eWave: Leveraging Energy-Awareness for In-line Deduplication Clusters

Raúl Gracia-Tinedo
Universitat Rovira i Virgili
Tarragona, Spain
raul.gracia@urv.cat

Marc Sánchez-Artigas
Universitat Rovira i Virgili
Tarragona, Spain
marc.sanchez@urv.cat

Pedro García-López
Universitat Rovira i Virgili
Tarragona, Spain
pedro.garcia@urv.cat

ABSTRACT

In-line deduplication clusters provide high throughput and scalable storage/archival services to enterprises and organizations. Unfortunately, high throughput comes at the cost of activating several storage nodes on each request, due to the parallel nature of superchunk routing. This may prevent storage nodes from exploiting *disk standby times* to preserve energy, even for *low load periods*. We aim to enable deduplication clusters to exploit load valleys to save up disk energy. To this end, we *explore the feasibility of deferred writes, diverted access and workload consolidation* in this setting.

We materialize our insights in `eWave`: a novel energy-efficient storage middleware for deduplication clusters. The main goal of `eWave` is to enable the energy-aware operation of deduplication clusters without modifying the deduplication layer. Via extensive simulations and experiments in an 8−machine cluster, we show that `eWave` reduces disk energy from 16% to 60% in common scenarios with moderate impact on performance during low load periods.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed Systems*; D.4.2 [**Operating Systems**]: Storage Management

Keywords

In-line deduplication; power management; disk energy

1. INTRODUCTION

Nowadays, data volumes to be managed by enterprises and data-centers are growing at an exponential rate [1]. This motivates researchers to devise novel techniques to improve storage efficiency. In this sense, *data deduplication*, a technique intended to eliminate data redundancies by splitting files into smaller and indexed chunks for avoiding storing repeated ones, has attracted much attention from both industry and academia. Particularly, *in-line deduplication clusters* are emerging and becoming increasingly popular, since

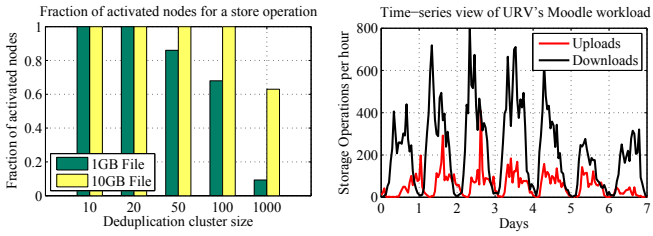


Figure 1: Fraction of nodes involved in a store operation in a deduplication cluster (10MB superchunks) (left). Workload supported by URV's Moodle servers (right).

they reproduce the operation of a single-node deduplication system at a larger scale [2, 3, 4, 5].

In a deduplication cluster, the *proxy* receives data from clients, such as files or backups, and performs an in-line process of chunking (e.g. content-based). The resulting small chunks (e.g. 4KB) are grouped into larger data units called *superchunks* [6]. Once a superchunk is created, the proxy's *data routing algorithm* decides to which storage node that superchunk will be stored *based on its content*. The main goal of routing data at superchunk-level is to improve throughput, since chunks are not large enough to make an effective use of network transfers. Then, the recipient storage node processes the chunks contained into that superchunk by looking-up for duplicates in a persistent *local index* of already stored chunks. When a duplicate chunk is detected, its contents are *deduplicated* providing thus storage savings.

Inherently, the design of in-line deduplication clusters is geared towards high performance and scalability. To wit, in a deduplication cluster, the proxy *exploits parallel access* to storage nodes depending on the *data routing algorithm*. Thus, superchunks belonging to a file are addressed based on their content towards different storage nodes [3, 4, 5]. This design leverages horizontal scalability since storage nodes can be dynamically added to the cluster to increase capacity and throughput. Existing commercial products provide high performance in-line deduplication services that are increasingly fitting the needs of scalable storage and archival of enterprises and organizations [7, 8, 9].

However, in terms of energy, the unintended consequence of this design is that *it may prevent storage nodes' disks from remaining in standby mode for larger periods to save up energy* [10, 11]. To wit, Fig. 1 (left) shows the fraction of activated nodes storing a single file in a simulated deduplication cluster. Clearly, Fig. 1 suggests that a large fraction of nodes may be in accessed in parallel on each request.

The actual problem arises if we consider the *variation on workload intensity* typically found in enterprises and organizations as a result of users' habits [12, 13] (see Fig. 1,

right). The parallel and randomized nature of data routing in a deduplication cluster makes it difficult for hard disks by themselves to *fully exploit load valleys to save up energy* by switching to low-power mode. This may lead to a potentially high expense in disk energy, since disks of storage nodes would be *kept idle even during low load periods.*

1.1 Motivation and Contribution

Our motivation is to make deduplication clusters able to *save up disk energy during load valleys.* Furthermore, to effectively achieve this goal in a dedupliction cluster it is desirable: i) *to respect the data placement defined by the deduplication system* [14] and ii) *to do not create additional data redundancy to preserve disk energy* [15], since we believe that it is contrary to the spirit of a deduplication system.

The core idea of our approach consists of decreasing transfer parallelism (i.e. performance) during load valleys to save up disk energy. To this end, we *investigate the feasibility of deferred writes, diverted access and workload consolidation* in this particular context. Technically, we contribute with:

- We propose an *analytical model to estimate the energy savings brought out by deferred writes in a deduplication cluster.* We provide insights about when deferring writes in a subset of cluster nodes pays off.
- We propose a *workload consolidation algorithm* based on our analytical model. Based on the workload, the algorithm dynamically diverts writes to a subset of nodes (or not) *considering the potential energy savings.*

To materialize our insights, we present `eWave`: a novel energy-efficient storage middleware targeted at supporting existing in-line deduplication systems. The main goal of `eWave` lies on enabling the energy-aware operation of deduplication clusters through a *temporary energy-aware data management that takes place during low load periods.* `eWave` does not modify the deduplication layer (e.g. data routing, deduplication rate) and it can be integrated with most existing systems that use stateless data routing (e.g. [2, 3, 4, 6, 5]). Through extensive simulations and experiments in an 8-machine cluster, we conclude that `eWave` is able to achieve energy savings from 16% to 60% in several scenarios with moderate impact on performance during low load periods.

This paper is organized as follows. The architecture and design of `eWave` are described in Sections 2 and 3, respectively. We describe our evaluation methodology in Section 4 and the results are reported in Section 5. We discuss the related work in Section 6 and we conclude in Section 7.

2. EWAVE ARCHITECTURE

Next, we present the overall architecture of `eWave` and its role within an in-line deduplication cluster (see Fig. 2).

Upon a file store operation, the deduplication proxy creates a set of superchunks as a result of splitting the input data stream. Moreover, the deduplication proxy's *data routing algorithm* assigns these superchunks depending on their contents to the available storage nodes within the cluster. Once this decision is taken, the deduplication proxy needs to transfer that superchunk to the selected storage node, and in that point is where `eWave` comes into play.

As shown in Fig. 2, `eWave` mediates between the deduplication proxy and the storage nodes. Basically, the deduplication proxy delegates on `eWave` the task of eventually transferring a superchunk to the selected storage node. Such a delegation is made via an *API call* from the deduplication proxy to the `eWave` proxy process running in the same physical machine. Concretely, the deduplication proxy executes

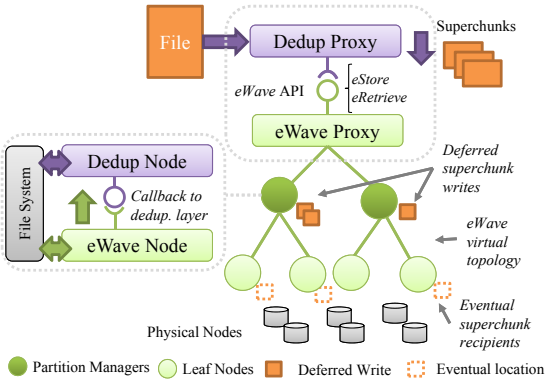


Figure 2: Architecture of `eWave`.

an `eStore(`$node$`,`sc_{id}`,`$data$`)` call, that informs `eWave` about the actual recipient (*node*) of a superchunk identified by sc_{id} (e.g. *hash*). After delegating the superchunk transfer, the deduplication proxy operates as normal by storing a new index entry relating sc_{id} with *node* for further file retrievals.

