Exploiting Redundancies and Deferred Writes to Conserve Energy in Erasure-Coded Storage Clusters

JIANZHONG HUANG and FENGHAO ZHANG, Huazhong University of Science and Technology
XIAO QIN, Auburn University
CHANGSHENG XIE, Huazhong University of Science and Technology

We present a power-efficient scheme for erasure-coded storage clusters—ECS^2—which aims to offer high energy efficiency with marginal reliability degradation. ECS^2 utilizes data redundancies and deferred writes to conserve energy. In ECS^2 parity blocks are buffered exclusively in active data nodes whereas parity nodes are placed into low-power mode. $(k+r, k)$ RS-coded ECS^2 can achieve $\lceil (r+1)/2 \rceil$-fault tolerance for k active data nodes and r-fault tolerance for all $k+r$ nodes. ECS^2 employs the following three optimizing approaches to improve the energy efficiency of storage clusters. (1) An adaptive threshold policy takes system configurations and I/O workloads into account to maximize standby time periods; (2) a selective activation policy minimizes the number of power-transitions in storage nodes; and (3) a region-based buffer policy speeds up the synchronization process by migrating parity blocks in a batch method. After implementing an ECS^2-based prototype in a Linux cluster, we evaluated its energy efficiency and performance using four different types of I/O workloads. The experimental results indicate that compared to energy-oblivious erasure-coded storage, ECS^2 can save the energy used by storage clusters up to 29.8% and 28.0% in read-intensive and write-dominated workloads when k = 6 and r = 3, respectively. The results also show that ECS^2 accomplishes high power efficiency in both normal and failed cases without noticeably affecting the I/O performance of storage clusters.

Categories and Subject Descriptors: B.4.5 [**Input/Output and Data Communications**]: Reliability, Testing, and Fault-Tolerance

General Terms: Design, Performance

Additional Key Words and Phrases: Clustered storage system, power efficiency, erasure codes, selective activation policy

ACM Reference Format:
Huang, J., Zhang, F., Qin, X., and Xie, C. 2013. Exploiting redundancies and deferred writes to conserve energy in erasure-coded storage clusters. *ACM Trans. Storage* 9, 2, Article 4 (July 2013), 29 pages.
DOI: http://dx.doi.org/10.1145/2491472.2491473

This work is supported in part by the National Basic Research Program of China under Grant No. 2011CB302303, the NSF of China under Grant No. 60933002, the National High Technology Research and Development Program of China under Grant No. 2013AA013203, and Fundamental Research Funds for the Central Universities (No. 2012QN100). X. Qin's work was supported by the U.S. National Science Foundation under Grants CCF-0845257(CAREER), CNS-0917137(CSR), and CCF-0742187(CPA).
Authors' addresses: J. Huang and F. Zhang, Wuhan National Laboratory for OptoElectronics, Huazhong University of Science and Technology, China; X. Qin, Department of Computer Science and Software Engineering, Shelby Center for Engineering Technology, Samuel Ginn College of Engineering, Auburn University, AL 36849-5347; C. Xie (corresponding author), Department of Computer, Huazhong University of Science and Technology, China; email: cs_xie@hust.edu.cn.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2013 ACM 1553-3077/2013/07-ART4 $15.00
DOI: http://dx.doi.org/10.1145/2491472.2491473

1. INTRODUCTION

Nowadays, data availability of cost-effective storage clusters is a key design issue for large-scale data centers. For example, both *Google File System* (GFS) [Ghemawat et al. 2003] and *Hadoop Distributed File System* (HDFS) [Borthakur 2008] have adopted the 3-way replication mechanism to maintain a high data survival rate. Although the N-copy mechanism is of low computational complexity and high performance, it may significantly increase overall cost (e.g., capacity and power) of storage systems when they scale up. Erasure-coded storage is a practical solution for multiple concurrent failures in storage devices [Plank 2005; Storer et al. 2008], because it can tolerate multiple failures using less storage space compared to replica-based storage [Weatherspoon and Kubiatowicz 2002]. Erasure codes have therefore been adopted in real-world storage systems. For instance, erasure codes are incorporated into Hadoop clusters in the following three studies: (1) DiskReduce stores data chunks in the background using erasure codes [Fan et al. 2009]; (2) Hadoop-EC [Zhang et al. 2010] performs erasure coding online instead of 3-way replication; (3) HDFS-RAID [Borthakur 2010] provides a RAID-like data layout for the source file in HDFS.

Apart from the two key metrics of reliable storage systems—performance and reliability—cost of building large-scale storage systems must be seriously considered. Typical cost of storage systems in data centers includes hardware and operational costs, of which energy is a significant contributor. The estimated normalized annual cost is about \$1500/TB (1TB = 10^{12}Bytes) for SATA disk storage systems in data centers [Moore et al. 2007]. For a large-scale storage system, neither hardware nor energy costs should be overlooked. Many existing fault-tolerant storage solutions like RAID are very expensive; thus, cost-effective erasure-coded storage clusters using commodity components are considered an alternative. An erasure-coded storage cluster requires fewer storage devices (e.g., disks and servers) to store redundant data, which leads to good energy efficiency. Furthermore, redundant components, which are logically envisioned as erased devices, can be powered off to save energy [Greenan et al. 2008].

Storage systems account for approximately 27% of the total energy consumed by typical data centers, which usually employ expensive RAIDs to store massive amounts of data [Zhu et al. 2004]. Existing power-aware storage solutions, especially those for RAIDs, normally conserve energy with caching schemes or powering-off disks during idle periods. Unfortunately, such energy-saving approaches in RAIDs can put only a few disks (e.g., one disk in case of RAID5) into standby mode, limiting energy-saving percentage in storage systems. For example, the power of an enterprise-level SAS disk of model *ST3146855SS* is around 14.9W in the active mode [Seagate 2011], while the power of a 5-disk EMC CX600-Series RAID is at least 370W [Yang et al. 2011].

We also measured the power consumption of both nodes and disks in our experiments (see Section 5.1). The measurements show that a node's power-consumption discrepancy between the standby and active mode is larger than that of a disk (see Table I). This is the evidence that managing power at the node level is more efficient than that at the disk level in storage clusters. Cluster-oriented energy management has been addressed to achieve high scalability and cost effectiveness of data storage [Lang and Patel 2010; Lang et al. 2010]. Thus, we focus on conserving the energy of erasure-coded storage clusters by managing the power of storage nodes rather than disks.

In an erasure-coded storage cluster, redundant data nodes (parity nodes) can be deployed to achieve fault tolerance. Importantly, redundant nodes can be placed into an energy-saving mode to reduce power consumption. Many power-aware schemes provide energy savings at the cost of performance and reliability, however, high reliability is required in modern storage systems. It is important that a system be able to

Table I. Comparison of Power Consumption (Node vs. Disk)

	Node (Server)			Disk		
State	Standby	Idle	Active	Standby	Idle	Active
Power (in Watts)	5.4	61.8	70.8∼75.6	0.8	7.2	10.2∼11.0

tolerate at least two failures [Hafner 2005; Rao et al. 2011]. Consequently, we designed and implemented an erasure-coded storage cluster: ECS^2 (Energy-Conservation Scheme in Erasure-coded Clustered Storage) that relies on data redundancies and deferred writes to conserve energy in clusters. Our ECS^2 system is a (k+r,k) erasure-coded storage cluster offering $\lceil (r+1)/2 \rceil$-fault tolerance for k active data nodes and r-fault tolerance for all $k + r$ storage nodes.

As to erasure codes, we adopted the Reed-Solomon codes (abbreviated *RS codes* [Plank et al. 1997; Reed and Solomon 1960]), where each file is encoded into k+r blocks: k data blocks and r parity blocks. The k data nodes exclusively keeping k data blocks are active at all times. Since RS codes belong to systematic codes, the read requests can be directly and efficiently serviced by the active data nodes. To shift r parity nodes to the low-power mode, all parity blocks must be temporarily buffered in the RAM of the active nodes and then flushed to the inactive parity nodes after reaching a predetermined buffer threshold.

Our major contributions made in this study are summarized in the following.

— We design a power-efficient erasure-coded storage cluster that offers high energy efficiency by exploiting redundancies and deferred, writes (see Section 3.1). Data nodes are kept active to sustain high I/O performance, while parity blocks are buffered in active nodes to keep parity nodes in the energy-saving mode.
— We develop the following three techniques to improve energy efficiency: (1) an adaptive threshold policy that maximizes the standby duration via considering the system configuration and workload (see Section 3.4), (2) a selective activation policy that minimizes the number of node power transitions by selectively activating standby nodes (see Section 3.5), and (3) a region-based buffer policy that speeds up synchronization by migrating parity data in a batch manner (see Section 3.3).
— We build a model (see Section 4.1) to evaluate the impact of ECS^2 on system reliability. We also construct a model (see Section 4.3) to predict the power consumption of the ECS^2-based system.
— We comprehensively evaluate the performance and energy efficiency of ECS^2 under real-world workloads. Our results show that compared with energy-oblivious erasure-coded storage, the ECS^2-based system improves energy efficiency by about 30% and 28% for read-intensive and write-dominated applications without noticeably affecting I/O performance when k = 6 and r = 3, respectively.

The remainder of the article is organized as follows. We briefly describe the background and related work in Section 2. Section 3 details the design of ECS^2. Section 4 presents the models of ECS^2's reliability, buffer thresholds, and energy consumption. We describe our experimental methods and results in Section 5. Further discussions can be found in Section 6. Section 7 concludes the article.

2. BACKGROUND AND RELATED WORK

Our work described in this article is related to erasure codes and power-aware storage systems.

2.1. Erasure Coding

Fault-tolerant storage systems offer high data reliability through redundant storage devices. For example, in a (k + r, k) erasure-coded scheme, each data set is divided into k blocks, and $k + r$ encoded blocks are stored on separate storage nodes. Given a (k + r, k) erasure code that is *maximum distance separable* (MDS), one can recover lost data from any k blocks. Reed-Solomon (RS) codes, which have the MDS property, are widely adopted in the field of erasure correcting. In coding theory, RS codes usually utilize a linear code construction using finite-field arithmetic (a.k.a., Galois Field arithmetic). For example, in Vandermonde RS coding, the addition operation over a Galois Field is equivalent to bitwise exclusive-or (XOR), and multiplication is relatively complex (e.g., it can be implemented via the logarithm and inverse logarithm).

In (n = k + r, k) RS coding, one can create n result words by multiplying a k $\times$ n *Generator Matrix* by k source words. The generator matrix is a k $\times$ k *Identity Matrix* followed by a k $\times$ r *Redundancy Matrix*; the identity matrix causes RS codes to have a systematic form. The generator matrix can be constructed from a *Vandermonde Matrix*. Any k $\times$ k submatrix in a k $\times$ n Vandermonde matrix is invertible, allowing all elements to be reconstructed from any k result words, viz. the MDS property [Plank et al. 2009]. Because each element of the $(k + 1)^{th}$ column in the Vandermonde matrix is 1, the first encoded parity word is the XOR sum of k source words. Hence, it is highly efficient to recover a faulty data node using the first parity node in an RS-based storage cluster.

