks to service all requests as proven in Theorem 2. As shown, schedule $B$ represents a 33% reduction in energy consumption as compared with schedule $A$.

2) Offline scheduling example: We next extend the above example to the offline model by associating each request with an arrival (release) time. As shown in Figure 3, the time for requests $r_1 \cdots r_6$ are 0, 1, 3, 5, 12 and 13, respectively.

Figure 3(a) re-examines the results of schedule $B$ in the offline model. Schedule $B$ still uses the minimum number of disks to service requests, but the energy consumption of $d_1$ and $d_3$ now becomes 13 and 10. As shown by the disk status at the bottom of Figure 3(a), disk $d_1$ is spun up at time 0 to service $r_1$, and the disk is not spun down until time 8 when it experiences an idle period of breakeven time (5s). At time 12, $d_1$ is spun up again to service $r_5$ and spun down after it reaches the breakeven time at time 17. Similarly, the energy consumes of disk $d_3$ is 10 because it is spun up at time 5 and 13 for $r_2$ and $r_6$, respectively. Therefore, the energy consumption of schedule $B$ is 23.

Although schedule $B$ consumes much less energy than the always-on configuration, 72(=18*4), it is no longer an optimal schedule in this example. Figure 3(b) shows an optimal schedule $C$, which uses three disks but has the minimum energy consumption. Schedule $C$ assigns requests $r_1 \cdots r_3$ to disk $d_1$, request $r_4$ to disk $d_2$ and requests $r_5, r_6$ to disk $d_3$. As a result, $d_1$ is idle between time 0 to 8, $d_3$ is idle between time
Consists of a set of parameters. While let scheduled requests, but also on their arrival time. Scheduling algorithms are not only dependent on the location of data for scheduled requests, but also on their arrival time. The notation of the scheduling problem is summarized in Table I.

### A. Offline Scheduling

Here, we formulate the offline scheduling problem as an optimization problem, and we propose a scheduling algorithm and discuss its correctness and complexity. The notation of the variables in our formulation is summarized in Table I.

#### 1) Problem formulation: An energy-aware scheduling problem ES is described by an input of \((R, D, L, P)\), where \(R = \{r_j\}\) is a request stream, \(D\) is a list of disks, \(L\) is the data placement information, and \(P = \{T_{up}, T_{down}, E_{up}, E_{down}, T_B, P_l\}\) is the configuration of 2CPM scheme. Each request \(r_i\) is associated with a data block and a release time \(t_i\) (i.e. the time a disk receives the request), requests are sorted in \(R\) by their release times in increasing order. The power configuration \(P\) consists of a set of parameters. \(T_{up}, T_{down}, E_{up}\) and \(E_{down}\) are the time and energy to spin up or spin down a disk, while \(T_B, P_l\) are the disk break even time and idle power. Let \(S_{ES}\) be the set of all feasible schedules for a scheduling problem ES. Then, our goal is to find an optimal schedule \(S_{ES}^* \in S_{ES}\), such that the energy consumption of \(S_{ES}^*\) is the minimum among the set of all feasible schedules, i.e.,

\[
S_{ES}^* = \min(S_{ES}^* | \forall S_{ES} \in S_{ES}).
\]

Specifically, we formulate our scheduling problem by analyzing the energy saving of a schedule instead of the energy consumption. Therefore, in the following analysis, we use \(X(\cdot)\) as a function of energy saving, such that \(X(S_{ES}^*)\) denotes the energy saving of a schedule \(S_{ES}^*\) for scheduling problem ES, and \(X(S_{ES}^*, r_i)\) denotes the energy saving of any request \(r_i \in R\) under the scheduling results of \(S_{ES}^*\). For simplicity, we assume the initial status of disks is standby and the standby power is 0, but these assumptions are not necessary for the correctness of our problem formulation.

To compute \(X(S_{ES}^*)\) and \(X(S_{ES}^*, r_i)\), we first define the energy consumption of a request as the amount of energy consumed by its scheduled disk from the time of servicing the request until the next request arrives on the disk. For example, in Figure 3(b), the energy consumption of \(r_1\) is 1, because disk \(d_1\) is idle from time 1 (arrival time of \(r_1\)) to time 2 (arrival time of \(r_2\)), and the disk idle power is 1 unit per second in the power configuration. Intuitively, the energy consumption increases proportionally to the inter-arrival time to the successor request. Therefore, we also know that the maximum energy consumption of any request is \((E_{up} + E_{down} + T_B * P_l)\), which occurs when the disk has been spun down before the successor request arrives.