Among other tasks, the `eWave` proxy organizes storage nodes as a 2-*level hierarchy* that we call *virtual topology.* In a *hierarchy partition*, a storage node may be a *partition manager or a leaf node* —denoted as dark and light green in Fig. 2, respectively. On the one hand, a *partition manager (PM)* absorbs writes to save up energy by lengthening disk standby times of the nodes it is responsible for. On the other hand, a *leaf node (LN)* is a storage node under the responsibility of a partition manager. As we will see later on, the number of hierarchy partitions depends on an energy estimator based on system's load (see Section 3).

Following the example at Fig. 2, upon an `eStore` call the `eWave` proxy routes a superchunk to the *partition manager* responsible for its actual recipient (*node*). As a result, the partition manager keeps *node* inactive, saving up energy. As the superchunk transfer finishes, the `eWave` proxy creates an entry into its *in-memory index of deferred writes* to keep the temporal owner of this superchunk. Similarly, partition managers build a local index to relate deferred superchunks with their actual recipients to perform future migrations.

Afterwards, the partition manager migrates deferred superchunk writes to their actual recipients. When a superchunk is migrated to the its actual owner, the `eWave` process calls back the deduplication layer to deduplicate the superchunk contents, *respecting the initial data routing decision.*

On the event of a file retrieval, the deduplication proxy executes `eRetrieve(`$node$`,`sc_{id}`)` for each superchunk to be retrieved. The `eWave` proxy looks-up into its deferred writes index to retrieve that superchunk from either its actual recipient or from the partition manager responsible for it.

Next, we describe the `eWave` components: *virtual topology*, *energy-aware data management* and *workload consolidation.*

2.1 Virtual Topology

The hierarchical topology of `eWave` pursues 2 main goals: i) It provides opportunities to develop power-reduction techniques at partition managers, and ii) `eWave` consolidates load by simply resizing partitions.

A subprocess within the `eWave` proxy, named *topology manager*, keeps track of the storage nodes in the cluster and is responsible for processing topology reconfiguration events. Concretely, the workload consolidation component triggers these events depending on the system's load to add or remove partition managers from the topology (Fig. 3).

The topology accepts two operations over partitions: *merge*

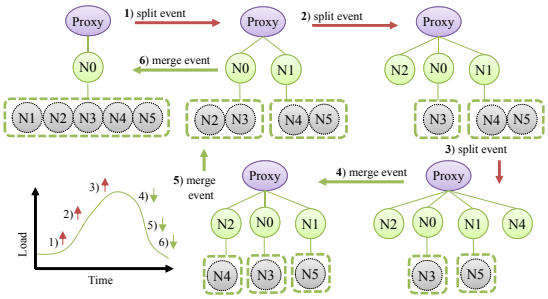


Figure 3: eWave dynamic topology adaptation.

and *split*. Upon a *merge partition event*, the topology selects the least populated partition and redistributes these nodes among the partitions with less nodes. With this policy we aim to: First, we balance load among partition managers, assuming the hash dispersion properties of content based data routing [6, 3]. Second, we maximize the size of partitions under low loads to increase the associated energy savings.

Upon a *split partition event*, the topology splits it in two the most populated partition. The node with *fewest* already stored deferred writes in the original partition is selected as partition manager of the new one. This decision pursues to avoid a partition manager to be overloaded.

The main goal of organizing storage nodes in such a way is that *partition managers are a powerful substrate for implementing energy-aware data management techniques.*

2.2 Energy-Aware Data Management

Partition managers in eWave are responsible for two major data management tasks: i) to *temporarily absorb writes* directed to leaf nodes within their partitions, ii) to *eventually migrate* deferred superchunks to the their actual recipients. All eWave nodes also exploit *superchunk caching* to increase the effectiveness of deferring writes under file retrievals.

Deferred writes. Partition managers act as *write buffers* to absorb superchunk writes of nodes they are responsible for. The objective is to enlarge standby periods of leaf nodes disks and save up power [16, 15] under write-dominated workloads, common in enterprise systems [12].

The data routing algorithm selects a certain storage node (n) to store a superchunk (s_n). Subsequently, the deduplication proxy delegates to eWave the responsibility of storing s_n at n via a *eStore* API call. Once called, the eWave proxy checks the *virtual topology* and discerns whether n is a partition manager or not. In the former case, the system acts as normal storing s_n at n. In the latter, the eWave proxy *forwards* s_n to the partition manager (pm) responsible for n, keeping the temporary location of s_n in a *in-memory index* of deferred writes for further retrievals. In turn, upon a $forward$ call, the eWave process at pm stores s_n at disk and creates the entry $s_n \rightarrow n$ into its own deferred writes index. This information will be needed for the further migration of s_n to n, its actual recipient. Similarly, when the deduplication proxy retrieves s_n via an *eRetrieve* API call, the eWave proxy process checks the deferred writes index and redirects the operation to pm, the current holder of s_n.

In terms of space complexity, the deferred writes index at the eWave proxy is the worst case since it grows linearly with the number of superchunk writes ($\mathcal{O}(n)$). However, considering that superchunks are migrated and their entries deleted from the index within a maximum time window (e.g. 30 minutes), the size of this index is limited and fits in memory. For instance, considering a sustained write throughput of 100MBps and 1MB of average superchunk size, that index would contain $\approx 180,000$ entries in 30 minutes. Thus, if every entry consists of a superchunk identifier (e.g. 64-bit hash) and a 32-bit pointer to the corresponding partition manager, the index size in memory will require 2.06MB.

Superchunk migrations. Deferring superchunk writes needs eventually reallocating them to their actual recipients.

For simplicity, we advocate for a *lazy migration policy*[1]. Under this policy, there are only two situations that force a partition manager pm to execute superchunk migrations. First, in the case that pm is exhausting its storage space. In that case, pm starts flushing superchunks belonging to a particular node or nodes, until its storage capacity becomes acceptable again. In that moment the flush operation stops. Second, if pm has not executed migrations for period greater than the *flush timeout* T (e.g. 30 minutes) it flushes all the buffered writes, which arrive coalesced to the leaf nodes.

When a superchunk s_n is reallocated at its actual recipient (n), pm deletes the content and index entry of s_n and notifies the eWave proxy to also delete that index entry.

Superchunk cache. Every eWave node incorporates an *in-memory superchunk cache*. We focus on caching at superchunk level for *avoiding leaf nodes' disks to be activated on file retrievals* and, therefore, increase the effectiveness of deferring writes under read/write workloads. Thus, we benefit from the available data locality [12, 17] to reduce read disk accesses. Superchunk caches in eWave implement a *write-through* approach. This is justified in our design given that we do not target scenarios such as file systems, where write-back caches may be suitable to absorb multiple changes on a file in memory, before persisting the state of that file on disk. Currently, the eWave cache implements a simple *LRU* eviction policy. Anyway, other policies can be introduced to improve superchunk caching effectiveness [18, 19].

3. ENERGY-AWARE CONSOLIDATION OF DEFERRED WRITES

In this section, we build a model to estimate the disk energy consumption of deferring writes in a deduplication cluster. As shown later on, this model is the corner stone of the eWave workload consolidation mechanism, which decides whether deferring writes to a subset of nodes pays off or not.

3.1 Definitions and Preliminaries

Hard disk model. Hard disks employ a Fixed Threshold (FT) scheme, which is the most popular Dynamic Power Management (DPM) mechanism for conventional disks [20]. FT disks use a fixed *idleness threshold* (I_t) to transition from *higher-power to lower-power modes*. Usually, I_t is set to the break-even time defined as the smallest period of time for which a disk must stay in low-power mode to offset the extra energy spent in spinning the disk down and up (e.g. 50 secs.). Bostoen et. al. [21] suggest to set $I_t = (E_u + E_d)/P_s$.

According to [15], a disk consumes P_h Watts when powered on and is ready to service requests (high-power mode). It also consumes P_l Watts when it is in standby mode and unable to service requests (low-power mode). A disk spin-up takes time T_u and energy E_u, whereas a spin down takes time T_d and energy E_d. P_a is the disk power of IO activity involved in data transfers. Further, we define $P_s = P_h - P_l$ as the power savings per unit of time from keeping a disk in low-power mode. In our model, these parameters are the basis to estimate the energy consumed by a hard disk.

Superchunk transfer energy (se). The maximum load that a storage node can absorb is L_M (i.e., *throughput*).

[1]We tested eager policies and we obtained only slight variations regarding performance and energy savings.