2.2. Power-Aware Storage Systems

In the last few years, there has been a great deal of work focusing on energy conservation techniques in storage systems. For example, caching mechanisms are used to save power in storage systems, because the mechanisms can direct major I/O accesses to a small set of disks while allowing other disks to stay in the low-power mode [Ganesh et al. 2007; Pinheiro and Bianchini 2004; Zhu et al. 2004]; on the other hand, the deferred update (a.k.a., buffered write) methods use a persistent device like NVRAM [Pinheiro et al. 2006; Yao and Wang 2006], Flash Drive [Chen et al. 2006] or log volume [Narayanan et al. 2008] to temporarily handle writes, thereby avoiding spinning up disks in the low power mode. Table II summarizes the major differences between our proposed ECS^2 and the existing energy-saving techniques for storage systems.

Colarelli and Grunwald designed the MAID [Colarelli and Grunwald 2002] storage system, in which recently used data blocks are cached in a subset of disks while the other disks are spun down to save energy. MAID offers significant energy savings for archival storage systems equipped with thousands of disks. Pinheiro and Bianchini proposed PDC [Pinheiro and Bianchini 2004], in which frequently accessed data is migrated to a subset of disks so that other disks in the RAID-based network servers can be put into low-power mode.

Wang et al. [2007] developed ERAID and EERAID [Li and Wang 2004] systems to save energy in RAIDs by exploiting redundant data. ERAID contains a request-redirecting module for RAID1 to spin down disks in a mirror group. A transformable read policy is introduced in EERAID, which transforms disk requests from active disks in RAID5.

Yao and Wang [2006] proposed a two-level redundancy-based I/O cache (storage cache and parity cache) architecture called RIMAC, which can significantly improve both energy efficiency and performance. RIMAC uses NVRAM as parity cache; UPS-supported RAM in RIMAC serves as storage cache. RIMAC exploits inherent redundancy of RAID5 to perform power-aware request transformations for reads and writes. Yao and Wang implemented two power-aware read request transformation

Table II. A Comparison of Existing Power-Aware Storage Systems

Power-efficient Scheme	Deactivated Unit	Transient Place of Write Data	Application or Reliability Scenario
MAID	Disk drive	Cache drive	Archival storage
PDC	Disk drive	Cache drive(Concentrating hot data)	RAID-based network servers
EERAID,ERAID	Disk drive	NVRAM,(and active disks for ERAID)	RAID1, RAID5
RIMAC	Disk drive	Both Storage and Controller Cache	RAID5
PARAID	Disk drive	Spare regions at active disks	RAID (specifically RAID0 and RAID5)
Diverted Access(DIV)	Disk drive	NVRAM	Replication-based, RAID, or Erasure code
Power-aware Coding	Disk drive	NO, Parity updates never be staged	Erasure code (specifically non-MDS code)
Pergamum	Disk drive	NVRAM(holding metadata)	Archival Storage
Chained Declustering	Entire Node	NO, Directly update the active replica	Data-intensive Cluster (2X Replication)
Hadoop Covering Set	Entire Node	NO. Always reading for data analysis	Read-intensive Cluster (3X Replication)
Rabbit	Entire Node	Disk at active nodes(Write offloading)	N-way Replication
ECS^2 (in this paper)	**Entire Node**	**DRAM in active nodes**	**MDS code (specifically RS code)**

schemes and a power-aware write request transformation policy in RIMAC. Unlike RIMAC, our ECS^2 makes use of active data nodes to directly serve read requests; both new and updated parity blocks are buffered in preallocated DRAM of the active data nodes.

Weddle et al. [2007] developed PARAID, which utilizes skewed striping patterns to adapt to different load levels by varying the number of powered disks. In PARAID, disks can be organized into overlapping sets of RAIDs; each set can serve all requests via either data blocks or their replicas. PARAID exploits unused disk space to replicate data blocks in order to form the overlapping sets of RAIDs.

Greenan et al. [2008] designed 'power-aware coding', a power-efficient erasure-coded storage system. In this system, reads are completed via a partial or whole-stripe reconstruction. Our ECS^2 system is different from power-aware coding, because writes in power-aware coding are served by deterministic disk activations. Our system conserves energy through deferred writes.

Pinheiro et al. [2006] introduced Diverted Accesses (DIV)—a technique that relies on redundant disks to save energy in disk systems. Redundant disks are placed in low-power mode under light or moderate loads. They are activated when the I/O load becomes high or when any disk in the systems fails. Redundant data is buffered in NVRAM and propagated to redundant disks when they are activated.

Storer et al. [2008] designed a distributed network of intelligent, disk-based, storage appliances (*Pergamum*). In this system, NVRAM is integrated into each node to store data signatures, metadata, and other small data items, allowing deferred writes, metadata requests, and interdisk data verification to be performed while a disk is powered off. *Pergamum* provides reliability using two levels of Reed Solomon encoding: intradisk redundancy and interdisk redundancy.

Many energy-saving techniques achieve power savings through hardware configurations (e.g., Hibernator [Zhu et al. 2005]) and data layout (e.g., DIV [Pinheiro et al. 2006], ERAID [Wang et al. 2007], and PARAID [Weddle et al. 2007]). Existing energy-efficient reliable storage schemes at the disk level have the following major limitations: (1) energy dissipation in disks accounts for a small fraction of the total system power; (2) the number of spun-down disks is limited by the encoding pattern; (3) some

XOR-based codes (e.g., EVENODD [Blaum et al. 1995] and RDP [Corbett et al. 2004]) generate parity data using a diagonal mode that potentially lengthens the recovery time.

Apart from disk-level energy-conservation schemes, there are a few node-level power-saving solutions. For example, Leverich and Kozyrakis [2010] designed a mechanism to recast the data layout to power down some nodes in Hadoop clusters; Lang et al. [2010] leveraged the Chain Declustering-based replication strategy to take nodes off line in the event of low system utilization.

Amur et al. [2010] designed a power-proportional distributed file system, called *Rabbit*, which arranges data-layout to minimize the number of powered-up nodes in case of any primary replica failure. PowerNap is an energy-saving solution that can rapidly transition blade servers between ultra-low power state and active state [Meisner et al. 2009]. Idle power consumption usually accounts for 50∼60% of the peak server power; three main hardware components (CPU, RAM, and system-board components) contribute to energy waste [Meisner et al. 2009; Tsirogiannis et al. 2010]. Table VI on Page 19 lists the power of a real-world storage node on three states (active, idle, and standby).[1] The power consumption of nodes is markedly more than that of disks; therefore, we are motivated in this study to conserve energy by transitioning storage nodes to a near-zero-power state.

Our DRAM-based ECS^2 is ultimately more cost-effective than the existing NVRAM-based solutions. Because DRAM is cheaper than NVRAM, ECS^2 makes it affordable to increase DRAM space to buffer an enormous number of parity blocks. At first glance, parity blocks buffered in the DRAM of active data nodes may be lost in the event of power outage. Nevertheless, the lost parity blocks can be generated from data blocks stored in the disks of the active nodes after power recovery.

Our proposed ECS^2 system is different from the aforementioned energy-efficient storage systems (see Table II), because ours—tailored for node-level erasure-coded storage clusters—utilizes data redundancies and deferred writes to conserve energy. Parity nodes in ECS^2 are logically envisioned as "erased ones," which are switched to the ultra-low-power mode. A group of active data nodes buffer parity blocks while serving incoming data requests, and therefore ECS^2 can achieve high energy efficiency and reliability without adversely affecting I/O performance.

3. DESIGN OF ECS^2

In this section, we present the design of ECS^2, which aims to offer high energy-efficiency, reliability, and I/O performance.

3.1. Overview

Unlike most existing techniques that save energy at the disk level, ECS^2 offers an energy conservation solution at the storage-node level; all components of storage nodes, including RAM, CPU, Disk, and so on, are sent to the low-power mode. In this study, we first build a (k + r, k) RS-coded storage cluster, which has $k + r$ storage nodes, each data or parity block is stored to an individual node, and data nodes $\{DN_1, DN_2, ..., DN_k\}$ and parity nodes $\{PN_1, PN_2, ..., PN_r\}$ are segregated. All data nodes are kept active to maintain high I/O performance, and all parity blocks are buffered in the active nodes before being synchronized to corresponding parity nodes that are in the low-power mode. Figure 1 illustrates such a power-efficient framework for RS-coded storage.

[1] In this article the term "standby" refers to the hibernate mode; and a node can be remotely awakened via *magic packet*.

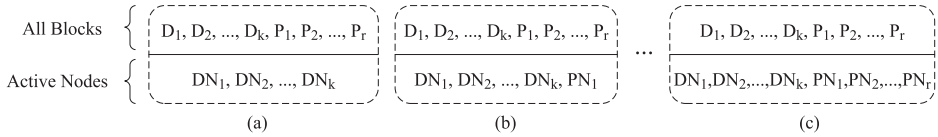


Fig. 1. A power-efficient framework that exploits redundancy of RS-coded storage cluster. All parity blocks are buffered at the active nodes so the remaining parity nodes can stay in the low-power mode. (a) no parity node is active; (b) a parity node is active; (c) all parity nodes are active.

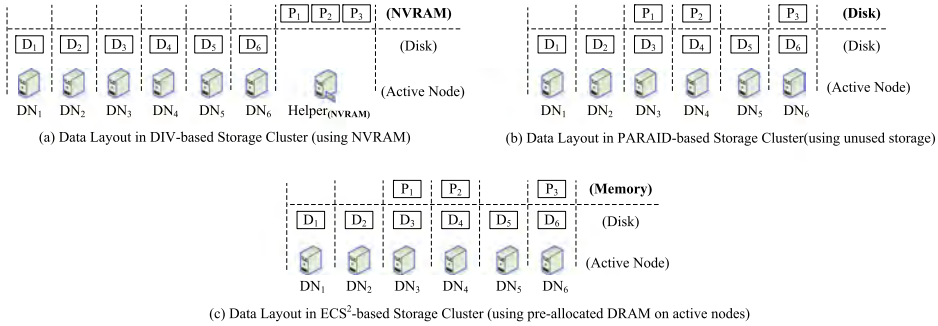


Fig. 2. Three parity-block layouts of the (9,6) RS-coded storage cluster, where all parity nodes are in the low-power mode. (a) Parity blocks are kept in the NVRAM of a helper node (similar to the DIV scheme); (b) Parity blocks are stored in unused disks on the data nodes (similar to the PARAID scheme); (c) our ECS^2 scheme stores parity blocks to preallocated memory on the data nodes.