We define the energy saving of a request \(r_i\) in schedule \(S_{ES}^*\) as \(X(S_{ES}^*, r_i)\) = the maximum energy consumption of \(r_i\) - the energy consumption of \(r_i\) under schedule \(S_{ES}^*\). For example, in Figure 3(b), the energy saving of \(r_1\) is \(4(=5-1)\), because its maximum energy consumption is 5 \((E_{up} = E_{down} = 0, P_l = 1, T_B = 5)\), and its energy consumption is only 1.

By the above definition of request energy saving, it is trivial to show that the energy saving of a schedule is equal to the aggregated energy savings of all its requests because a schedule iteratively assigns requests to disks. Thus, we have

\[
X(S_{ES}^*) = \sum_{r_i \in R} X(S_{ES}^*, r_i)
\]

Furthermore, the amount of energy saving of a request is determined by the time of its successor request on the disk because the energy consumption of a request is computed between the time of the request and its successor request. Therefore, we re-formulate the value of \(X(S_{ES}^*, r_i)\) by an energy saving amount \(X(i, j, k)\), such that

\[
X(S_{ES}^*, r_i) = \begin{cases} X(i, j, k), & \text{if } S_{ES}^* \text{ schedules } r_i \text{ on disk } d_k \text{ and the successor request of } r_i \text{ on } d_k \text{ is } r_j, \\ 0, & \text{otherwise} \end{cases}
\]

Intuitively, we can only save energy from a request \(r_i\) if its successor request \(r_j\) arrives before disk \(d_k\) is spun down. In addition, more energy is saved from \(r_i\) when the arrival time of \(r_i\) and its successor \(r_j\) get closer, because the amount of disk idle time between the two requests is reduced. As described in our technical report [9], the exact value of \(X(i, j, k)\) can be computed as follows, and it is always a positive value.

\[
X(i, j, k) = \begin{cases} E_{up} + E_{down} + (T_B - (t_j - t_i)) * P_l, & \text{if } 0 \leq t_j - t_i < T_B + T_{up} + T_{down} \\ 0, & \text{otherwise} \end{cases}
\]

and

\[
X(i, j, k) \in ES, \text{iff } d_k \text{ has the data of } r_i, r_j \text{ and } t_i < t_j.
\]

Let \(U\) be the set of all \(X(i, j, k), \forall i, j, k\), in a given scheduling problem ES. Based on the above equations, we find that the energy saving of any schedule, \(X(S_{ES}^*)\), can be computed by summing up the values of \(S\) where \(S \subseteq U\), and each \(X(i, j, k) \in S\) is equal to \(X(S_{ES}^*, r_i)\) (i.e. the energy saving of \(r_i\) in the schedule).

However, to insure \(S\) is from a valid schedule, any pair of \(X(i, j, k)\) and \(X(i', j', k')\) in \(S\) must satisfy the following two scheduling constraints. The first constraint is that a request can only be scheduled at one disk. Hence, if \(X(i, j, k)\) and \(X(i', j', k')\) are both in \(S\), they share a common request (i.e. \(\{i, j\} \cap \{i', j'\} \neq \emptyset\)), then their associated disks \(k \quad k'\) must be the same. The second constraint is that a request can only one predecessor and successor on its scheduled disk. Hence, any pair of \(X(i, j, k)\) and \(X(i', j', k')\) in \(S\) cannot have the same value for \(i\) and/or \(j\).

Therefore, we formulate the energy saving of an optimal schedule, \(X(S_{ES}^*)\), as an optimization problem in Table II, such that the value of \(S\) corresponding to an optimal schedule \(S_{ES}^*\) is maximized while satisfying the scheduling constraints.
Given a scheduling problem \( ES(R, D, L, P) \) and a set of energy saving \( U \):
\[
U = \{ X(i, j, k) \mid X(i, j, k) \in ES \}
\]
Find a subset of energy saving \( S \subseteq U \) such that:
\[
X(S) = \sum_{X(i, j, k) \in S} X(i, j, k)
\]
is maximized subject to the following constraints:
- For each pair of \( X(i, j, k) \in S \) and \( X(i', j', k') \in S \):
  - \( i \neq i', j \neq j' \)
  - \( k = k' \) if \( \{i, j\} \cap \{i', j'\} \neq \emptyset \)

2) Algorithm: Based on the above problem formulation, we propose MWIS offline scheduling algorithm by reducing the problem into a maximum weighted independent set problem.