Parameter	Description	Measure
c	Storage node hard disk capacity	Bytes
L_M	Maximum in/out load of a storage node	Bytes/second
τ	Threshold that defines the point from which a node is overloaded	$\in [0, 1]$
P_h	Disk power in high-power mode	Watts
P_l	Disk power in low-power mode	Watts
P_a	Extra disk power during seek & transfer	Watts
P_s	Power savings resulting from keeping a disk in low-power mode	Watts
E_u	Energy to transition to high-power mode	Joules
E_d	Energy to transition to low-power mode	Joules
T_u	Transition time to high-power mode	seconds
T_d	Transition time to low-power mode	seconds
I_t	Hard disk idleness threshold	seconds
r	Deduplication ratio	$\in [0, 1]$

Table 1: System model parameters and description.

Accordingly, the time needed to read/write a superchunk of b bytes to disk is $T_a = b/L_M$. Therefore, considering a storage node disk in *high power* mode, the disk energy of storing/reading a superchunk of b bytes to/from disk is:

$$se = T_a \cdot P_a \tag{1}$$

Eq. 1 only captures the energy associated to the disk IO activity (seeks/transfer). This means that the rotational power (P_h) is not included in se.

However, the disk energy of transferring a superchunk (se) varies if a storage node's disk *receives the operation in low-power mode*, since we should account for the energy associated to *disk state transitions*. Concretely, we should account for the energy of transitioning to low-power mode (E_d) and the subsequent disk spin-up (E_u). In this case, the actual cost of transferring the superchunk is $se^{low} = se + E_d + E_u$.

Storage nodes progressively achieve a deduplication ratio $r \in [0, 1]$, depending on the deduplication effectiveness. A value of $r = 0$ means that no chunk has been deduplicated and $r = 1$ is full deduplication. Thus, storing a deduplicated superchunk of b bytes only requires writing $b_{dedup} = b \cdot (1 - r)$ bytes at disk in a storage node. Note that assuming a mean superchunk deduplication ratio (r) is reasonable since deduplication ratios across distinct samples of real datasets have been observed to be normally distributed [22].

To reflect this effect on the energy associated to transfer a superchunk, we denote by se_{dedup} the fact that only b_{dedup} bytes are actually written to disk. This notation is convenient to distinguish deferred writes in `eWave` (not deduplicated) from a regular stores in a deduplication node.

The problem of amortization. Merely switching disks to low-power mode may not be sufficient to save up energy if the *next storage operation is too close in time after the transition to low-power mode occurs*. To better understand this, consider that a hard disk in low-power mode spins up and serves an incoming storage request A that arrived at T^A_{ini}. At this point, after $I_t + T_d$ seconds it reaches the low-power mode again at T^A_{end}. Moreover, consider the arrival of a new storage operation B at time T^B_{ini}. We define that A *has been amortized in terms of energy* if the power savings from keeping the disk in standby mode (P_s) compensates the energy spent on disk transitions, until the arrival of B. More technically, this can be expressed as:

$$P_s \cdot (T^A_{end} - T^B_{ini}) > E_u + E_d \tag{2}$$

If Eq. 2 does not hold, operation A *is not amortized* and incurs extra energy waste compared with an energy oblivious disk. Fig. 4 illustrates this problem. Once operation A is served the hard disk reaches the low-power mode. At this point, there is a lapse of time where new storage operations would cause an extra energy waste (red area). Once the condition in Eq. 2 is satisfied, new storage operations may be served ensuring that some energy has been saved.

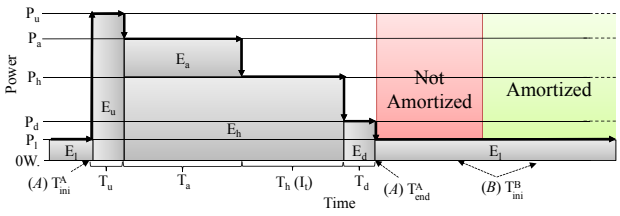


Figure 4: A disk spins up to serve operation A at T^A_{ini} and it switches to standby mode at T^A_{end}. Depending on the arrival time of operation B (T^B_{ini}), A would be amortized or not.

Unfortunately, since FT disks operate ignoring the arrival time of the next request, *we cannot ensure that storage operations will be amortized under a dynamic workload.*

Principle of coalescence. Our intuition suggests that, depending on the request inter-arrival times, we may increase amortization if requests *arrive in batch, i.e. coalesced.*

To maximize energy amortization, we apply the *principle of coalescence* [23] for superchunk writes. Concretely, partition managers in `eWave` defer superchunk writes, acting as *write buffers* and eventually migrating deferred superchunks to their actual destination as a group. However, buffering and migrating superchunks produces an additional disk energy cost which has not been properly studied in the literature. The question that we study next is, *is it beneficial to make use of deferred writes in a deduplication cluster? Under which conditions?*

3.2 Energy model of deferred writes

Let us consider a deduplication cluster that consists of one proxy node[2] and a set of s storage nodes. We assume that `eWave` organizes these nodes in a *single partition* formed by a partition manager pm and $n = (s - 1)$ leaf nodes. pm will absorb writes directed to the n leaf nodes for a time period T (i.e. *flush timeout*), after which pm migrates superchunks.

Energy overhead of deferred writes. In our system, partition managers defer writes and perform migrations of superchunks belonging to leaf nodes they are responsible for. Clearly, this buffering process requires extra disk operations that are not needed in a regular deduplication cluster.

Let us consider that a leaf node ln and a partition manager pm are both initially in high-power mode. Deferring and migrating a superchunk write involves 2 disk operations at pm (write/read), and 1 deduplicated write at ln. Thus, the associated disk energy cost is $2 \cdot se + se_{dedup}$. The worst case cost occurs when both nodes are in low-power mode. In that case, the cost is $2 \cdot se^{low} + se^{low}_{dedup}$. Thus, the *energy overhead* of deferring a write ranges from $2 \cdot se$ to $2 \cdot se^{low}$.

In what follows, we model the energy of deferring superchunk writes to better understand if it is profitable. To this end, we distinguish 3 different energy costs: i) Disk energy at partition managers (E_{pm}), ii) Leaf nodes disk energy (E_{ln}) and iii) Cost of migrations (E_m). They are described next.

Partition manager disk energy (E_{pm}). As in [21], let us define Δ as the mean time interval between the completion time of a superchunk write and the start time of the next write for a time period T. Thus, we capture the disk energy consumed *for a single write* at pm until the arrival of the next operation as:

$$e_{pm}(\Delta) = \begin{cases} se + \Delta \cdot P_h & \Delta \leq I_t \\ se^{low} + I_t \cdot P_h + (\Delta - I_t) \cdot P_l & \Delta > I_t \end{cases} \tag{3}$$

[2]We do not account for the energy consumed by the proxy.

Clearly, in Eq. 3 the I_t parameter sets a critical point beyond which the tendency of the cost function changes.

During time period T, pm will store $W = T/\Delta$ superchunks with a total energy cost of $E_{pm}(\Delta) = e_{pm}(\Delta) \cdot W$.

Leaf nodes disk energy (E_{ln}). While pm absorbs writes ln remains inactive saving up energy. Thus, the energy consumption e_{ln} of ln during T is:

$$e_{ln} = (T - I_t) \cdot P_l + I_t \cdot P_h + E_d, \quad (4)$$

where e_{ln} considers the time that ln has spent in both idle and standby modes in the absence of storage requests. We can easily extend Eq. 5 for an arbitrary number of leaf nodes within a topology partition as $E_{ln}(n) = e_{ln} \cdot n$.

Cost of migrations (E_m). Once pm has absorbed W writes and T is reached, it is the moment of performing *superchunk migrations*. To constrain the problem, we assume that the number of superchunk migrations is higher than the number of leaf nodes, i.e. $W >> n$, which is the most common scenario. Therefore, we can assume the dispersion properties of hash functions used as data routing features [6].

In this case, the number of deferred writes that will receive any leaf node will not deviate too much from the mean. Then, we can reasonably assume that $M = \lceil W \cdot n/(n+1) \rceil$ writes will be migrated as they do not belong to pm. Thus, the cost of migrating these writes (E_m) can be described by:

$$E_m(n) = M \cdot (se + se_{dedup}) + n \cdot E_u, \quad (5)$$

where we capture the energy of reading superchunks from pm and writing them at deduplication nodes plus their disk spin-up cost. Note that this is the worst case cost, since pm reads all the superchunks from disk before migrating them. This yields that the superchunk cache provides no benefit.

Therefore, the total disk energy cost of deferring writes is:

$$E_{total}(\Delta, n) = E_{pm}(\Delta) + E_{ln}(n) + E_m(n) \quad (6)$$

Eq. 6 describes the disk energy consumption of a s-node cluster deferring writes in a single manager pm for a time period T, plus the associated migrations. Furthermore, we apply this model to an arbitrary number of `eWave` partitions by scaling n and Δ.