Figure 1(a) illustrates a case where energy efficiency is maximized by transitioning all r parity nodes into low-power mode at the cost of reliability. Figure 1(c) represents a power-oblivious case, in which the storage cluster achieves high reliability without saving power. In the case of Figure 1(a), the parity blocks can be buffered in NVRAM (see Figure 2(a)) or stored in the active data nodes (see Figure 2(b)). The data layout using NVRAM can be regarded as the deployment of DIV [Pinheiro et al. 2006] or EERAID [Li and Wang 2004] on a storage cluster. The layout scheme keeps updated parity blocks at the disks of active nodes and is similar to PARAID [Weddle et al. 2007].

Note that PARAID relies on spare space on active disks to replicate data blocks. Unlike PARAID, ECS^2 buffers parity blocks in main memory of active nodes, allowing all r parity nodes to be placed into the low-power mode (see Figure 2(c)). To reinforce the fault-tolerance of an active group, we propose a data-layout scheme that enables the active group to tolerate $\lceil (r+1)/2 \rceil$ concurrent node failures. ECS^2 deploys an adaptive threshold strategy to selectively activate inactive nodes to optimize power efficiency.

3.2. Data Layout Scheme

Active data nodes that exclusively buffer r parity blocks can provide $\lfloor r/2 \rfloor$-fault tolerance. One or two extra parity blocks ($\{Q_1\}$ or $\{Q_1, Q_2\}$) can make the active group tolerate $\lceil (r+1)/2 \rceil$ concurrent node failures. A matrix-based construction scheme for extra parity blocks is described in Appendix 7, where (k + r + Δr, k) RS codes are used to generate the extra parity blocks and $\Delta r \in \{1, 2\}$. This matrix-based construction scheme for extra parity blocks is CPU-expensive, so we adopt an efficient XOR-based construction method to enable the active group 2-fault tolerant when r is set to 2 or 3.

If r equals 2, two extra parity blocks $\{Q_1, Q_2\}$ must be generated to tolerate two concurrent node failures, e.g. $Q_1 = P_1 \oplus D[P_2]$ and $Q_2 = D[P_1] \oplus D[P_2]$, $D[P_j]$ denotes the data block stored in the node that buffers parity block P_j.

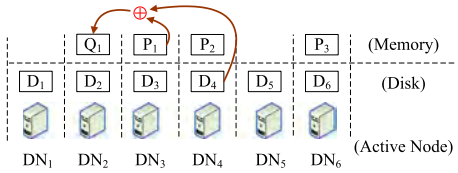


Fig. 3. The data layout of an active group that can tolerate 2 faults when $r = 3$. An extra parity block $Q_1 = P_1 \oplus D[P_2] = P_1 \oplus D_4$ is exclusively buffered in an active node.

Table III. Available Blocks in Case of Double Node Failures

1st Failed Node	2nd Failed Node	Available Blocks	Auxiliary Block
DN_1 or DN_5	There are still 9 surviving blocks, so can tolerate one node failure		
DN_2	DN_3	$D_1, D_4, D_5, D_6, P_2, P_3$	-
	DN_4	$D_1, D_3, D_5, D_6, P_1, P_3$	-
	DN_6	$D_1, D_2, D_3, D_4, D_5, P_1$	-
DN_3	DN_4	$D_1, D_2, D_5, D_6, Q_1, P_3$	D_3^*
	DN_6	$D_1, D_2, D_4, D_5, Q_1, P_2$	P_1^*
DN_4	DN_6	$D_1, D_2, D_3, D_5, Q_1, P_1$	D_4^*

D_3^*: $D_3 = P_1 \oplus D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_6 = Q_1 \oplus D_1 \oplus D_2 \oplus D_5 \oplus D_6$
P_1^*: $P_1 = Q_1 \oplus D_4$; D_4^*: $D_4 = P_1 \oplus Q_1$
See Figure 3 for the data layout.

When r is set to 3, only an extra parity block, Q_1, needs to be generated, where $Q_1 = P_1 \oplus D[P_2]$, or $Q_1 = P_1 \oplus D[P_3]$. For space reasons, we limit our study to the case of r = 3. Figure 3 shows a data layout of ECS^2 using the (9,6) RS codes.

We use an exhaustive method to quantify the fault-tolerance of an active group. Table III illustrates available blocks in the case of two-node failures. We observe from Table III that the active group can tolerate two-node failures using the data layout plotted in Figure 3. In addition, the active group can tolerate two-node failures with a redundancy configuration, where $k \geq r + \Delta r$ ($k \geq 4$ when r = 3).

3.3. Power-Aware Access Policies

Since ECS^2 adopts the RS coding scheme, the features of RS codes must be exploited to improve I/O performance and data availability. In this section, we describe the access policies that allow ECS^2 to avoid unnecessary power-state transitions in the inactive storage nodes.

3.3.1. The Read Policy. Responding to read requests with an active data node does not introduce any extra decoding overhead, because RS codes are systematic in nature. When it comes to read requests, parity blocks stored in main memory do not consume any disk bandwidth; thus, the read performance of the ECS^2 system is arguably close to that of an energy-oblivious erasure-coded storage cluster.

In case of storage node failures, it is critical and challenging to make trade-offs between performance and reliability. Because the MTTDL value (Mean Time To Data Loss) of a reliable storage system is inversely proportional to the r-th power of recovery time [Rao et al. 2011] (r is the number of failures tolerated by the storage system), speeding up the recovery process can substantially enlarge the MTTDL value. The following two methods can potentially accelerate the reconstruction procedure by exploiting the structural feature of RS codes. First, if inactive parity node PN_1 is activated in response to a data node failure, then the node will join other active data nodes to

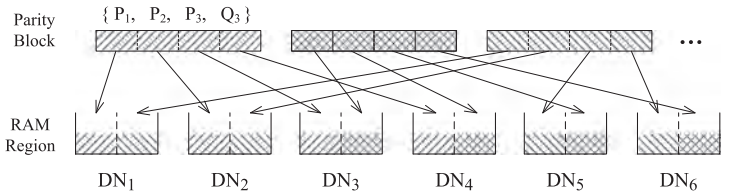


Fig. 4. A data layout of buffer regions among the active group.

quickly rebuild a replacement node by reconstructing data stored on the failed node with expression "$P_1 = D_1 \oplus D_2 \oplus \cdots \oplus D_k$". Second, when a node failure occurs, an inactive parity node will be activated according to the Selective Node Activation Policy (see Section 3.5). Such a node activation introduces disk-spin-up overhead, ultimately increases user response times, and delays the data recovery procedure. To amortize such overhead, our ECS^2 continues to serve read requests using surviving active data nodes. Such a continuous service is feasible, because parity blocks buffered in nonfailed nodes may help to rebuild data requests directed to the failed data node. Furthermore, data rebuilt for read requests can be stored on the corresponding replacement node to speed up the reconstruction process. If two concurrent failures occur, two inactive parity nodes will be powered on, and these approaches still work.

In the process of synchronizing parity blocks among active and inactive nodes, I/O bandwidths of data nodes are shared between user I/O requests and the data synchronization module in ECS^2. To reserve a sufficient amount of I/O bandwidth for the synchronization process, user I/O requests can be asynchronously processed (a.k.a., *Asynchronous I/O*) because disk I/O is time-consuming.

3.3.2. The Write Policy. Minimizing the number of power-state transitions can substantially improve the energy efficiency of storage clusters. The goal of the write policy in ECS^2 is to avoid unnecessary power transitions of inactive nodes. All parity blocks including extra ones are buffered on a set of active data nodes. Usually, distributing parity blocks across the active nodes has an impact on the synchronization performance. For example, if all parity blocks are evenly and randomly distributed among the active nodes, then their localities are ignored. In the ECS^2 system, we preallocate a set of memory regions on certain data nodes, where each region is allocated to a specific parity block-*region-based buffer policy*, thus constituting a mapping between a RAM region and a parity node (see Figure 4).

Algorithm 1 shows the pseudocode for the implementation of the write policy, under which write operations of file data (e.g., $\{D_1, D_2, ..., D_k, P_1, P_2, ..., P_r\}$) are always synchronized to maintain data consistency. Buffering parity blocks to active nodes can shorten write response time by shortening the synchronization chain. On the other hand, this region-based buffer policy enables the buffered parity blocks to be migrated to the activated parity nodes in a batch manner, which speeds up the synchronization. Small writes can benefit more than large writes from such a batch synchronization process.

3.3.3. The Update Policy. When any data block is modified, its parity block also must be updated accordingly, however it is not energy efficient to wake up the standby parity nodes at this time. To address this problem, we apply the read-modify-write parity update scheme [Aguilera et al. 2005; Hafner et al. 2010], where existing parity blocks are not immediately accessed. This keeps parity nodes in the standby mode for an increased period of time. Under this update policy, active data nodes buffer the differences between old data blocks and new data blocks. When standby parity nodes are

Algorithm 1: The Write Algorithm. See Figure 3 for the Data Layout

```
/* Let S_data be data node set, S_data = {DN_1, DN_2, ..., DN_k}; Let R_i,j be a buffer region at data node DN_i for
parity block P_j.                                                                                              */
/* Refer to P_r+1 as the extra parity block Q_1.                                                               */
Input: D_i, P_j   (D_i is i^th data block and P_j is j^th parity block)

// Dispatch the data blocks;
foreach D_i ∈ {D_1, D_2, ..., D_k} do
    Store D_i to the disk of data node DN_i;
end

// Buffer the all parity blocks among data nodes;
foreach P_j ∈ {P_1, P_2, ..., P_r, P_r+1} do
    if R_i,j has free space then
        Buffer P_j to the region R_i,j;
    else
        Insert node DN_i to the node set S_data;
        Select a new node DN_i from node set S_data;
        Remove node DN_i from node set S_data;
        Allocate a new buffer region R_i,j at data node DN_i;
    end
end
```

activated, the active data nodes push the differences to the activated parity nodes in parallel. These differences along with existing parity blocks are used by each parity node to create new parity blocks.

3.4. Adaptive Buffer Threshold

Since I/O load may fluctuate due to variation in application characteristics and daily cycles, erasure-coded storage clusters may encounter unexpected events like bursts of traffic and TCP throughput collapse [Phanishayee et al. 2008]. To address these issues, we incorporate an adaptive buffer threshold $V_{\text{threshold}}$ into the ECS^2 system to dynamically manage buffers according to changing I/O workload conditions. In particular, we set an extra deviation $V_{\text{deviation}}$ for buffer threshold $V_{\text{threshold}}$. Hence, an inactive parity node can be activated when the buffer utilization reaches $(V_{\text{threshold}} - V_{\text{deviation}})$ in the case of heavy I/O loads.

We can allocate the buffer according to the I/O performance of an individual active node. For example, weight w_i is assigned to the i^{th} active data node; the sum of all the weights equals 1 ($\sum_{i=1}^{k} w_i$=1). The i^{th} data node offers the buffer whose size is $S_{\text{buf,i}}$, which is derived from weight w_i and the total buffer size S_{buf} ($S_{\text{buf,i}} = w_i \times S_{\text{buf}}$). On the other hand, We develop a buffer threshold model (see Section 4.2), which suggests that the value of $V_{\text{threshold}}$ can be set to $1\text{-}R_{\text{req\_w}}T_{\text{up}}S_{\text{req\_buf}}(r+1)/S_{\text{buf}}$. Table IV summarizes the notation used in the model, where $R_{\text{req\_w}}$ is the average arrival ratio of write requests issued to active data nodes; it takes T_{up} to power up an inactive node. The buffer threshold policy implemented in ECS^2 is adaptive, because both the buffer size and threshold are judiciously adjusted based on node configurations and workload characteristics.