Given a scheduling problem \( ES(R, D, L, P) \), we construct a graph \( G = (V, E) \) in two steps. Step 1 adds a node to \( V \) for each \( X(i, j, k) \in ES \) with non-zero value according to Eq. 3 and Eq. 4. Then Step 2 adds an edge between any pair of nodes \( X(i, j, k) \) and \( X(i', j', k') \) that do not satisfy the constraints in our problem formulation. Once the graph is constructed, Step 3 applies a maximum weighted independent set algorithm to select the set of \( S \). Finally, Step 4 derives an optimal schedule \( S'_{ES} \) from \( S \) by scheduling requests \( r_i \) and \( r_j \) to disk \( d_k \) for each \( X(i, j, k) \in S \). Since Step 1 ignores any \( X(i, j, k) \) with zero value, a request may not be associated with any \( X(i, j, k) \in S \) when the request cannot save energy at any disk location. Thus, the request is allowed to be scheduled on any of its associated data locations. Figure 4 illustrates our algorithm by computing the optimal solution schedule \( C \) in Figure 3(b).

3) Complexity:

**Theorem 1:** An energy-aware offline scheduling problem is equivalent to a weighted independent set problem.

**Proof outline:** Our offline scheduling algorithm already shows that an offline scheduling problem can be reduced to a weighted independent set problem. In addition, Theorem 3 in Appendix proves a weighted independent set problem can be reduced to an offline scheduling problem as well. Therefore, the two problems are equivalent. A more detailed proof is given in our technical report [9].
As proven by Theorem 1, the offline scheduling problem is NP-complete and equivalent to the weighted independent set problem. Hence, here we briefly discuss the complexity of our scheduling algorithm based on a greedy weighted independent set algorithm, GWMIN [29], which is also used for our evaluation in Section VI. As proven in [29], the complexity of the greedy GWMIN algorithm is polynomial in the size of the graph $O(|V| \times |E|)$, where $|V|$ and $|E|$ are the number of nodes and edges in the graph respectively. But, according to our reduction algorithm shown in Step 1 and Step 2, in the graph the number of nodes is $O(|R|^2 + C)$ and the number of edges is $O(|R|^2 + C)^2)$ where $|R|$ is the number of requests, and $C$ is the number of replicas per data. As a result, the worst case time complexity of our MWIS scheduling algorithm is relatively expensive. However, its worst case complexity should not be a major concern because of the following reasons:

- The algorithm will not be run frequently as it was developed mainly for the purpose of theoretical analysis.
- The offline algorithm does not need to provide real time answers so it can be pre-computed on machines with substantially more computational power and more generous time constraints.
- We also note that the expected time complexity of the algorithm should be much smaller than its worst case for several reasons. First, the number of nodes in the graph decreases with an increase in the number of disks, because there is less chance for the data from two requests to be located on the same disk. Second, the number of edges decreases as the length of the breakeven time is reduced, because edges are created by a schedule constraint between two requests, and the constraint can not occur if the two requests are separated by more than the breakeven time.

As shown by our experimental setup, we were able to compute the offline scheduling algorithm for 70,000 requests with breakeven time 54 sec, while disks used in previous techniques [27] have breakeven time less than 20 sec.

### B. Batch Scheduling using Weighted Set Cover

As described in Section III, under the batch scheduling model, requests are queued and dispatched simultaneously at the end of each scheduling interval. Thus, the optimality and problem space of a batch schedule should also be considered with respect to the set of requests queued at each scheduling interval. As a result, an energy-aware batch scheduling problem is equivalent to a weighted set cover problem, and it has better known approximations.

**Theorem 2:** An energy-aware batch scheduling problem is equivalent to a weighted set cover problem.

*Proof outline:* We can find the equivalence between the two problems by mapping each disk to a set and each data to an element. A more detailed proof is given in our technical report [9].