Regular cluster disk energy (E_{reg}). In a regular deduplication cluster armed with FT disks, the energy cost is described by Eq. 3 at any given node. So the total energy cost should be multiplied by s. The difference is that the average arrival rate per node is λ/s, since superchunks are immediately routed from the proxy towards their recipients instead of being deferred. Additionally, the cost of disk writes in Eq. 3 is se_{dedup} (se^{low}_{dedup}) instead of se (se^{low}), since store operations always involve deduplication.

Model Discussion. The cost of deferring writes in `eWave` mainly depends on: the *workload intensity* (λ, b) and the disk *idleness threshold* I_t. Moreover, the number of *leaf nodes per partition* greatly determines the energy savings.

To shed some light on this, in Fig. 5 we compare the relative energy consumption (y axes) of a simple write-only workload in a deduplication cluster with/without deferring writes—denoted by DW and No DW, respectively. Thus, a value $< 0\%$ means that deferring writes saves up energy. The x axes is the inter-arrival time Δ between writes of $b = 1$MB superchunks. The period before migrations is $T = 1$h and, for simplicity, $r = 0.0$ (no deduplication).

From Fig. 5 we infer three main observations:

O1: Deferring writes may report important benefits in the case of *low system loads*. Otherwise, it may incur extra

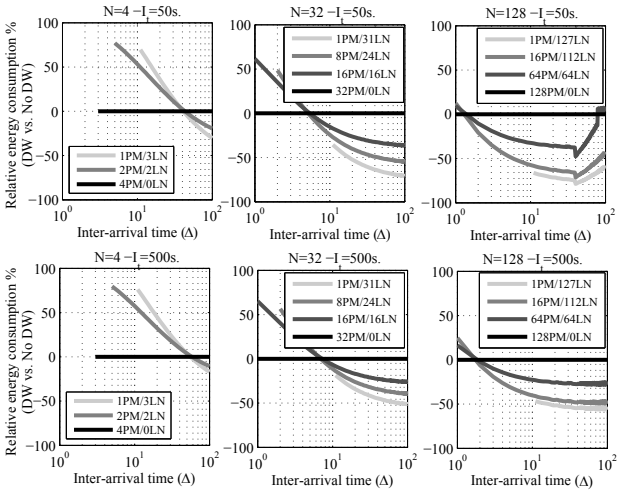


Figure 5: Results of our energy model in a cluster with/without deferred writes (No DW/DW). We consider a write-only workload of superchunks arriving every Δ seconds during $T = 1$ hour (IBM Ultrastar disk, see Table 2).

energy waste due to the cost of migrations. To wit, as visible in Fig. 5, in a 4-node cluster deferring writes saves up energy for write loads $<$ 2.4MBps whereas for a 32-node cluster deferring writes pays off for loads $<$ 20MBps ($I_t = 50$s).

O2: A *shorter I_t induces higher energy savings* when deferring writes. This can be inspected comparing the energy savings of the same topology for both I_t values. This is reasonable since leaf nodes exploit better their standby periods.

O3: There is a *cross-over point* beyond which deferring writes is beneficial. Moreover, beyond this point, the *topology with fewest managers exhibits the highest savings.*

We also observed that varying the deduplication rate r in our model has a low impact on the achieved energy savings. This will be evaluated in Section 5.

Our analysis lead us to the design of an energy-aware workload consolidation mechanism for `eWave`. In the next section, we describe how `eWave` scales the number of partition managers to adopt the most energy efficient topology.

3.3 Workload Consolidation

`eWave` exploits workload variability to reduce power consumption by dynamically resizing topology partitions (i.e. *workload consolidation*). Conversely to other systems [24, 25, 26], `eWave` takes consolidation decisions based on an *energy estimation model* (see Section 3.2) rather than just considering load (i.e. requests/sec.). We observed that depending on several parameters (e.g. I_t, cluster size) a certain topology may incur energy savings or extra expenses given the same load. This makes the load itself an insufficient source of information to take consolidation decisions.

Every time interval t (e.g. 10 minutes) a monitoring process running at the `eWave` proxy collects information about the system's load. Such information is easily gathered by inspecting the `eWave` API calls executed by the deduplication proxy. Thus, we obtain a trace ℓ_t that represents the system load during the previous interval t, where λ_t is the operation mean arrival rate and b_t the mean superchunk size. We aim to create a *reactive mechanism to dynamically adapt the topology* to the most energy efficient configuration. Thus, the problem that we face can be expressed as: given a system load trace $\ell_t(\lambda_t, b_t)$, which is the number of partition

managers m that minimizes the energy waste?

As a natural requirement, the number of partition managers that minimizes disk energy costs must also absorb the current workload. Furthermore, we observed that, in those situations where deferring writes pays off, the topology with *fewest* partition managers (m) exhibits the *highest energy savings* —see *O3* in Section 3.2. This leads us to find m as:

$$m = \left\lceil \left(\frac{b_t}{\lambda_t} \right) / (L_M \cdot \tau) \right\rceil \quad (7)$$

Eq. 7 outputs the minimum number of partition managers that can serve writes for a workload characterized by λ_t and b_t. Besides, Eq. 7 introduces a new parameter, called *load threshold* and denoted by $\tau \in [0, 1]$, that scales down the maximum load that partition managers can support.

On the one hand, a value $\tau = 1$ means that partition managers will absorb writes up to their maximum capacity (L_M). This yields potentially *higher energy savings, at the cost of degraded transfer performance* (i.e. aggressive migrations, lower transfer parallelism). On the other hand, a value of τ close to 0 prioritizes transfer performance, since `eWave` adds new partition managers before overloading the existing ones. In our model, τ is a key setting that should be appropriately defined depending on the deployment scenario (e.g. workload, hardware). We show in the evaluation how τ enables us to *trade-off energy and transfer performance*.

However, building a topology with the minimum set of partition managers capable of absorbing a workload does not guarantee that it is beneficial in terms of energy. In other words, `eWave` should operate only if deferring writes to m partition managers consumes less energy than the cluster in regular operation. Formally, considering $\Delta_t = 1/\lambda_t$, we express the previous condition as:

$$m \cdot E_{total}(\Delta_t \cdot m, \left\lceil \frac{s}{m} \right\rceil) < s \cdot E_{reg}(\Delta_t \cdot s) \quad (8)$$

The left member of Eq. 8 represents the energy estimated of deferring writes in a topology with m partition managers. The right member of Eq. 8 estimates the energy for a regular cluster. If inequality Eq. 8 holds, it is beneficial to change the topology and let partition managers start buffering writes. Otherwise, the workload is too intense to amortize the subsequent migration costs, and we let the system operate as normal (i.e. $m = s$).

Our workload consolidation mechanism can work either using *mean workload values* or replaying *complete interval traces*. Making use of mean workload values (λ_t, b_t) requires little information at runtime. However, the associated downside is that we assume: i) *good load balancing* among storage nodes, and ii) the *mean arrival rate* and *superchunk size* (λ_t, b_t) are representative values of the workload for an interval t. We assess the accuracy of our model using both mean workload values and complete interval traces in Section 5.

4. EVALUATION FRAMEWORK

4.1 Prototype and testbed scenario

We developed an in-line deduplication system prototype. This prototype follows a 1-Proxy/N-Nodes architecture. The proxy node includes content-based chunking/superchunking, several data routing algorithms (as proposed in [6]), a simple membership protocol as well as other utilities (e.g. logging, failure detection). It consist of 2,000 lines of Python code and is based on the ClusterDFS system[3] that provides fast I/O and concurrency thanks to the asynchronous network library and lightweight threads offered by `gevent`.

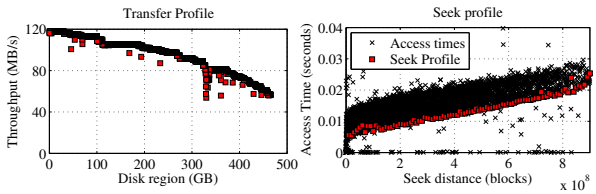


Figure 6: Western Digital WD5000AAKX seek/transfer profiles used to estimate disk energy in our experiments.

We also developed `eWave` as a library which can be called by the deduplication prototype. `eWave` consists of 1,200 lines of code, and it is also written in Python to ease the interaction with the deduplication prototype. The `eWave` proxy includes a process to create the *virtual topology*, a *load monitor* that updates the system's load on each API call, and an index of superchunks deferred to partition managers. In addition to a deferred writes index, `eWave` storage nodes include one write-through LRU superchunk cache.