Although one can set various buffer thresholds for different active nodes, customized thresholds may cause three problems: (1) The overall synchronization time is determined by the highest buffer threshold; the highest threshold delays deactivating time points of parity nodes. (2) Activating time points of inactive nodes are affected by the lowest threshold, which may increase activation frequency. (3) Managing memory resources for the customized thresholds is nontrivial.

To solve the aforementioned three problems, we design a simple yet effective threshold-setting policy, in which all active nodes uniformly deploy an upper bound of buffer threshold determined by Formula (5).

3.5. The Selective Node Activation Policy

We design a *selective activation policy* to minimize the overall number of node-power transitions. It is expensive to activate all inactive nodes in response to an active-node failure, because an activated parity node can join the active group to serve both foreground user I/Os and background reconstruction I/Os. We propose a node-activation algorithm to optimize both power efficiency and cost effectiveness. Algorithm 2 details the *selective activation policy* adopted in our ECS^2 system. Thus, if one or two node failures occur, one or two inactive parity nodes will be activated respectively. Then, corresponding replacement nodes can be reconstructed.

Algorithm 2: The Node Activation Algorithm in (k + r, k) RS-based ECS^2-based System

```
/* Assume there are N_failed failures in active group, and N_failed ≤ ⌈(r + 1)/2⌉ */

if Buffer utilization does not reach threshold V_threshold then
    switch N_failed do
        case One
            Prepare a replacement node;
            Activate the parity node PN_1; break;
        endsw
        case Two
            Prepare two replacement nodes;
            Activate two parity nodes (including PN_1); break;
        endsw
        otherwise
            Prepare N_failed replacement nodes;
            Activate all r parity nodes; break;
        endsw
    endsw

    // Using the surviving data nodes;
    Recover parity blocks and associated data blocks;
    Buffer the recovered parity blocks to replacement nodes(s);
    Store the recovered data blocks to replacement nodes(s)
    // Using the surviving data nodes and parity node(s);
    Reconstruct the disk data for replacement node(s);
else
    Activate all inactive parity nodes;
    Synchronize the buffered parity blocks.
end
```

All inactive parity nodes will be activated when buffer utilization reaches a specified threshold. After the parity nodes are activated, buffered parity blocks are synchronized to the parity nodes $\{PN_1, PN_2, ..., PN_r\}$ via TCP connections. Duration $(T/N_{transit})$ between two consecutive state changes (active/inactive) is determined by several factors: average arrival ratio R_{req_w} of write requests, average write request size S_{req_w}, total buffer size S_{buf}, and number N_p of parity nodes (see Equation (10)).

If the buffer size of all active nodes is smaller than the amount of parity data produced by writes during the node-state-transitioning period, then the parity nodes could not be placed into the standby mode; parity-buffering operation would introduce extra write latency; and new write requests could not be served immediately during the parity-synchronization process. Such a deadlock case accomplishes no power efficiency and degrades write performance. In Section 4.2, we model the lower bound of the buffer

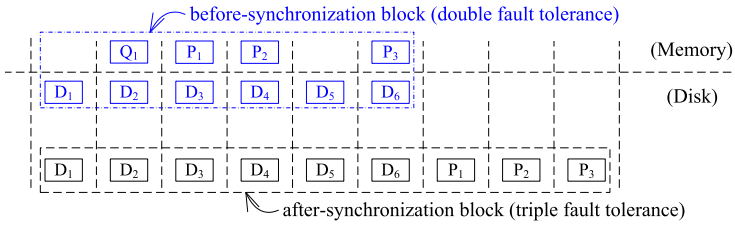


Fig. 5. Data layouts of both before-synchronization and after-synchronization blocks.

size (see S_{buf} in Equation (4)) as well as the upper/lower bounds of buffer utilization threshold (see $V_{\text{threshold}}$ in Equations (5) and (6)).

With the threshold-based synchronization policy in place, all requests can be served by existing active nodes prior to the synchronization phase; therefore, this synchronization approach defers the time at which inactive nodes are powered up. In addition to the synchronization policy, the selective activation policy helps to prevent unnecessary power-state transitions in storage nodes, thereby improving the power efficiency of storage clusters.

3.6. Buffering Parity Blocks

Unlike the existing power-aware schemes (e.g., PARAID [Weddle et al. 2007] and ERAID [Wang et al. 2007]) that keep redundant data in active disks, our ECS^2 buffers parity blocks in the memory of active nodes. This strategy allows ECS^2 to achieve high I/O performance because of two reasons. First, writing parity blocks to disks of active nodes (write-through) consumes extra disk bandwidth and increases write response time. Second, migrating parity blocks from disks in active nodes slows down the synchronization process and compromises the energy efficiency.

In the (k + 3, k)RS-coded ECS^2, buffered blocks (a.k.a., *before-synchronization blocks*) in the active group can tolerate double node failures. Written blocks (a.k.a., *after-synchronization blocks*) among all $k + r$ nodes can be reconstructed when any triple nodes fail. Figure 5 shows the data layouts of both before-synchronization and after-synchronization blocks. Although the MTTDL of an entire system is higher than that of an active group (see Section 4.1 for a quantitative reliability analysis), the data-loss probability of *before-synchronization blocks* is lower than that of *after-synchronization blocks*; on the other hand, the chance of having three concurrent failures is extremely low ($MTTDL_{\text{Active-Group}} \approx 7.54 \times 10^5$ years when k = 6). We conclude that our ECS^2 achieves power efficiency with marginal degradation in reliability.

4. MODELS

In this section, we describe models for system reliability, buffer threshold, and power consumption respectively. For simplicity without loss of generality, we consider homogeneous storage clusters where all nodes are identical.

4.1. Reliability Model

We evaluate the impact of our ECS^2 on system reliability using a reliability model. Given k available blocks, it is able to rebuild any failed blocks simultaneously. We make use of traditional Markov modeling with reconstruction-in-parallel [Hafner and Rao 2006]). Let us assume that nodes in the ECS^2 system are enclosed entities, where all disks attached to each node are organized in the form of internal RAIDs [Rao et al.

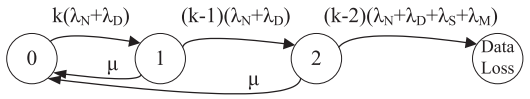


Fig. 6. The Markov model for an ECS^2-based active group with the concurrent reconstruction policy in which r is 3.

2011]. We build a high-level Markov model for erasure codes among data nodes in ECS^2. After analyzing the data flow of ECS^2, we observe that only the failure of an internal RAID can result in permanent data loss; RAM data error or node outage simply leads to transient loss of data. Therefore, we construct a reliability model of *MDS* codes for permanent data loss (see Figure 6). In this model, λ_N is the failure rate of a node; λ_D and λ_d represent the failure rate of an internal RAID and that of disks, respectively; λ_S is the sector error rate during the restripe of the internal RAID; λ_M is the RAM error rate during reconstruction; μ is the recovery rate at which the ECS^2 system can be recovered back to the zero loss state. Typically, the internal RAID's failure rate (λ_D) is much smaller than the recovery rate μ ($\lambda_D \ll \mu$).

In ECS^2, k data nodes constitute the active group that can tolerate two concurrent failures when r is set to 3. Figure 6 shows the Markov model for the active nodes with internal RAIDs. The MTTDL of the active group can be calculated by Equation (1).

$$MTTDL_{\text{Active-Group}} \approx \frac{\mu^2}{k(k-1)(k-2)(\lambda_N+\lambda_D)^2(\lambda_N+\lambda_D+\lambda_S+\lambda_M)} \tag{1}$$

Similarly, the entire system's MTTDL can be expressed as Equation (2),

$$MTTDL_{\text{Entire-System}} \approx \frac{\mu^2}{(k+3)(k+2)(k+1)k(\lambda_N+\lambda_D)^3(\lambda_N+\lambda_D+\lambda_S+\lambda_M)} \tag{2}$$

Let us consider a simple case where each node has one disk, so the internal RAID failure rate equals the disk failure rate ($\lambda_D = \lambda_d$). Suppose that the mean time to failure of a node, $MTTF_{\text{node}}$, follows an exponential distribution. The node failure rate, λ_N, is inversely proportional the mean time to failure $MTTF_{\text{node}}$ ($\lambda_N = 1/MTTF_{\text{node}}$). The node mean time to failure equals approximately one fifth of the disk mean time to failure ($MTTF_{\text{node}} \approx 0.2 \times MTTF_{\text{disk}} = 100{,}000$hrs), because the annual failure rate (AFR) of disks is about 20% of the AFR of the storage node [Jiang et al. 2008], and $MTTF_{\text{disk}} \approx 500{,}000$hrs [Hafner and Rao 2006]. Recovery rate μ is inversely proportional to the mean time to repair of nodes $MTTR_{\text{node}}$; thus, we have $\mu = 1/MTTR_{\text{node}}$. The mean time to repair $MTTR_{\text{node}}$ is approximately equal to 10 hours, because a storage node with 1TB data can be reconstructed with a recovery rate of 100GB/hour [Xin et al. 2003].

We neglect the RAM-error rate λ_M in our model because the error rate of DDR2 SDRAM is extremely low (e.g., one per 100 years) [Samsung 2008]. Latent sector errors affect system reliability during the reconstruction process [Bairavasundaram et al. 2007]. We set λ_S to 1/13245 hours [Storer et al. 2008]. The active group MTTDL ($MTTDL_{\text{Active-Group}}$) is approximated as 7.54×10^5 years when k is 6 (see Equation (1)), indicating that the probability of having triple-node failures is extremely low.

The probability of data loss in a time period T can be calculated as follows (see Holland et al. [1994] for the details of computing this probability).

$$P_{\text{T}} = 1.0 - e^{-T/MTTDL} \tag{3}$$

Table IV. Notation

Symbol	Definition
N_d	number of the data nodes
N_p	number of the parity nodes
S_{strip}	size of the strip unit
S_{req_w}	average size of the write request
S_{req_buf}	size of parity in buffer at an active data node, for write request of size S_{req_w}
R_{req_w}	average arrival ratio of the write requests of the system
S_{buf}	total buffer size of all active data nodes
T_{up}	average activating time of an inactive parity node
T_{down}	average deactivating time of an active node
T_{full}	time of full transition, $T_{full} = T_{up} + T_{down}$
$T_{standby}$	standby time of an inactive node
T_{syn}	time of synchronization
E_{up}	average activating energy (in Joules) of an inactive node
E_{down}	average deactivating energy (in Joules) of an active node
E_{full}	energy (in Joules) of full transition, $E_{full} = E_{up} + E_{down}$
$P_{standby}$	average power (in Watts) of a standby node
P_{power}	average power (in Watts) of a powered-on node
$P_{(active,T)}$	power consumption (in Watts) of active nodes at time T
$P_{(inactive,T)}$	power consumption (in Watts) of inactive nodes at time T
B_{node}	average synchronization bandwidth
$V_{threshold}$	threshold of buffer utilization, $0 < V_{threshold} < 1$

If the lifetime of a *before-synchronization block* is 1 hour and $MTTDL_{Active-group}$ is 7.54×10^5 years, the probability of losing data is about 1.54×10^{-10} for the *before-synchronization block* (see Equation (3). Let the lifetime of a *after-synchronization block* be 10 years—an optimistic estimate of RAID's lifetime [Holland et al. 1994]), the probability of losing data is around 3.34×10^{-8} for the *after-synchronization block*. In this estimation, the MTTDL of the entire system ($k + r$ nodes) is 2.99×10^8 years when using the aforementioned assumptions.