We propose a weighted set cover (WSC) batch scheduling algorithm as follows. For a given batch scheduling problem, we construct its corresponding weighted set cover problem $WSC(S, N)$ where the sets $S$ are disks and the elements of $N$ are the requests in the queue. A disk represents a subset of the requests that can be satisfied by data blocks on that disk. The weight of a set is determined by the amount of energy consumption of a disk when requests are scheduled on the disk. Specifically, in Theorem 2, we show the weight (energy consumption) of a disk can be computed according to its current status as the following:

$$E(d_k) = \begin{cases} 0, & \text{if the disk is active} \\ E_{up/down} + T_B \cdot P_1, & \text{if the disk is standby} \\ (T_{now} - T_{last}) \cdot P_t, & \text{if the disk is idle} \end{cases}$$

where $T_{now}$ is the current time, and $T_{last}$ is the time that $d_k$ receives its previous request.

Once the graph is constructed, a weighted set cover algorithm is applied to select a set of disks with the minimum weight to cover all requests. Accordingly, we achieve the minimum energy cost by scheduling requests on these disks.

In the experimental evaluation, we implemented our WSC scheduling algorithm by the greedy weighted set cover algorithm. As mentioned in Section II, greedy algorithm is essentially the best-possible polynomial time approximation algorithm for the problem with a guaranteed approximation ratio, and its time complexity grows polynomially with the size of our problem input $O(|D| \times |R|)$ (i.e. the product of number of disks and requests). Since a storage system has at most thousands of disks and the number of requests in the queue is limited, it is practical to implement and deploy such a scheduling algorithm in real systems.

### C. Cost Function Online Scheduling

Finally, an online scheduling problem can be considered as a special case of the batch scheduling problem when only one request needs to be scheduled at a time. Thus, we could again apply our WSC scheduling algorithm to the online scheduling problem, so that the scheduler simply assigns each request to one of its data locations with the minimum energy consumption, $E(d_k)$.

But, in this subsection, we further extend the WSC scheduling algorithm by using a composition cost function, $C(d_k)$, to optimize for both energy cost, $E(d_k)$, and performance cost, $P(d_k)$, as follows

$$C(d_k) = E(d_k) \cdot \frac{\alpha}{\beta} + P(d_k) \cdot (1 - \alpha)$$

The performance cost could be defined as any metric measured by the system. In this paper, we are interested in request response time, the main performance penalty incurred by our technique. By concentrating requests on fewer disks to keep the other disks idle, requests could suffer longer response time delays because more requests are queued on disks waiting to be processed. Since response time delay has strong correlation with the number of requests waiting on a disk queue, and the information of disk queue length is relatively easy to be measured and collected, it is selected to represent the performance cost $P(d_k)$ as follows:

$$P(d_k) = \text{number of requests queued on disk } d_k.$$
Parameters $\alpha, \beta$ are defined to adjust the cost, $\alpha$ determines the cost ratio between energy and performance. In other words, the cost function, $C(d_k)$, only considers energy cost when $\alpha = 1$, and it only considers performance cost when $\alpha = 0$. On the other hand, $\beta$ determines the unit factor between energy cost and performance cost. Since $E(d_k)$ and $P(d_k)$ could be computed based on different cost metrics, $\beta$ is used to scale the cost between them.

From Eq. 5 and Eq. 7, we clearly see the trade-off between energy and response time. For instance a standby disk with no load has high energy cost but low performance cost, while an active disk has low energy cost but high performance cost. Also, for saving energy, a scheduler actually prefers a disk which is in the process of being spun-up rather than a disk in idle mode, because it could satisfy more requests together and reduce disk idle time, even though requests could suffer some disk spin-up delay. More sophisticated cost functions could be defined and computed if more information and techniques are available. For example, a prediction technique could be used to estimate the access probability of a disk and assign lower cost to a more frequently used disk. But in this paper, we propose a basic solution as a first step to demonstrate our cost function strategy. A more detailed evaluation of the trade-off between energy and response time is shown in Section VI-E.