Our experimental scenario consisted of 10 machines; 1 client, 1 proxy and 8 storage nodes. All the machines incorporate an i5 processor, 4GB DDR2 of RAM and 500GB of hard-disk storage (see Table 2). The operating system employed was a Debian Linux distribution. Moreover, nodes were connected via Fast Ethernet (100Mbps) switched links.

Our objective is to quantify the energy gains of `eWave` compared with a traditional deduplication system under different scenarios and the implications on performance.

4.2 Experimental energy estimation

In our experiments, we want to account for the disk energy of all storage nodes in the cluster. This makes it difficult to instrument every node with physical power meters. For this reason, we opted for a trace-based energy estimation approach capturing the disk activity of all nodes. In our view, this method provides the best trade-off between practicality and accuracy, as proven in other works [27, 28].

Our estimation of a hard disk energy consumption consists of modeling first its behavior (benchmarking) and then trace all the operations scheduled to the device during a experiment. Thanks to benchmarking, we can infer the *seek profile* (S_p) and *transfer profile* (T_p) of a hard disk for any access (see Fig. 6). Provided the power consumed by the disk in these stages, we can easily infer the energy consumed. From these benchmarks we can deduce a function which accurately models the energy consumed by a operation. Afterwards, we need to trace all the operations issued to the hard disk: *location, size and seek time*. Replaying these traces with the inferred energy consumption models, we are able to estimate the amount of energy consumed by a disk.

4.3 Simulation Environment

To asses our solution in a wider spectrum of parameters and configurations we make use of simulations. In our simulator, `eWave` contains almost the same logic and functioning than the deduplication prototype in a real experiment. However, deduplication and `eWave` nodes are embedded into processes of our discrete-event simulation framework[4]. This enables us to simulate the communication network and the energy estimation upon storage requests.

We simulate a deduplication proxy with a raw throughput of 100MB/s connected in parallel via a Fast Ethernet (10MB/s) network to storage nodes —this is the network

[3] `https://github.com/llpamies/clusterdfs`

[4] `https://simpy.readthedocs.org/en/latest/`

Parameter	`IBM Ultrastar 36z15` (simulation)	W. Digital `WD5000AAKX` (experiments)
Interface	SCSI	SATA
Disk Rotation	15,000 RPM	7,200 RPM
High Power (P_h)	13.5W	4.9W
Low Power (P_l)	2.5W	0.816W
Power Active (P_a)	65mJ. per 8Kb block [20]	n.a.
Power Transfer	n.a.	1.59W ($+P_h$ = 6.49W)
Seek Power	n.a.	3.03W ($+P_h$ = 7.93W)
Spin-up Time (T_u)	10.9s.	3.794s.
Spin-down Time (T_d)	1.5s.	0.291s.
Spin-up Energy (E_u)	135J.	63.125J.
Spin-down Energy (E_d)	13J.	4.419J.

Table 2: Hard disk specifications used in our simulations and experimental trace parsing.

used in our experimental cluster. Besides, in our simulations we make use of an IBM Ultrastar 36z15 disk (see Table 2).

In our simulator, we calculate the energy consumption of disks by considering their idle/standby periods (dominant cost [15]) and the energy of transferring data to disk (active power in Table 2). Therefore, we do not take into account the movement of the mechanical disk actuator, as we do in our experimental assessment. Due to the complexity of our evaluation scenario, our objective is to verify our simulation results through experimentation in relative terms.

4.4 Workload model

In our simulations, we make use of the following workloads to describe file accesses (see Fig. 8):

URV's Moodle. This trace captures the upload and download patterns of the URV Moodle's system for 8 months. For each trace row, the provided information contains the operation type, the file id, file size, among other features.

Ubuntu One Service (U1). In the context of the FP7-CloudSpaces EU project on Personal Clouds we are analyzing the U1 service. We captured upload requests of users during one month (November 2013). The (anonymized) trace provides upload access patterns, file sizes and file types.

The additional complexity of a workload model to evaluate `eWave` is that we simultaneously need access patterns to files and the actual content of files (or at least hash information). However, to our knowledge, there is no public trace containing both sources of information. In essence, we want to analyze the impact of data routing features and different deduplication ratios on `eWave` performance. To this end, in our simulator we model the contents of each file store operation with a sample of superchunk hashes calculated from real backup data (Wikipedia dumps, Linux images). These superchunks include two data routing features as proposed in [6]: superchunk *minimum chunk hash* and *first chunk hash*. Besides, we emulate different degrees of deduplication at each file with a mean value (r).

In the experiments, we execute a write workload consisting of a set of Linux distribution images (e.g. archival), which is a common use case of deduplication clusters [2, 3, 29].

4.5 Metrics

We compare `eWave` with a regular deduplication cluster making use of the following metrics:

Disk Energy: Amount of Joules consumed by a single disk during a period of time. We also use the *relative energy consumption* metric to compare `eWave` (e_1) with a regular cluster (e_2) defined as $(e_1 - e_2)/e_2$.

Disk idle times: This metric quantifies the length of disk idle times to understand the potential energy gains achievable by turning off disks.

Performance: We make use of *file transfer times* from the proxy's perspective to quantify performance in our simulations. Moreover, in our experiments we use the *proxy goodput* defined as file size/transfer time (measured in MBps).

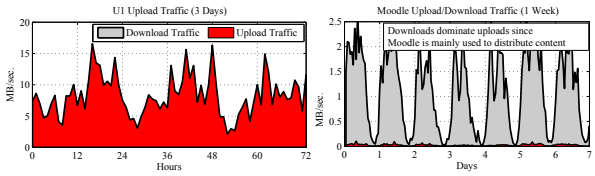


Figure 8: Workload traces used in our simulations.

We do not quantify deduplication metrics (e.g. effective deduplication), since it is responsibility of the deduplication system. The above metrics will be analyzed on several of scenarios to observe the impact of varying the value of important parameters (e.g. idleness threshold, cluster size).

5. RESULTS

Unless otherwise stated, for both `eWave` and a regular cluster, we set by default $I_t = 100$s (IBM Ultrastar). `eWave` includes a 500MB LRU cache and a flush timeout $T = 1800$s to trigger migrations. We set $\tau = 0.1$ to avoid saving up energy at the cost of high transfer speed degradation.

Regarding the workload characteristics, the average superchunk size is 1MB and we use the *first chunk hash* as data routing feature. We simulate that the mean superchunk deduplication ratio is $r = 0.2$. In any case, we inspect the role of all these parameters throughout this section.

5.1 Measuring energy savings

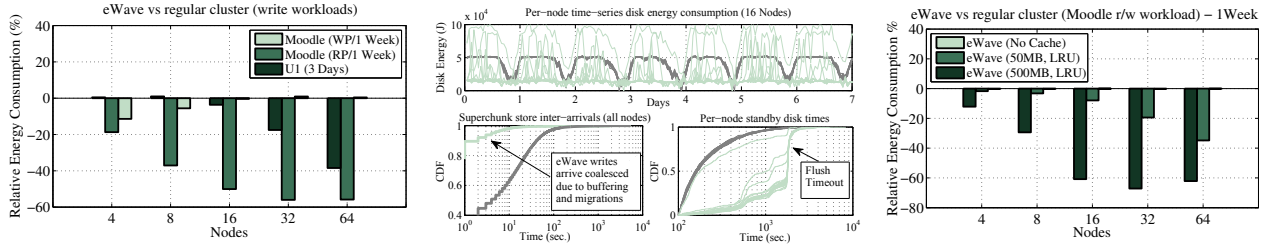
Next, we overview the energy savings provided by `eWave` compared with a regular deduplication system. To better understand the performance of `eWave`, we discern between write workloads and mixed read/write workloads. Note that the energy savings presented in this section are not at the cost of high performance degradation (see Section 5.2).

Write workloads. We evaluate `eWave` against a regular cluster under 3 *write workload patterns*: Moodle and U1 write patterns (WP) and Moodle read pattern (RP).

Fig. 7a shows the energy savings that `eWave` provides under write workloads. In general, we observe that `eWave` may bring important disk energy savings in many scenarios compared with a regular deduplication cluster ($16\% - 57\%$). `eWave` exhibits the best energy savings in the Moodle (RP) case ($19.3\% - 57\%$), since it is a light workload that makes it very difficult for FT disks to remain in standby mode. Furthermore, in those situations where the workload impedes to save up energy, the dynamic workload consolidation of `eWave` keeps the system in the same degree of disk energy consumption. That is, an energy overhead of only 0.93% (U1 trace, 4 nodes) has been the worst case tested.