This reliability model confirms that (k + 3, k) RS-coded ECS^2 can provide sufficient reliability for both the active group (k nodes) and storage cluster ($k + r$ nodes).

4.2. Buffer Threshold Model

We summarized the notation used throughout this article in Table IV.

During a standby and transitioning period, the buffers of all active nodes must be sufficiently large to accommodate all parity blocks generated by write requests. Consequently, the lower bound S_{buf} of the buffer size can be calculated by Equation (4), where the right-hand side represents the amount of data produced by writes during the standby and transitioning period. The lower bound S_{buf} is a product of the arrival rate of writes, the standby and transitioning time, the number of parity blocks, and parity block size.

$$S_{buf} \geq R_{req\_w}(T_{full} + T_{standby})S_{req\_buf}(N_p + 1) \tag{4}$$

To increase the standby time of inactive parity nodes, we have to buffer a large number of parity blocks to reduce power-transition penalties. Increasing the number of buffered parity blocks can compromise the reliability of storage clusters; therefore, we need to make a good trade-off between energy efficiency and reliability by determining the most appropriate buffer size.

Inactive nodes will be powered up if the buffer utilization reaches the threshold $V_{\text{threshold}}$; thus, $(1 - V_{\text{threshold}}) * 100\%$ is the percentage of the buffer space available to handle writes during the course of the powering-up process. Note that the buffer in active nodes should be large enough to contain redundant data even when inactive nodes are in the power-transition process. In our proposed data layout scheme (see Section 3.2), the ratio of parity blocks to data blocks is $(N_{\text{p}} + 1)/N_{\text{d}}$ before the synchronization process is triggered. Thus, the data amount of parity blocks to be buffered during the activating time T_{up} of an inactive node is $R_{\text{req\_w}} T_{\text{up}} S_{\text{req\_buf}} (N_{\text{p}} + 1)$. Since the data amount of the buffered parity blocks must be smaller than the available buffer space ($(1 - V_{\text{threshold}}) S_{buf}$), the upper bound of the threshold $V_{\text{threshold}}$ can be given via the following expression.

$$V_{\text{threshold}} \leq 1 - R_{\text{req\_w}} T_{\text{up}} S_{\text{req\_buf}} (N_{\text{p}} + 1)/S_{\text{buf}} \tag{5}$$

Let P_{standby} (in Watts) be the power of a storage node in the standby mode, and let P_{power} be the average power of an active storage node in the powered-on mode. E_{up} is the energy overhead caused by transitioning a storage node from the standby mode to the powered-on mode; whereas E_{down} is the energy overhead incurred by placing the powered-on node into the standby mode. A full transition from the standby to the powered-on mode and back consumes energy $E_{\text{full}} = E_{\text{up}} + E_{\text{down}}$.

According to the break-even strategy, conserved energy yielded during the standby period must be equal to the energy consumed by a full transition. We derive the minimal standby duration as $E_{\text{full}}/(P_{\text{power}}\text{-}P_{\text{standby}})$, which suggests the minimal standby time that can offset the energy overheads caused by power transitions. In order to achieve energy savings, $V_{\text{threshold}}$ of the buffer space should be large enough to contain parity blocks during any powering-down and standby period of a parity node. The lower bound of the threshold, $V_{\text{threshold}}$, is given as Expression (6), where the right-hand side represents the amount of data to be buffered during the powering-down and standby period.

$$V_{\text{threshold}} \geq R_{\text{req\_w}} (T_{\text{standby}} + T_{\text{down}}) S_{\text{req\_buf}} (N_{\text{p}} + 1)/S_{\text{buf}} \tag{6}$$

4.3. Power Consumption Model

In this section, we build a power consumption model for storage clusters by incorporating the characteristics of storage nodes and fault-tolerance features.

4.3.1. Energy Consumption of ECS^2. The total energy consumption $E_{(\text{ECS}^2,\text{T})}$ (measured in Joules) of the ECS^2 system is the sum of the energy ($E_{(\text{active,T})}$, see Equation (8)) consumed by the active group and the energy ($E_{(\text{inactive,T})}$, see Equation (15)) caused by the inactive nodes. Thus, we have

$$E_{\text{ECS}^2,\text{T}} = E_{(\text{active,T})} + E_{(\text{inactive,T})}. \tag{7}$$

Let P_{standby} be the average power (in Watts) consumed by a standby storage node. When a node is running, it has two different average power levels, namely, P_{light} and P_{heavy} (in Watts) under light and heavy loads, respectively. For simplicity, we assume that nodes respond to I/O requests at the average power level P_{power} (see Figure 7(a)).

Initially, we keep data nodes in the power-on mode to achieve high I/O performance, while placing parity nodes in the standby mode to achieve good energy efficiency. Figure 7(b) illustrates that a parity node stays in the standby mode at time-point $T = 0$, and it is powered on when the buffer utilization reaches threshold $V_{\text{threshold}}$. The parity node is transitioned into the standby mode after completing the parity synchronization process. That is, the parity node's power state is switched in a standby→activating→synchronization→deactivating→standby cycle.

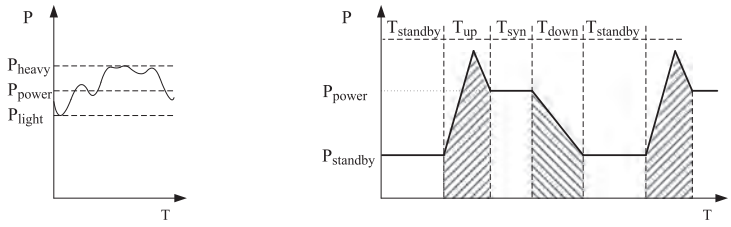


Fig. 7. The power curve of a storage node as a function of time. (a) Power fluctuation between P_{heavy} and P_{light} in the power-on mode; (b) Power transitions of a parity node.

There are N_{d} active data nodes in the active group. The energy dissipation in each storage node during the time period T is $P_{\text{power}}T$. The overall energy $E_{(\text{active,T})}$ (in Joules) consumed by the active group is the sum of the energies consumed by all the active nodes. Thus, we have

$$E_{(\text{active,T})} = P_{\text{power}} \times N_{\text{d}} \times T. \tag{8}$$

The write data can be striped across N_{d} data nodes with the strip size of S_{strip}. As to a write request of size $S_{\text{req\_w}}$, it may occupy the buffer space of size $S_{\text{req\_buf}}$,

$$S_{\text{req\_buf}} = \lceil S_{\text{req\_w}}/(N_{\text{d}}S_{\text{strip}}) \rceil \times S_{\text{strip}}. \tag{9}$$

If the buffer utilization reaches a predetermined threshold, all inactive parity nodes will be activated. A full transition includes powering up and powering down, and the number of full transitions N_{transit} can be expressed as

$$N_{\text{transit}} = \left\lfloor \frac{R_{\text{req\_w}}S_{\text{req\_buf}}T(N_{\text{p}}+1)}{S_{\text{buf}}} \right\rfloor. \tag{10}$$

The total transition time depends on the number of full transitions and time spent in each transition; therefore, the total transition time T_{transit} can be written as

$$T_{\text{transit}} = \begin{cases} T_{\text{full}}N_{\text{transit}} + T_{\text{up}}, & \text{Case1} \\ T_{\text{full}}N_{\text{transit}}, & \text{Otherwise} \end{cases}. \tag{11}$$

The total energy overhead E_{transit} (measured in Joules) caused by power-state transitions can be derived from the total transition time and power in the transitions. Hence, we have:

$$E_{\text{transit}} = \begin{cases} N_{\text{p}}E_{\text{full}}N_{\text{transit}} + E_{\text{up}}, & \text{Case1} \\ N_{\text{p}}E_{\text{full}}N_{\text{transit}}, & \text{Otherwise} \end{cases}. \tag{12}$$

where *Case1* represents the synchronization occurring at time point T.

The average time spent in synchronizing buffered parity blocks is the maximum value of reading time of parity blocks from N_{d} data nodes and the writing time of parity blocks to N_{p} parity nodes. Thus, we have the following equation where reading time is expressed as the first item on the right-hand side of the first line, and writing time is written as the second item on the right-hand side of the first line.

$$T_{\text{syn}} = MAX\left\{ \frac{R_{\text{req\_w}}S_{\text{req\_buf}}TN_{\text{p}}}{(N_{\text{d}}B_{\text{node}})}, \frac{R_{\text{req\_w}}S_{\text{req\_buf}}TN_{\text{p}}}{(N_{\text{p}}B_{\text{node}})} \right\} = MAX\left\{ \frac{N_{\text{p}}}{N_{\text{d}}}, 1 \right\} \times \frac{R_{\text{req\_w}}S_{\text{req\_buf}}T}{B_{\text{node}}} \tag{13}$$

The energy (in Joules) caused by synchronizing parity blocks to the parity nodes is expressed as $P_{\text{power}}N_{\text{p}}T_{\text{syn}}$. In Equation (13) the synchronization time is the maximum of the reading and writing times, so the data nodes can offer a portion of data transfer bandwidth for user I/O requests in addition to synchronization if $N_{\text{p}} < N_{\text{d}}$.