V. EXPERIMENTAL SETUP

For our experiments, we built an energy-aware storage system framework using OMNeT++ [25]. In addition, we incorporated our framework with a disk simulator Disksim [12] by utilizing its interfaces to synchronize the event time between the two simulators. The disk model simulated in the experiments is Seagate Cheetah 15.5 enterprise disks [7]. For this disk model, however, some power information, such as standby power, is missing in the associated documents. Therefore, we alternatively used power parameters from Seagate Barracuda specification [3] as shown in Figure 5.

A. Workload Trace

Our approach was evaluated with two real disk I/O traces, namely Cello [6] and Financial1 [30]. Cello is collected from a timesharing system used by a group of researchers at Hewlett-Packard Lab to do simulation, compilation, editing, and mail. Financial1 is collected from an OLTP application running at a financial institute and made available in [30]. Both traces record low-level (block) disk I/O activities over an extended period of time, the information includes request release time (i.e. the time a disk receives a request), logical block address, block size, etc. In the experiments, we consider each unique combination of disk id and block address as a data in the storage system. For both traces, we used 70,000 requests accessing over 30,000 data blocks. The reason we limited the number of requests is due to the computation complexity of our offline scheduling algorithm. However, increasing the number of requests has a limited impact on the results as we observed from the online and batch scheduling algorithms.

![Fig. 5: 2CPM configuration for experimental evaluations.](image)

B. Data Placement

Our experiments used a storage system with 180 disks. Although our approach makes no assumption about data placement, our evaluation was based on a particular one with a higher degree of data skewness among disks. Specifically, we use replication factor to indicate the number of replications for each data block. The first data location is referred to as the original data location, and the rest of the data locations are referred to as the replica data locations. In the experiments, we assume that the original data locations were randomly selected among disks by a Zipf distribution, such that the probability of choosing a disk ($p$) and its rank $r$ is $p = c/r^z$ where $c$ is a constant and $z$ is often close to 1. This skewness of data locality was observed from the Cello trace, as well as from web servers [5]. On the other hand, we assume that replica data locations were evenly distributed among disks, because it is a common data placement scheme to ensure fault tolerance in a storage system. The results of a range of other data placement configurations are also discussed in Section VI-D.

C. Scheduling Algorithms

In the experiments, we compared our proposed energy-aware scheduling algorithms to two other commonly used scheduling algorithms without energy consideration (i.e., Static and Random). We didn’t compare our results with more intelligent scheduling algorithms because to the best of our knowledge there are currently no existing energy-aware scheduling algorithm proposed or developed. The results of our static scheduling under highly skewed data placement configuration can also be used to represent the energy saving techniques that rely on data placement reconfiguration, such as [28], [27], [26].

**Random**: uniformly sends requests to one of data locations.

**Static**: always sends requests to original data locations.

**Energy-aware Heuristic**: the online algorithm proposed in Section IV-C. The cost function parameters are configured as $\alpha = 0.2$ and $\beta = 100$ to reach the balance of energy and response time as observed in Section VI-E.

**Energy-aware WSC**: the batch algorithm proposed in Section IV-B with 0.1s scheduling interval. The disk weights are computed by the same cost function of Heuristic, and the reduced MWIS problem is solved the greedy set cover algorithm described in Section II.

**Energy-aware MWIS**: the offline algorithm proposed in Section IV-A. The scheduler is configured to an offline model with no disk spin-up delay. The reduced MWIS problem is solved by the GWMIN [29] greedy algorithm.
VI. EXPERIMENTAL RESULTS

This section presents the primary results of the Cello trace. The results of the Financial1 trace were similar to the Cello trace and omitted from the paper due to space limitations. Additional results for both traces are available in our technical report [9].

A. Energy Consumption

Figure 6 compares the energy consumption among different scheduling algorithms when the data replication factor is increased from 1 to 5. The results in the figure are normalized to the energy consumption of an always-on power management configuration where disks are never spun down. In the case of replication factor 1 (no data replication), all schedules have the same scheduling results. But we still observed about 12% less energy consumption comparing to the always-on configuration because of the 2CPM scheme.

As the data replication factor increases, a scheduler now has the option to assign a request to one of its original data locations and replica data locations. But, Static still only assigns requests to their original data location, and its energy consumption remains the same, as shown in Figure 6. On the other hand, Random uniformly assigns requests to all data locations. Since the replica data locations in our experiments are uniformly selected, Random also more evenly scatters requests among disks as the replication factor increases. As a result, disks constantly receive requests and cannot be spun down, and the energy consumption of Random becomes close to the always-on configuration.