The executed workload patterns help us to understand the behavior of `eWave` in disparate situations. In fact, as predicted by our energy model, `eWave` does not provide significant benefits when the workload is high or extremely low *compared with the number of nodes in the cluster*. This can be confirmed looking at Moodle (WP) for $N \geq 32$ and U1 for $N \leq 8$ in Fig. 7a, respectively. Under high workload intensity, `eWave` cannot amortize buffered writes at the partition manager with sufficiently large standby times of leaf nodes. Conversely, FT disks seem to deal well with extremely low workloads since request inter-arrival times become larger.

However, the strategy of diverting and deferring writes to a subset of partition managers is successful in most situations. To understand this, Fig. 7b provides a time-series view of the disk energy consumption of each node in the cluster. We observe that a several `eWave` nodes (partition managers) exhibit a much higher energy consumption than the rest of nodes, since they are absorbing writes for them.



(a) Disk energy of `eWave` vs a regular cluster under 3 write workloads.

(b) `eWave` (green) and a regular cluster (gray). Moodle (RP) write workload.

(c) Disk energy of `eWave` vs a regular cluster under a mixed read/write workload.

Figure 7: Energy savings provided by `eWave` under various workloads compared with a regular deduplication cluster.

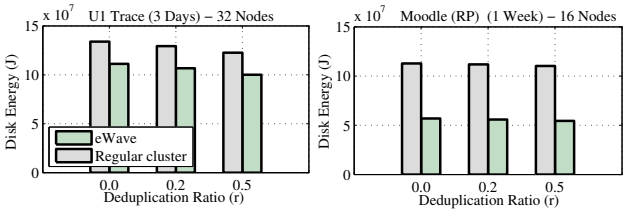


Figure 9: Impact of r on disk energy for write workloads.

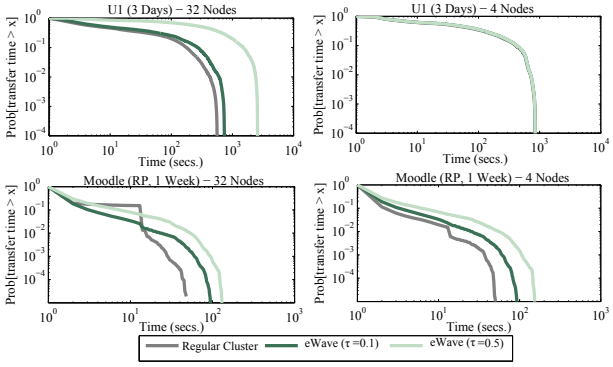


Figure 10: `eWave` transfer performance for various τ values.

Consequently, the rest of nodes are in low-power mode for longer periods of time, saving up energy. On the other hand, in a regular cluster (gray lines) all nodes exhibit very similar energy consumption numbers as well as shorter inter-arrival times among write requests. This fact prevents nodes from switching to low-power mode, being worse for larger clusters.

We executed the same write workloads considering other deduplication ratios (see Fig. 9). The most important conclusion that we draw from Fig. 9 is that `eWave` is still beneficial even for high deduplication rates ($r = 0.5$). The main reason for this behavior is that the dominant energy costs belong to the rotational disk power [15]. Thus, even for high r values, upon a store request a node in standby mode should spin-up the disk if a superchunk contains a single not deduplicated chunk. `eWave` mitigates this problem by coalescing store operations. Besides, as expected, we observe that both a regular cluster and `eWave` consume less energy for higher r values. Furthermore, in the case of U1, the energy consumption rates among distinct r values are more significant due to the higher workload intensity.

We conclude that `eWave` is specially beneficial to reduce the energy waste of over-provisioned deduplication clusters and/or scenarios with high variance of workload intensity[5].

Read/Write workloads. Now we explore the impact of reads in the energy savings achieved by `eWave`. To this end, we execute the complete Moodle trace (reads and writes), varying the `eWave` superchunk cache configuration.

In Fig. 7c we see that deferring writes to save up energy is by itself useless when file retrievals are frequent (*No Cache* case). The reason is that nodes which aim to remain in low-power mode are frequently activated by read requests.

Fig. 7c shows that the energy savings in the presence of reads are related with the cache size. To wit, a larger cache has greater probabilities of containing the requested block in memory, avoiding thus extra disk accesses. For instance, in a 4-node cluster, nodes with 50MB caches reported around $9.09\% - 13.06\%$ of cache read hits, whereas 500MB caches provided read hit ratios between $39.33\% - 47.78\%$.

[5]We did not observed significant differences between the min and first chunk routing policies, suggesting that `eWave` is able to handle distinct data routing algorithms.

Therefore, `eWave` is able to tackle file retrievals, specially in the case of high data locality. However, if the system supports read dominated workloads with low locality, deferring superchunk writes may provide poor energy savings.

5.2 Performance impact

Now we analyze the performance impact of `eWave`. We pay special attention on the load threshold parameter (τ), which is responsible for trading-off performance and energy. Fig. 10 plots file transfer times (CCDF) of both a regular cluster and `eWave` for various τ values under two write workloads.

First, the performance penalty of `eWave` greatly depends on the τ parameter. A mistaken value of τ may negatively affect the performance of transfers, as we demonstrate in Fig. 10 ($\tau = 0.5$, U1 workload). The reason for this performance deterioration is twofold: First, `eWave` *inherently limits the exploitable transfer parallelism* since writes are served by a subset of partition managers. Second, partition managers may be *simultaneously* serving requests from the proxy and migrating superchunks, which induces congestion.

In the U1 case ($N = 32$), `eWave` uploads for $\tau = 0.1$ are only moderately slower than a regular cluster. To wit, in the 90^{th} percentile, `eWave` uploads take 39 seconds more to complete compared with a regular cluster (+31%). Note that around the 30% of uploads exhibit the *same performance* in both `eWave` and a regular cluster. This is because these uploads occur in periods of higher load where `eWave` does not take action.

Following the U1 case, `eWave` consumes $-17,55\%$ and -24.22% of disk energy compared with a regular cluster, for $\tau = 0.1$ and $\tau = 0.5$ respectively. This leads us to the next conclusion: `eWave` *makes it feasible to save up energy introducing only moderate performance penalty over non-peak workload transfers*. Further, we observe that the *energy savings are not proportional to the impact on performance*. To wit, by sacrificing a $6,67\%$ savings on disk energy `eWave` re-

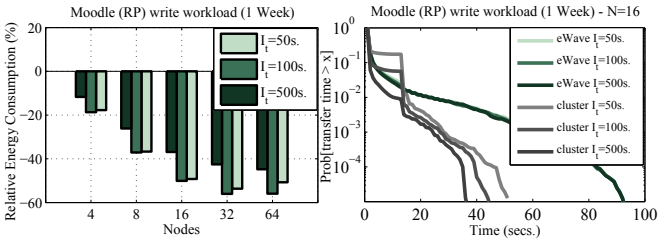


Figure 11: Impact of I_t on energy savings and transfer times.

ports transfers several times faster depending on the τ value. The Moodle trace follows a similar trend, but the impact of varying τ is less significant on energy since the workload is less intense than for U1. To automatically adjust τ given performance constraints is left for future work.

In the U1 scenario with $N = 4$, we observe that `eWave` does not take action since the workload is too intense compared with the cluster size, making the cluster to work as normal.

Interestingly, daily patterns present in the Moodle trace reveal that FT disks by themselves may not only fail on exploiting load valleys to save energy, but they also may induce performance problems. To wit, under low loads superchunk inter-arrival times become larger, which produces frequent disk spin-ups upon new requests. This fact is more evident for larger clusters and higher load variability and motivates diverting access to `eWave` partition managers.

5.3 Sensitiveness to idleness threshold

We explore the effect of the idleness threshold (I_t) on the disk energy and transfer performance. Thus, Fig. 11 shows the relative energy consumption and transfer times of `eWave` and a regular cluster under the Moodle (RP) write workload.

Considering energy, both a regular cluster and `eWave` consume more energy for larger I_t values. For instance, a regular cluster disks consume +10.12% for $N = 4$ and +26.43% for $N = 32$ of energy comparing $I_t = 50$s with $I_t = 500$s. We also analyzed this for the U1 trace, observing a similar trend for larger cluster sizes (e.g. $N \geq 32$), since the workload is more intense than the Moodle scenario. As expected, a shorter I_t provide more opportunities to conserve energy.