The total standby time can be calculated as $T - T_{\text{transit}} - T_{\text{syn}}$. Therefore, the energy (in Joules) consumed by N_{p} parity nodes in the standby mode is

$$P_{\text{standby}}N_{\text{p}}(T - T_{\text{transit}} - T_{\text{syn}}) \tag{14}$$

The overall energy $E_{(\text{inactive,T})}$ (in Joules) incurred by the inactive nodes is contributed by three factors: standby energy (see the first item on the right-hand side of Equation (15)), synchronization energy (see the second item on the right-hand side of Equation (15)), and transition energy (see the third item on the right-hand side of Equation (15)). Thus, the overall energy $E_{(\text{inactive,T})}$ can be expressed as

$$E_{(\text{inactive,T})} = P_{\text{standby}}N_{\text{p}}(T - T_{\text{transit}} - T_{\text{syn}}) + P_{\text{power}}N_{\text{p}}T_{\text{syn}} + P_{\text{transit}}. \tag{15}$$

Equation (9) represents cases where all written data is striped across N_{d} data nodes. If a write request is smaller than the strip unit, i.e. $S_{\text{req\_w}} < S_{\text{strip}}$, then such a striped layout can cause relatively high disk and network bandwidth. When it comes to small writes, written data can be stored in a single active node to improve write performance. Therefore, in this model we replace Equation (9) with Equation (16).

$$S_{\text{req\_buf}} = \begin{cases} \lceil S_{\text{req\_w}}/(N_{\text{d}}S_{\text{strip}}) \rceil \times S_{\text{strip}}, & S_{\text{req\_w}} > S_{\text{strip}} \\ S_{\text{req\_w}}, & \text{Otherwise} \end{cases} \tag{16}$$

4.3.2. Energy Consumption of Non-ECS^2. For comparison purpose, we model the energy consumption of the power-oblivious (non-ECS^2) system, which maintains the number $N_{\text{d}} + N_{\text{p}}$ of active nodes. Hence, the overall energy $E_{\text{oblivious,T}}$ (in Joules) consumed by non-ECS^2 during the time period T can be derived from power P_{power} and the number $N_{\text{d}} + N_{\text{p}}$ of data/parity nodes as shown below:

$$E_{\text{oblivious,T}} = P_{\text{power}}(N_{\text{d}} + N_{\text{p}})T \tag{17}$$

4.3.3. Energy Consumption in the Case of Failures. As mentioned in the preceding, we model the energy consumption of both the non-ECS^2 system and our ECS^2 system without considering node failures. In the case of any node failure, the power P_{power} is increased due to node reconstruction. The increase of power P_{power} in ECS^2 is close to that in non-ECS^2, because the same reconstruction process is employed in both systems. The parameters in the energy consumption model of ECS^2 are given as follows.

— When a failure occurs, an inactive parity node is awakened to join the active group. The failed node is replaced by a spare functional node. As a result, the active group size is increased by one from N_{d} and the inactive group size is reduced by one (i.e., N_{p}-1). The energy $E_{(\text{active,T})}$ (in Joules) consumed by the active group increases to $P_{\text{power}} \times (N_{\text{d}} + 1) \times T$ from $P_{\text{power}} \times N_{\text{d}} \times T$ (see Equation (8))
— Upon double failures, two parity nodes are activated. The active group size is increased by 2 (N_{d} + 2); the inactive group size is decreased by 2 ($N_{\text{p}} - 2$).
— When the number of concurrent node failures is i ($3 \leq i \leq N_{\text{p}}$), the ECS^2 system must activate all standby parity nodes, and the total power is the same as that of a non-ECS^2 system (see Equation (17)).

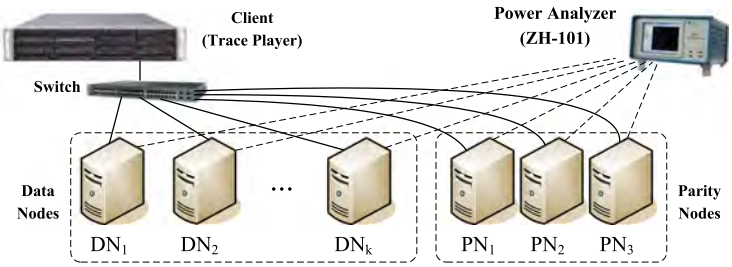


Fig. 8. Experimental Testbed. Red lines represent that the power analyzer records the current of each storage node through a probe. The power state of a parity node is transitioned between Standby and Active (Standby Mode ↔ Active Mode).

5. EVALUATION

5.1. Experimental Environment

In our experiment, we employ a server-level machine as a client node, and 12 PC-level machines as storage nodes. Each storage node is composed of a 3.2GHz Intel Core Duo processor(E5800), with 2GB of DDR3 RAM, Seagate disk model ST31000525SV with 1TB capacity, and Linux Ubuntu 10.04 *X*86_64 (Kernel 2.6.32) as its operating system. The client is a SuperMicro server with two Intel(R) Xeon(R) X5560 @2.67GHz (6 cores) CPUs, 12GB DDR3 RAM, and its operating system is Fedora 12 *X*86_64 (Kernel 2.6.32). All machines are connected to a WS-C3750G-24TS-S Cisco(R) switch with 24 Ethernet 10/100/1000 ports. To measure power consumption, we use an Electric Power Analyzer model ZH-101 [Zyhd 2010], which can record the current of each storage node in a sample range of 1∼10Hz.

Figure 8 shows the experimental testbed.

5.2. Prototype and Evaluation Methodology

5.2.1. Prototype. To create the power-efficient erasure-coded environment, an application-level trace player is implemented on the client node, which serves as multiple concurrent users. The trace player issues I/O requests according to the timestamps of the workload trace and records the request duration between sending requests and receiving the required data via TCP connections. If it is a read request, the request duration is the read response time; otherwise, it is the write response time. Here, we measure the duration of I/O requests in terms of *average user response time*. When a data node fails, and the I/O requests are located on the failed node, the user response time includes the duration of rebuilding the lost data. The data nodes not only respond to the I/O requests from the client node, but also periodically provide status info, such as buffer utilization, which is used to activate inactive parity nodes. As to the underlying erasure codes, we use the Zfec library [Wilcox-O'Hearn 2011], which is the implementation of the fastest Vandermonde-RS codes [Plank et al. 2009].

As shown in Algorithm 1, $R_{i,j}$ is a buffer region at node $\mathrm{DN_i}$ for block $\mathrm{P_j}$, thus $R_{i,j}$ can hold a set of parity blocks that will be migrated to *j*th parity node $\mathrm{PN_j}$. We encapsulate several parity blocks into a large request unit that can use the spare link capacity and maximize disk head utilization of the parity node, improving synchronization.

5.2.2. Evaluation Methodology. We compare the energy efficiency and performance of the ECS^2 system with those of a power-oblivious system in both normal and failure cases. In the power-oblivious (non-ECS^2) case, all the $k + r$ storage nodes are active; parity

Table V. The Characteristic of the Four Traces

Trace	Write Rate	IOPS	Avg. Req. Size	Avg. Write Req. Size	Data Set Size (1 Hour)
Web-2	0.02%	287	14.02KB	8.28KB	13.81GB
Fin-2	17.8%	128	2.89KB	3.05KB	1.27GB
DTRS	36.44%	57	27.87KB	30.47KB	5.45GB
RAD-BE	81.6%	91	25.25KB	10.67KB	7.89GB

blocks are stored in corresponding parity nodes. The primary evaluation metrics used in our experiments are the percentage of power savings and average user response time. We divide our experiments into the following four groups:

— under four different application scenarios;
— in the case of a data node failure;
— using the workload of different intensities;
— sensitivity to the number of data nodes.

To evaluate our ECS^2 system, we replay four real-world traces representing different workloads on the client node. These traces include WebSearch2.spc (Web-2), Financial2.spc (Fin-2), Developer Tools Release Server (DTRS), and RADIUS Back-end Server (RAD-BE). The Web-2 and Fin-2 traces were respectively collected from a popular search engine and an OLTP application server by the Storage Performance Council(SPC) [Application 2007]. The DTRS and RAD-BE were collected by the Microsoft Corporation in 2008. The DTRS trace was obtained from a file server accessed by more than 3000 users downloading various daily builds of Microsoft Visual Studio; the RAD-BE trace was derived from a back-end SQL server of the RADIUS authentication server that is responsible for worldwide corporate remote access and wireless authentication [Kavalanekar et al. 2008]. We choose a segment of each trace that covers a 1-hour period. Table V summarizes the I/O characteristics of the traces. Web-2 represents read-intensive workloads; Fin-2 resembles applications issuing small writes; both DTRS and RAD-BE represent write-dominated workloads. DTRS contains requests issued by large-write applications; RAD-BE consists of requests from write-intensive workloads (write rate is 81.6%).

To evaluate ECS^2 in terms of user response time, we adopt the open-loop model [Lumb et al. 2000], which aims to test the service capability of an underlying system where the I/O arrival rate is independent of I/O request completions. To illustrate node transitions, we configure the buffer space and buffer threshold according to the dataset size (see Table V). In the experiments, we allocate 200Mbytes of memory space in each data node to buffer parity blocks; we also set the buffer threshold as 0.9. That is, $S_{\text{buf}} = 1200$MBytes. Evidence shows that a configuration of r = 3 allows ECS^2 to achieve sufficient MTTDL for 10PB of archival storage [Storer et al. 2008], so we set N_{p} to be 3 in our experiments.

5.3. Experimental Results

We measure system parameters (see Table VI) of a storage node in a real-world cluster in our laboratory. The node power P_{light} is 70.8W when the node's I/O throughput is 10MBps; the power P_{heavy} is increased to 75.6W when the node's throughput goes up to 80MBps. Hence, the parameter P_{power} is set to 73.2W in our experiments. Because both user and synchronization I/Os are sharing the storage and network bandwidth, the value of B_{node} varies with dynamic workload conditions.

5.3.1. Four Different Scenarios. Figure 9(a) shows the power consumption of ECS^2 processing the Web-2 trace. In the non-ECS^2 system, parity nodes can stay in the idle

Table VI. Parameters and Their Values

Symbol	Value	Symbol	Value	Symbol	Value
N_d, k	6	$B_{node}(MB/s)$	40	$P_{idle}(Watt)$	61.8
N_p, r	3	$T_{up}(Sec)$	13	$P_{standby}(Watt)$	5.4
$S_{buf}(MB)$	1200	$T_{down}(Sec)$	7	$P_{light}(Watt)$	70.8
$S_{strip}(KB)$	16	$E_{up}(Joule)$	1270	$P_{heavy}(Watt)$	75.6
$V_{threshold}$	0.9	$E_{down}(Joule)$	569	$P_{power}(Watt)$	73.2

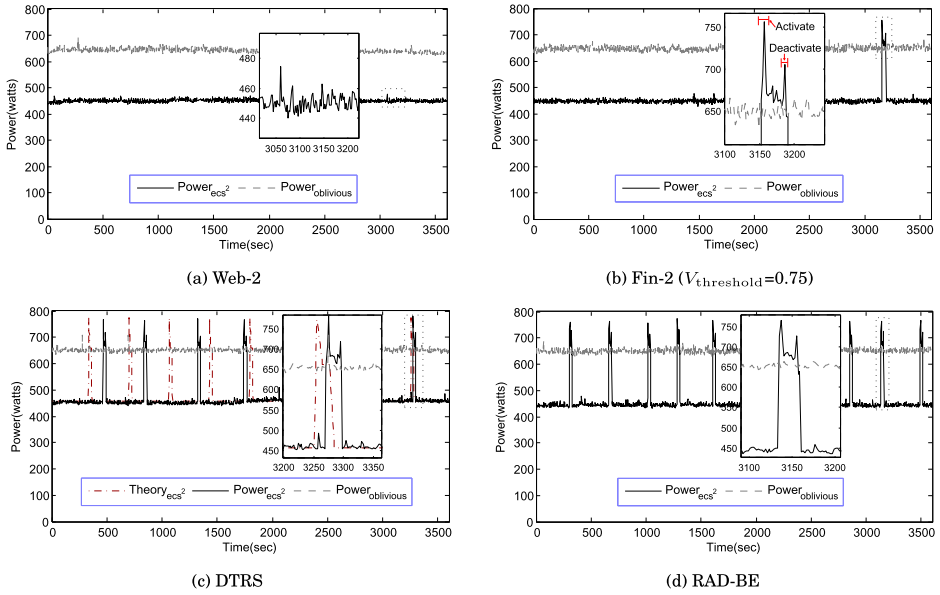


Fig. 9. Power-consumption comparison between ECS^2 and non-ECS^2 under four different workloads. The threshold in the Fin-2 case is set to 0.75; a node transition takes place within 3600 seconds. In (b), the activation and deactivation processes are marked in the partially enlarged curve. In (c), the theoretical power consumption is plotted as the red dotted line.

mode until a write request arrives. ECS^2 activates standby parity nodes when buffer utilization reaches threshold $V_{threshold}$. Table VII shows that ECS^2 can save energy by up to 29.8%—an energy-saving rate close to the optimal value $N_p \times (P_{power} - P_{standby})/(P_{power} \times (N_d + N_p))$ (e.g., 30.9% when $N_d = 6$ and $N_p = 3$).