In contrast, our algorithms (WSC, MWIS, Heuristic) save more energy as the replication factor increases because more locations become available for minimize energy consumption. For example, WSC gradually reduces its energy from 0.88 to 0.73, 0.63, 0.57 and 0.52, as the replication factor increases from 1 to 5. Also as expected, among the three energy-aware scheduling algorithms, MWIS has the lowest energy consumption, while Heuristic has the highest energy consumption. This is because WMIS has a priori knowledge about requests while Heuristic only has the information of a single request at a time. However, even with Heuristic, under replication factor 3, it still significantly reduces energy consumption of the always-on configuration, Random and Static by 32.2%, 22.72% and 13.40%, respectively. Further, WSC and MWIS could achieve even lower energy by using more sophisticated set cover and independent set algorithms.

B. Number of Disk spin-up/down Operation

Figure 7 compares the number of disk spin-up/down while varying replication factors. Since the results of Static remains the same, we normalize the results to Static. In the case of replication factor 1, all scheduling algorithms except MWIS have the same number of disk spin-up/down because no scheduling is involved. The MWIS has much fewer number of disk spin-up/down because according to the offline scheduling assumption, a disk does not spin down if its successor request would suffer disk spin-up delay. As the value of replication factor increases, the number of disk spin-up/down operations is decreased for both our energy-aware scheduling algorithms and Random but for different reasons. With higher replication factor, Random more evenly distributes requests on disks, and disks become less likely to be spun down. As a result, we observe increasing energy consumption but decreasing number of disk spin-up/down for Random in Figure 6 and 7. On the other hand, more available data replications allow our algorithms to maximize energy saving by assigning requests on idle disks (and further preventing standby disks from spinning up). WSC has slightly more number of disk spin-up/down than Heuristic because WSC could attempt to spin up a disk if more requests could be serviced on it.

C. Request Response Time

Figure 8 compares mean request response time. Since offline scheduling model is an ideal case, and requests do not suffer from any disk spin-up delay in this model, the results of MWIS are omitted from this comparison. As shown in the figure, Heuristic and WSC achieve shorter mean request response time than Random and Static, because our energy-aware schedules have fewer disk spin-up/down operations as shown in Figure 7. As a consequence, requests also have less probability of suffering disk spin-up delay. The response time of WSC is longer than Heuristic because WSC batches requests with a scheduling interval (0.1s), and requests could suffer additional queuing delay. However, under data replication factor 3, WSC still significantly reduces the response time of Static from 1.11s to 0.68s (38.7% reduction).
Data replication factor limited space, we only present data placement configurations and plotted in Figure 9. Due to disks. The energy consumption was measured for each of the and replica data locations are uniformly selected among the of the original data locations follows a Zipf distribution (rather than Zipf-like), and the detailed results have been discuss ed in Section VI. With addition, we changed the degree of the locality of original data location by using a set of Zipf-like distributions, such that the value of the exponent $z$ in a Zipf distribution ($p = c/r^z$) was varied from 0 to 1 in steps of 0.1. With $z = 1$, the distribution of the original data locations follows a Zipf distribution (rather than Zipf-like), and the results have been discussed in Section VI. With $z = 0$, the distribution of both original and replica data locations are uniformly selected among the disks. The energy consumption was measured for each of the data placement configurations and plotted in Figure 9. Due to limited space, we only present Heuristic as the worst case of our energy-aware scheduling approach, since WSC and MWIS achieved less energy consumption than Heuristic.

Since the impact of the data replication factor has been discussed in Section VI-A, here we focus on the impact of data locality. As expected, the results of Random and Static rely heavily on the skewness of data locality as shown in Figure 9(a), (b). Both Random and Static cannot reduce energy consumption when data locality is uniformly distributed ($z = 0$), and they only achieve lower energy consumption when data locality is more heavily skewed among disks. In contrast, Figure 9(c) shows that when the data replication factor is 5, our energy-aware solution was still able to reduce energy by over 40% under an uniform data placement. In addition, the impact of data locality becomes less significant as the data replication factor increases. Summarizing, our technique takes advantage of the skewness of data locality and also achieves less energy consumption by exploiting more available replications.