However, focusing on transfer times, we observe that in a regular cluster making use of a shorter I_t may degrade performance under moderate/low loads. This is due to the number of disk spin-ups needed upon storage requests, which in turn has an important role on the disk lifetime. To wit, in Fig. 11 for $N = 16$ a regular cluster executed $45,652$ disk spin-ups for $It = 50$s, whereas for $It = 500$s that number was $1,472$. Thus, a regular cluster by itself importantly degrades transfer performance and disk reliability to obtain energy savings by reducing I_t. On the other hand, `eWave` seems less sensitive to the value of I_t in terms of performance —for U1 the performance impact of I_t on `eWave` is higher than for the Moodle trace, but it is still lower than in a regular cluster. Besides, in Fig. 11 for $N = 16$ `eWave` exhibited $7,266$ disk spin-ups for $It = 50$s and $5,305$ disk spin-ups for $It = 500$s, providing an attractive trade-off among energy savings, transfer performance and disk reliability.

We conclude that there exists a tight relationship between the disk I_t parameter and the inter-arrival times of requests (i.e. workload intensity). This has to be considered for making an efficient use of the cluster in various aspects. We consider to dynamically adjust I_t as future work [30, 31].

5.4 Assessing the deferred writes energy model

Next, we evaluate the accuracy of our energy estimation

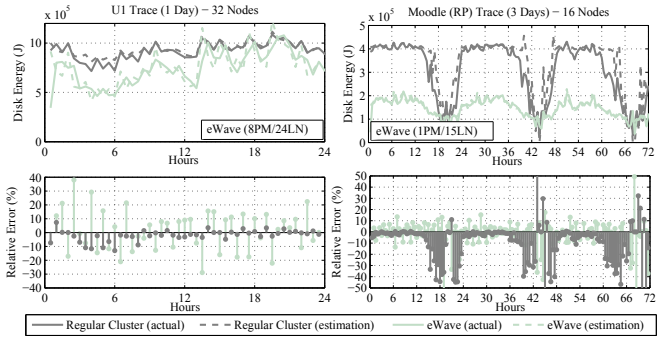


Figure 12: Accuracy of our deferred writes estimation model based on mean workload values.

model for deferred writes. We have shown that our model satisfactorily guides `eWave` based on mean workload values. However, we want to understand in depth its accuracy in different scenarios. To this end, Fig. 12 shows the actual and estimated energy consumption of `eWave` (fixed topology) and a regular cluster under various write workloads.

In first place, we observe that for an important fraction of samples (72%) the relative error falls within a range of ±10%. This makes our model accurate enough to guide `eWave` on taking decisions about the topology.

However, we observe few moments that report important estimation errors —specially in the case of a regular cluster under the Moodle (RP) workload. We identified two factors that are difficult to capture with simple mean measures: i) Rapid and high workload variability (e.g. day to night workload change) and ii) Non-perfect load balancing.

We also tested our energy estimation model in a trace based fashion, that is, periodically replaying (e.g. 600s) a trace sample to estimate the energy consumption of the cluster. As expected, this approach reported a much higher accuracy (100% of samples fall within an error range of ±1% for a regular cluster in the Moodle tarce) but at the cost of keeping more workload information at runtime to feed our model —instead of using simple mean measures.

We conclude that our model provides a fair estimation accuracy when working just with mean workload measures. However, we can fully exploit its potential by adding a complete view of the workload.

5.5 Experiments

Next, we show the results of our experimental testbed in our university labs. The testbed consists on a single client uploading several linux distribution images. Concretely, the client uploaded 20GB of data in 2 hours. We set up a cluster of 8 storage nodes. At each storage node we captured disk activity during the experiment. We repeated the experiment 5 times with/without `eWave` to confirm our results.

Throughout the experiment, `eWave` maintains the same fixed topology (1 partition manager and 7 leaf nodes). `eWave` executes a lazy migration policy (flush timeout $T = 1800$s). We parsed the disk IO traces assuming $I_t = 100$s

Energy savings. `eWave` provides important energy savings to the deduplication system. That is, `eWave` consumed −24.06% of energy compared with a regular cluster. We observe in Fig. 13 (left) that deferring writes in the partition manager significantly reduces the rotational disk power consumption at leaf nodes, which is the most important fraction of energy [32]. To wit, in our experiments, the disk rotation accounts for the 99.72% of disk energy consumed in a regular cluster, whereas transfer and seek energy represent the

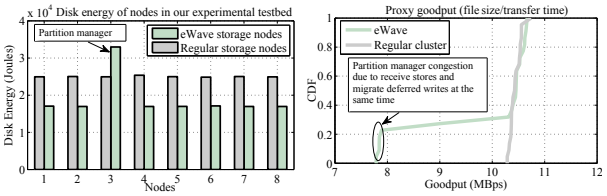


Figure 13: Per-node disk energy (left) and performance of proxy transfers (right) in our experiments.

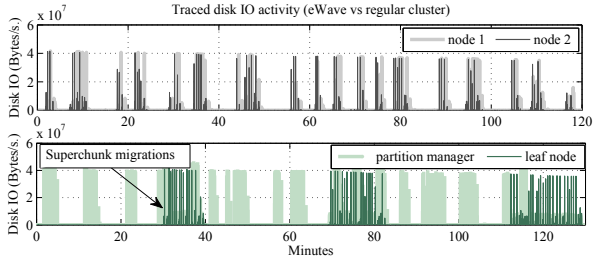


Figure 14: Disk IO of nodes in `eWave` and a regular cluster.

0.139% and 0.136%, respectively.

Fig. 14 provides a time-series view of the traced disk activity for both `eWave` and a regular cluster. Observably, the partition manager is activated on each file store operation. This causes an increased energy consumption compared with leaf nodes (as shown in Fig. 13, left). However, leaf nodes are activated only during superchunk migrations, being in low-power mode in the rest of experiment. This reduces the overall energy cost compared with a regular cluster.

Performance impact. In terms of performance, Fig. 13 (right) shows the *proxy goodput*, that relates the size of a file with the time needed to store it in the system. For most store operations, `eWave` and a regular cluster exhibit very similar performance numbers. This is due the limitations of our experimental hardware: the proxy is a regular server (as any other storage node) and the network maximum speed is 100Mbps (Fast Ethernet). Therefore, the proxy cannot exploit parallel connections in the same way that in a real cluster to observe the impact of loosing parallelism in `eWave`.

In Fig. 13 (right) we see that the 23% of store operations in `eWave` exhibit a degraded performance. This is due to congestion, since the partition manager may be concurrently serving storage request from the proxy as well as performing migrations. This opens the door to further optimize the virtual topology of `eWave` to avoid this situation.

6. RELATED WORK

Energy-aware Storage. Much effort has been devoted on making storage systems energy-efficient. Bostoen et. al. [1] excellently describe the literature in this field and provide a comprehensive taxonomy of the existing techniques.

Diverted access. A relevant power-reduction approach is *popular data concentration* (PDC) [20, 33]. PDC benefits from the typical skewed distribution (e.g. Zipf) of file access to distribute data across disks accordingly. This approach intrinsically *diverts* the majority of accesses towards a subset of disks. This concept has been also applied in Hadoop clusters to differentiate among cold/hot cluster zones [34, 35]. Rabbit [26] diverts access in a distributed file system (HDFS) to achieve *power proportionality*. Rabbit targets read-dominated workloads, common in a MapReduce scenario. Rabbit elaborates on the idea of *primary set*, which is the minimal number of nodes to hold a single file replica. Rabbit places r different replicas on r different subsets of nodes, which enables diverting access to active nodes while keeping other nodes saving energy. Similarly, Sierra [25] targets general workloads including read and writes by integrating write off-loading. SRCMap [24] combines several power reduction techniques such as live migration, replication and write off-loading. SRCMap increases the idle periods substantially by also off-loading popular data reads. SRCMap uses replication by creating partial replicas that contain the working sets of data volumes. `eWave` focus is on write dominated workloads diverting access to partition managers to defer writes. In contrast to most works, `eWave` respects the data placement defined by the system. Moreover, `eWave` makes use of superchunk caching to handle reads instead of creating additional redundancy to divert read accesses.

Workload consolidation. SRCMap [24] provides fine grained power proportionality via storage virtualization and workload consolidation. Rabbit [26] and Sierra [25] also provide workload consolidation mechanisms based on the load of the system. In contrast, `eWave` provides workload consolidation based on an dynamic energy estimation.

Deferred writes. Write off-loading [16] is a technique intended to save up energy by increasing idle disk times in write-dominated workloads. Every data volume (one/many physical disks) has a space reserved for temporary storage of off-loaded blocks. This means that other active volumes transiently absorb writes directed to inactive volumes to preserve their idleness. An off-loaded block is copied to its target volume when this volume becomes active because of a read-cache miss. In this line, authors in [36] study the use of write buffers in a parallel I/O storage system to reduce power consumption. The present paper extends these works providing a analytical energy model of deferred writes.