Figure 9(b) shows the power consumption of both the ECS^2 and non-ECS^2 systems driven by the Fin-2 trace. The energy-saving rate of ECS^2 in the OLTP scenario can be as high as 30.3% (see Table VII), which is very close to the optimal value. High energy efficiency of ECS^2 is possible, because parity nodes stay in the standby mode for a longer time period than power-transition and data synchronization times. Although buffered parity blocks can be migrated to activated parity nodes in a batch way, it takes about 19 seconds due to small writes.

Figures 9(c) and 9(d) show the power consumptions of ECS^2 under the DTRS and RAD-BE traces, respectively. The energy-saving rates of ECS^2 in the DTRS and RAD-BE cases are up to 28.0% and 28.4%, respectively. The energy-saving rates for DTRS and RAD-BE are lower than those for Web-2 and Fin-2, because both DTRS and RAD-BE have high write load (e.g., $R_{req\_w} \times S_{req\_w}$), which leads to a large number of

Table VII. Average Power Consumption and Savings

Trace	Web-2	Fin-2	DTRS	RAD-BE
non-ECS^2(Watt)	648.68	654.09	660.37	657.56
ECS^2(Watt)	450.6	451.42	475.64	470.80
Power Saving	29.8%	30.3%	28.0%	28.4%

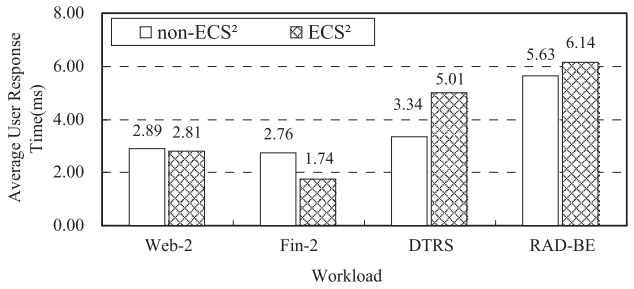


Fig. 10. Comparison of average user response time between ECS^2 and non-ECS^2 under different workload conditions.

node-power transitions. The synchronization time in the DTRS case is approximately 8 seconds—smaller than that of Fin-2 thanks to large data requests.

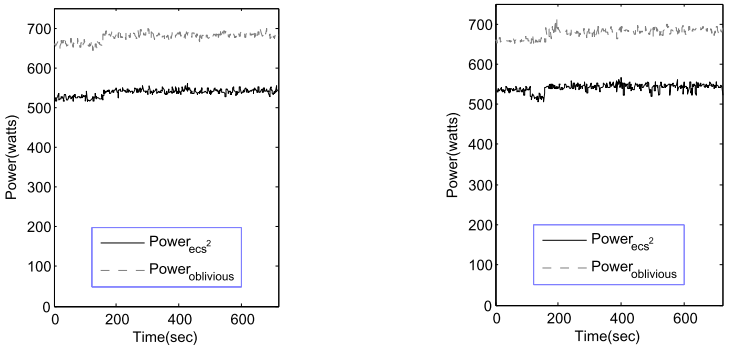
The discrepancy of power consumption between light and heavy loads is small (e.g., P_{light} = 70.8W and P_{heavy} = 75.6W); therefore, the fluctuation of the power consumption curve is small during the regular I/O access (see Figure 9(a)). The peak power consumption takes place during the parity-synchronization and node-transitioning periods (see the partially enlarged curve in Figure 9(b)).

Both experimental and theoretical results (see Figure 9(c)) indicate that there exists deviation in activation time points of inactive nodes in ECS^2, because DTRS has fluctuating workloads. Nevertheless, the partially enlarged curve shows that the theoretical results derived from the power model are consistent with the empirical results.

The average write throughput of DTRS is 61.80MB/s, compared to which, the RAD-BE trace has higher write throughput (77.37MB/s). Therefore, the RAD-BE trace causes more node transitions than DRTS does (see Figure 9(d)).

Figure 10 depicts the average response time in various scenarios. In read-intensive cases like Web-2, the average response times offered by ECS^2 are almost identical to those provided by the non-ECS^2 system. ECS^2 improves energy efficiency without affecting I/O performance, because read requests are directly served by data nodes. When it comes to small writes (e.g., the Fin-2 trace), requests are quickly responded to in ECS^2. This is because the writes are synchronized I/O. There are $N_{\text{d}} = 6$ and $N_{\text{d}} + N_{\text{p}} = 9$ writes in ECS^2 and non-ECS^2, respectively, thus the average response time in ECS^2 is less than that in non-ECS^2. On the other hand, the requested data in Fin-2 is very small, and the data stored close to the data that has been already requested will be cached owing to spatial locality; consequently, the average response time of Fin-2 is less than that of Web-2.

Under write-dominated workloads, the ECS^2-based system may lead to longer response times than the non-ECS^2 system (see Figure 10), because ECS^2 tends to issue more I/O requests to data nodes than non-ECS^2 does. For example, to create a file in the (9,6) RS-coded system, there are 10 I/Os (6 disk writes and 4 memory writes) issued to data nodes in ECS^2, as apposed to 6 in the non-ECS^2 system. To modify a data



(a) Reconstruction bandwidth is 450Mbps (b) Reconstruction bandwidth is 750Mbps

Fig. 11. The impact of reconstruction bandwidths on power consumption. The experiment is driven by the Web-2 trace.

block, there are 6 requests (1 disk read, 1 disk write, and 4 memory writes[2]) to the data nodes in ECS^2, but only 2 requests (1 disk read and 1 disk write) in non-ECS^2. Thus, data nodes in ECS^2 may face heavy I/O loads.

5.3.2. Handling Failures. Erasure-coded storage systems must efficiently deal with data recovery while providing regular storage services. We are now in a position to evaluate the energy efficiency and performance of ECS^2 when a data node fails. While handling any failure in ECS^2, both user and reconstruction I/Os must be responded to by surviving data nodes as well as an activated parity node (e.g., PN_1).

Upon a failure of any data node, the ECS^2 system enables a replacement node to sustain a certain reconstruction bandwidth (via bandwidth limiting using *iptables*). The replacement node first issues read requests to *k-1* surviving data nodes and an activated parity node; then, the replacement node rebuilds lost data stored on the failed node. Figure 11 plots the power consumption of ECS^2 driven by the Web-2 trace under various reconstruction bandwidths (450Mbps and 750Mbps). We observe that when the reconstruction bandwidth is set to 450Mbps, the average powers of ECS^2 and non-ECS^2 are 538W and 677W, respectively. In this case, ECS^2 conserves power by 20.5%. When the reconstruction bandwidth becomes 750Mbps, the average powers of ECS^2 and non-ECS^2 are 541W and 678.5W, respectively. The power saving rate in the second case is 20.3%. The power-saving rate of approximately 20% is attributed to the fact that only $N_p - 1$=2 parity nodes are kept in the standby mode during the reconstruction process. The power-saving rate's optimal value is $(N_p - 1) \times (P_{power} - P_{standby})/((N_d + N_p) \times P_{power})$ (e.g., optimal power-saving rate is 20.6% when N_d = 6 and N_p = 3).

Figure 12 depicts that average response times processed by both ECS^2 and non-ECS^2 for a failure. The results suggest that regardless of ECS^2 or non-ECS^2, the reconstruction process can slow down response times. If an aggressive reconstruction scheme is adopted (e.g., the reconstruction bandwidth is set to 750Mbps), the system will experience much higher response times than the same system using a nonaggressive scheme (e.g., the reconstruction bandwidth is only set to 450Mbps).

5.3.3. Impacts of Workload Intensities. To examine the sensitivity of ECS^2 to I/O workloads, we increase the workload intensity (e.g., $\times$1.5, $\times$2.0, $\times$2.5) of the DTRS trace

[2]The parity block can be updated within the memory.

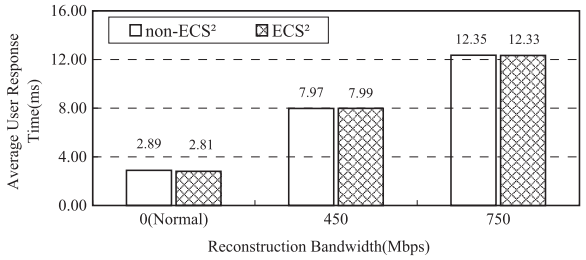


Fig. 12. The impact of reconstruction bandwidth on average user response time in the ECS^2 and non-ECS^2 systems. The experiment is driven by the Web-2 trace.

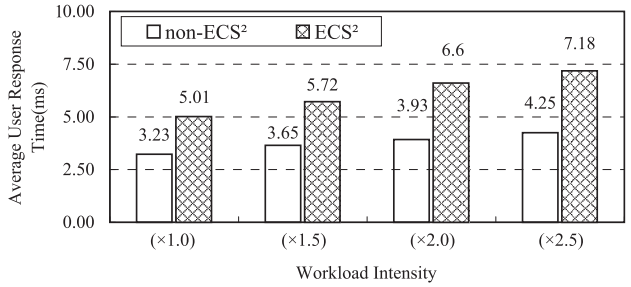


Fig. 13. The impact of workload intensity on average user response time in the ECS^2 and non-ECS^2 systems. The experiment is driven by the DTRS trace.