E. Cost Function Configuration

Finally, we investigate the cost function configuration of Heuristic using the results with data replication factor of 3 as a case study. Figure 10 plots mean request response time and energy consumption under varied $\alpha$ and $\beta$ values for the cost function parameters (Recall the cost function is defined with two parameters $\alpha$ and $\beta$ in Eq. 6). For comparison, results in the figures are normalized to the number when $\alpha = 0$.

As discussed with Eq. 6, a schedule considers only the energy cost when $\alpha = 1$, and only the response time cost when $\alpha = 0$. Thus, as $\alpha$ increases, energy consumption decreases (Figure VI-E(a)), while mean request response time increases (Figure VI-E(b)). More importantly, Figure 10(a) shows that energy consumption was reduced by more than 35% when using $\alpha = 1$, while Figure 10(b) shows the request response time was reduced by more than a factor of 2 when using $\alpha = 0$. Therefore, our cost function provide a great range of trade-offs for performance optimization.

On the other hand, larger $\beta$ means less percentage of the energy cost to the overall cost. Therefore, given the same $\alpha$ value, we found larger $\beta$ has higher energy consumption as in Figure 10(a) but lower response time as in Figure 10(b). More specifically, $\beta$ determines the slope of results in Figure 10. For example, if $\beta$ is smaller, the results will shift more to the value of $\alpha = 1$. In contrast, if $\beta$ is larger, the results will shift more to the value of $\alpha = 0$.

In sum, although we show the trade-off and ability of our cost function strategy, it is not trivial to choose the best $\alpha$ and $\beta$ values, and we leave it as future work to further investigate the optimal parameter setting for our cost function as well as a better design of the cost function itself. Nonetheless, from Figure 10(a) and (b), we observed that $\alpha = 0.2$ and $\beta = 100$ gave us the most desired results in terms of both energy saving and response time. For this reason, we use this parameter setting for Heuristic and WSC schedules in the evaluation.
This paper investigated algorithms and the complexity of energy-aware scheduling for disk access requests in storage systems. Our technique is more likely to be adapted by data center administrators as it is less "invasive" as it does not interfere with existing data placement or power management policies used by the storage system at the data center. Rather, it uses these existing policies to maximize expected disk standby time and minimize energy consumption.

We discussed our approach for three scheduling models: online, offline and batch. We showed that an offline and batch scheduling problem can be solved by a weighted independent-set and weighted set-cover algorithm, respectively. A cost function strategy was also proposed to consider the trade-off between energy and performance in our online and batch scheduling algorithms. Comparing to scheduling algorithms without energy consideration, the results show that our technique significantly reduces energy consumption with fewer disk spin-up/down operations and shorter response times. Future planned work involves testing our algorithms on popular file systems that use replication such as HDFS and GFS as well as commercially available disk systems with built in power management software such as COPAN-400 and Nексsan.

VIII. ACKNOWLEDGMENTS

This work is supported by the Director, Office of Laboratory Policy and Infrastructure Management of the U.S. Department of Energy under Contract No. AC02-05CH11231.

REFERENCES


APPENDIX

Here we give our additional proofs and theorems. The variables and notations used in the proofs are shown in Table I.

**Theorem 3:** Energy-aware offline scheduling problem is NP-complete.

**Proof outline:** We prove it by using reduction from maximum independent set problem. Given any graph \( G(V,E) \), we will generate a request stream \( R \) from it as follows. For each edge \( e(v_i, v_j) \), we associate a unique request \( r_e \), and generate two additional dummy requests \( r_{e_i} \) and \( r_{e_j} \). We assume the requested data of \( r_e \) appears on disks \( d_i \) and \( d_j \), \( r_{e_i} \) only on disk \( d_i \) and \( r_{e_j} \) only on disk \( d_j \). We can then generate a request stream \( R \) by placing all \( r_e \) associated with \( e \in E \) (in any order) with large time intervals between their arrival times \( > T_B \), and \( r_{e_i} \) and \( r_{e_j} \) with the same arrival time as that of \( r_e \). It is then easy to show that an optimal schedule for \( R \) can be polynomially translated into a maximum independent set of \( G \).