Key differences with previous works. The goal of `eWave` is to make current systems (e.g. [29, 6, 5, 37]) energy aware, without altering their behavior (e.g. data routing).

However, in our context, this goal can only be effectively achieved by i) respecting the *data placement defined* by the deduplication system [14] and, ii) being *agnostic of the file system*, since the deduplication layer may be really coupled with it [17]. Besides, conversely to other works, `eWave` *does not create additional redundancy to preserve energy*, since we believe that it is contrary to the spirit of a deduplication cluster [15]. To our knowledge, there is no power-aware storage middleware that copes with these requirements.

7. CONCLUSIONS

This work is the first attempt to leverage energy-awareness for in-line deduplication clusters. To this end, we propose `eWave`: an energy-efficient data management middleware tailored to the specific needs of deduplication clusters. The design of `eWave` lead us first to study the potential energy savings of deferring writes in this kind of storage systems, which has not been properly analyzed in the literature. As a result, we obtained an energy model of deferred writes that guides the `eWave` workload consolidation mechanism. To our knowledge, `eWave` is the first system that consolidates load purely based on an energy metric.

Via extensive simulations and experiments in an 8-machine cluster, we evaluated the effectiveness of our design and the potential energy savings of `eWave`. We conclude that `eWave` may report energy savings up to 60% in several scenarios and workloads with moderate impact on performance during non-peak loads. Further, `eWave` is specially beneficial to reduce the energy waste of over-provisioned clusters and/or scenarios with high variance of workload intensity. We believe that this makes `eWave` suitable for a variety of enterprise and organizational deduplication clusters.

Acknowledgment

This work has been partly funded by the European Union through project FP7 CloudSpaces (FP7−317555). We thank Toni Cortés for his suggestions on the final stages of this article. We also thank to Jordi Pujol-Ahulló for his help on gathering the URV's Moodle traces and John Lenton from Canonical for his help on collecting traces from U1.

8. REFERENCES

[1] T. Bostoen, S. Mullender, and Y. Berbers, "Power-reduction techniques for data-center storage systems," *ACM Computing Surveys*, vol. 45, pp. 33:1–33:38, 2011.

[2] D. Bhagwat, K. Eshghi, D. D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *MASCOTS'09*, 2009, pp. 1–9.

[3] D. Frey, A.-M. Kermarrec, and K. Kloudas, "Probabilistic deduplication for cluster-based storage systems," in *SoCC'12*, 2012, pp. 17:1–17:14.

[4] T. Yang, H. Jiang, D. Feng, Z. Niu, K. Zhou, and Y. Wan, "Debar: A scalable high-performance de-duplication storage system for backup and archiving," in *IPDPS'10*, 2010, pp. 1–12.

[5] C. Dubnicki et. al., "Hydrastor: A scalable secondary storage," in *FAST'09*, 2009, pp. 197–210.

[6] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters," in *FAST'11*, 2011, pp. 15–29.

[7] "Emc centera," http://www.emc.com.

[8] "Sepaton s2100," http://www.sepaton.com.

[9] "Nec hydrastor," http://www.necam.com/hydrastor/.

[10] J. Leverich and C. Kozyrakis, "On the energy (in) efficiency of hadoop clusters," *ACM SIGOPS Op. Sys. Review*, vol. 44, no. 1, pp. 61–65, 2010.

[11] Z. Li, K. M. Greenan, A. W. Leung, and E. Zadok, "Power consumption in enterprise-scale backup storage systems," in *FAST'12*, 2012, pp. 1–6.

[12] A. W. Leung, S. Pasupathy, G. R. Goodson, and E. L. Miller, "Measurement and analysis of large-scale network file system workloads." in *USENIX ATC'08*, vol. 1, no. 2, 2008, pp. 213–226.

[13] R. Gracia-Tinedo, M. Sánchez-Artigas, A. Moreno-Martınez, C. Cotes, and P. Garcıa-López, "Actively measuring personal cloud storage," in *IEEE CLOUD'13*, 2013, pp. 301–308.

[14] D. Harnik, D. Naor, and I. Segall, "Low power mode in cloud storage systems," in *IPDPS'09*, 2009, pp. 1–8.

[15] E. Pinheiro, R. Bianchini, and C. Dubnicki, "Exploiting redundancy to conserve energy in storage systems," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 34, no. 1, 2006, pp. 15–26.

[16] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage (TOS)*, vol. 4, no. 3, pp. 10:1–10:23, 2008.

[17] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "idedup: Latency-aware, inline data deduplication for primary storage," in *FAST'12*, 2012, pp. 1–14.

[18] D. Meister, J. Kaiser, and A. Brinkmann, "Block locality caching for data deduplication," in *SYSTOR'13*, 2013, pp. 15:1–15:12.

[19] Q. Zhu et. al., "Reducing energy consumption of disk storage using power-aware cache management," in *IEEE Software*, 2004, pp. 118–118.

[20] E. Pinheiro and R. Bianchini, "Energy conservation techniques for disk array-based servers," in *Supercomputing'04*, vol. 26, 2004, pp. 68–78.

[21] T. Bostoen, S. Mullender, and Y. Berbers, "Analysis of disk power management for data-center storage systems," in *e-Energy'12*, 2012, pp. 2:1–2:10.

[22] D. Harnik, O. Margalit, D. Naor, D. Sotnikov, and G. Vernik, "Estimation of deduplication ratios in large data sets," in *IEEE MSST'12*, 2012, pp. 1–11.

[23] P. Ranganathan, "Recipe for efficiency: principles of power-aware computing," *Communications of the ACM*, vol. 53, no. 4, pp. 60–67, 2010.

[24] A. Verma, R. Koller, L. Useche, and R. Rangaswami, "Srcmap: Energy proportional storage using dynamic consolidation." in *FAST'10*, 2010, pp. 267–280.

[25] E. Thereska, A. Donnelly, and D. Narayanan, "Sierra: practical power-proportionality for data center storage," in *EuroSys'11*, 2011, pp. 169–182.

[26] H. Amur et. al., "Robust and flexible power proportional storage," in *SoCC'10*, 2010, pp. 217–228.

[27] H. Huang, W. Hung, and K. G. Shin, "Fs2: dynamic data replication in free disk space for improving disk performance and energy consumption," *ACM SIGOPS Op. Sys. Review*, vol. 39, no. 5, pp. 263–276, 2005.

[28] A. Hylick, R. Sohan, A. Rice, and B. Jones, "An analysis of hard drive energy consumption," in *MASCOTS'08*, 2008, pp. 1–10.

[29] A. Wildani, E. Miller, and O. Rodeh, "Hands: A heuristically arranged non-backup in-line deduplication system," in *ICDE'13*, 2013, pp. 446–457.

[30] D. P. Helmbold, D. D. Long, T. L. Sconyers, and B. Sherrod, "Adaptive disk spin-down for mobile computers," *Mobile Networks and Applications*, vol. 5, no. 4, pp. 285–297, 2000.

[31] T. Bisson, S. A. Brandt, and D. D. Long, "Nvcache: Increasing the effectiveness of disk spin-down algorithms with caching," in *MASCOTS'06*, 2006, pp. 422–432.

[32] M. Allalouf, Y. Arbitman, M. Factor, R. I. Kat, K. Meth, and D. Naor, "Storage modeling for power estimation," in *SYSTOR'09*, 2009, pp. 3:1–3:10.

[33] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes, "Hibernator: helping disk arrays sleep through the winter," in *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5, 2005, pp. 177–190.

[34] R. T. Kaushik and M. Bhandarkar, "Greenhdfs: towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster," in *USENIX ATC'10*, 2010, pp. 1–9.

[35] R. T. Kaushik, L. Cherkasova, R. Campbell, and K. Nahrstedt, "Lightning: self-adaptive, energy-conserving, multi-zoned, commodity green cloud storage system," in *HPDC'10*, 2010, pp. 332–335.

[36] X. Ruan, A. Manzanares, S. Yin, Z. Zong, and X. Qin, "Performance evaluation of energy-efficient parallel i/o systems with write buffer disks," in *ICPP'09*, 2009, pp. 164–171.

[37] R. Koller and R. Rangaswami, "I/o deduplication: Utilizing content similarity to improve i/o performance," *ACM Transactions on Storage (TOS)*, vol. 6, no. 3, pp. 13:1–13:26, 2010.