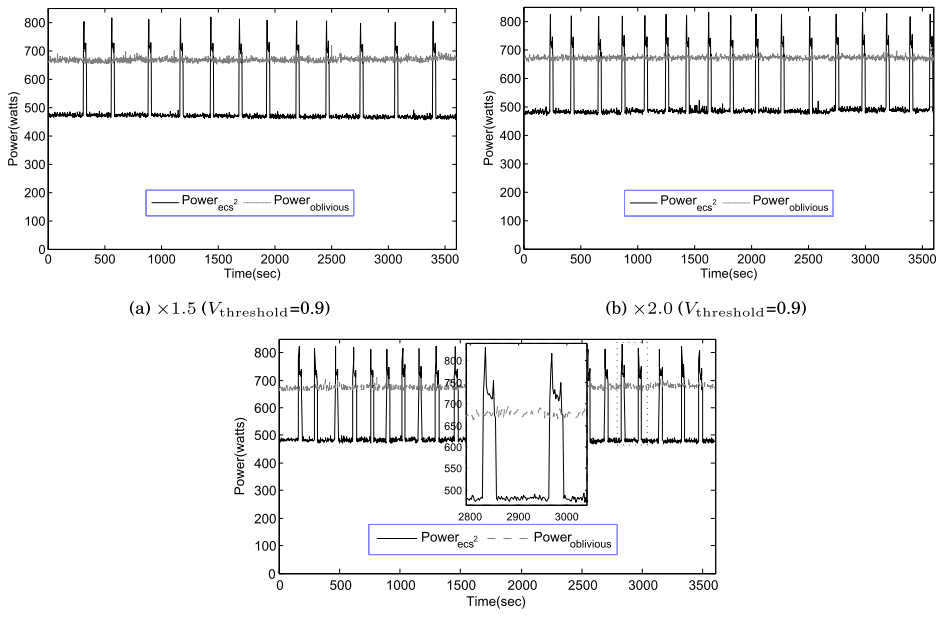
by shrinking its time-stamps. To make ECS^2 handle heavy I/O workloads (e.g., $\times$2.5 intensity), threshold $V_{\text{threshold}}$ is set to 0.8 so that there is sufficient buffer space to hold the parity blocks (see Equation (5)). Thus, the parameter $V_{\text{deviation}}$ becomes 0.1.

Figure 13 shows that the average response times in both ECS^2 and non-ECS^2 systems increase with I/O load. The response time of ECS^2 is more sensitive to I/O intensity than that of non-ECS^2. From Figures 14 and 9(c), it is indicated that with the increase of I/O intensity, the number of power transitions of parity nodes is increasing, because large values of $R_{\text{req\_w}}$ and $S_{\text{req\_w}}$ lead to more power transitions (N_{transit}). Therefore, I/O workload conditions noticeably affect the energy efficiency of ECS^2.

Compared to read-intensive or small-write loads, write-dominated workloads give rise to a large number of power transitions of parity nodes in ECS^2. Such a high power-transition frequency not only adversely affects the life-cycle of parity nodes, but also reduces the energy efficiency of ECS^2 because the standby duration of the parity nodes is shortened (see Table VIII). Write throughput, buffer size, and buffer threshold have impacts on the energy efficiency of ECS^2, therefore, the deployment of ECS^2 to write-intensive applications should be carefully considered.

5.3.4. Impact of Data Node Number. In this experiment, we focus on the sensitivity of ECS^2 to the number of data nodes. We set the fault-tolerance parameters k and r to be 9 and 3, respectively. Figure 15 shows the average user response times in the ECS^2 and non-ECS^2 systems with both (9,6)RS and (12,9)RS codes. The trace replayed in this experiment is DTRS.

We observe from Figure 15 that the average user response time of the ECS^2-based system decreases when the number of data nodes increases. This trend is reasonable, because a large number of nodes gives rise to large available buffer space. As such,



(a) $\times 1.5$ ($V_{\text{threshold}}$=0.9)

(b) $\times 2.0$ ($V_{\text{threshold}}$=0.9)

(c) $\times 2.5$ ($V_{\text{threshold}}$=0.8)

Fig. 14. The impact of workload intensity on power consumption. The experiment is driven by the DTRS trace.

Table VIII. Average Power Consumption of the System Under Different Workload Intensities

Workload Intensity	$\times 1.0$	$\times 1.5$	$\times 2.0$	$\times 2.5$
non-ECS^2(Watt)	660.37	665.10	672.87	675.85
ECS^2(Watt)	475.64	492.79	517.58	526.39
Power Saving	28.0%	25.9%	23.1%	22.1%

The experiment is driven by the DTRS trace.

the impact of synchronization on I/O performance is weakened. On the other hand, the average user response time of the non-ECS^2-based system increases sightly when the number of nodes increases, because the overhead incurred by synchronized writes may go up with increasing node numbers.

6. FURTHER DISCUSSION

In this study, we leverage memory space in active nodes to buffer parity blocks, which are flushed to activated parity nodes when the I/O buffer's utilization reaches a specified threshold. In the worst case, a second node fails when the system is recovering the first failed node. Although all buffered parity blocks can be rebuilt without loss of data, the system will experience high I/O response times due to the fact that a portion of the active nodes' disk bandwidth is reserved for data reconstruction. To alleviate the performance degradation due to the data recovery process, one can simply lower the buffer threshold when the system is in the process of reconstructing data.

Our study shows that ECS^2 can significantly improve the energy efficiency of storage clusters, because all $\boldsymbol{r}$ parity nodes are placed into an inactive group. Although one or two extra parity blocks may require additional I/O bandwidth and memory space, this data layout is worthwhile, because the extra parity blocks can be used to

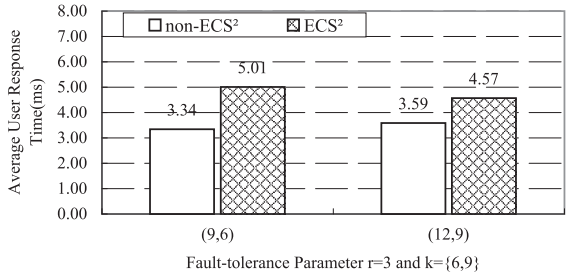


Fig. 15. The impact of data node number on average user response time in the ECS^2 and non-ECS^2 systems. The experiment is driven by the DTRS trace.

reconstruct data upon double node failures, and the fault tolerance level of the active group increases.

It is critical to synchronize parity updates to persistent storage in a timely manner during the synchronization process, which can interfere with system performance due to shared disk and network bandwidth. The aforementioned experimental results (see Section 5) indicate that the synchronization process in our ECS^2 is fairly efficient; synchronizations are completed in a short time with marginal periodic effects on system performance.

During a utility power outage in which buffered data may be lost, we can extend ECS^2 to employ per-server battery back-ups (see, for example, Ongaro et al. [2011]). This approach provides power long enough for ECS^2 to synchronize all the buffered parity blocks.

When data nodes fail, parity node PN_1 is activated to boost the reconstruction performance. Alternatively, all the parity nodes in ECS^2 can be activated in a round robin mode to offer longevity of PN_1.

If parity nodes fail, the reconstruction process is equivalent to encoding operations. In other words, the following three phases are involved in the reconstruction. First, data blocks are retrieved from k active data nodes. Second, the failed parity blocks are produced according to the *Generator Matrix*. Finally, the recovered parity blocks are written to the corresponding replacement parity nodes.

Each node may store both data and parity blocks simultaneously, meaning that some nodes can be transitioned into the low-power mode via scheduling the I/O requests. This alternative process allows us to balance load and lengthen the life of data nodes compared to ECS^2, but it may cause high response time and recovery time without exploiting the systematic feature of RS codes. Such a scheduler is not incorporated into our system, because ECS^2 aims to offer energy savings with marginal degradation of reliability and performance.

If the loss of user data is not affordable and desirable, one or more parity nodes in ECS^2 can be placed in the active mode all the time (see Figure 1(b)) to enhance system reliability. Such a configuration improves the system reliability at the cost of energy efficiency, that is, there is a trade-off between energy efficiency and reliability.

7. CONCLUSIONS AND FUTURE WORK

This article shows that existing energy conservation techniques for storage clusters reduce energy consumption at the cost of performance or reliability of the systems. To

overcome this limitation, we design an energy-efficient erasure-coded storage cluster or ECS^2 for high-performance data-intensive scenarios. Compared to energy-oblivious reliable storage systems, our ECS^2 has four salient characteristics: (1) ECS^2 achieves high energy efficiency by keeping all parity nodes in the power-saving mode while exclusively buffering new parity blocks in active nodes to handle deferred writes; (2) to improve reliability, ECS^2 transitions the fault-tolerance level of an active group from $\lfloor r/2 \rfloor$ to $\lceil (r+1)/2 \rceil$ by buffering one or two extra parity blocks; (3) ECS^2 is cost effective, because new parity blocks are temporarily buffered to preallocated memory space of active nodes without requiring extra dedicated devices (e.g., cache disks); and (4) ECS^2 maintains good I/O performance as I/O requests are quickly and directly serviced by powered-on data nodes, fully utilizing the systematic feature of RS codes.

Our quantitative analysis indicates that ECS^2 can achieve high reliability of the active group. Our evaluation shows that for the (9,6) RS-coded prototype, ECS^2 effectively saves power by up to 29.8% and 20.4% under normal conditions and one failure, respectively, while achieving similar I/O performance of energy-oblivious erasure-coded storage for read-intensive workloads. Our experimental results also show that ECS^2 improves energy efficiency by 28.0% for write-dominated applications.

In the future, we plan to extend this work by designing an energy-efficient non-MDS-coded storage cluster. This is because relative to (k + r, k) MDS codes, non-MDS codes have many opportunities to reconstruct the data directed to standby nodes from less than k active nodes.

APPENDIX

Matrix-Based Scheme to Construct Extra Parity Blocks

In a (k + r, k) RS-coded storage cluster, any k surviving blocks can rebuild the original blocks. Assume r parity blocks are exclusively buffered at the k active data nodes, so the active group can tolerate $\lfloor r/2 \rfloor$ node failures. Namely,

$$FT_{\mathrm{r}} = \lfloor r/2 \rfloor . \tag{18}$$

In order to improve the fault tolerance of the active group (e.g., $FT_{\mathrm{r}'} = FT_{\mathrm{r}} + 1$), it is normal to adopt a $(k + r', k)$ coding scheme, where $r' > r$. According to Formula (18),

$$FT_{\mathrm{r}'} = \lfloor r'/2 \rfloor . \tag{19}$$

To minimize encoding overhead, we get $r' = 2FT_{\mathrm{r}'}$ from Formula (19), then

$$r' = 2FT_{\mathrm{r}} + 2 = 2\lfloor r/2 \rfloor + 2 = \begin{cases} r+1, & \mathrm{r\ is\ odd}, \\ r+2, & \mathrm{r\ is\ even} \end{cases} . \tag{20}$$

From Formula (20), we find that

$$\Delta r = r' - r = \begin{cases} 1, & \mathrm{r\ is\ odd} \\ 2, & \mathrm{r\ is\ even}, \end{cases} \tag{21}$$

$$FT_{\mathrm{r}'} = \lceil (r+1)/2 \rceil . \tag{22}$$

Thus, We can adopt (k + r + Δr, k) RS codes to construct r + Δr parity blocks $\{\mathrm{P}_1, \mathrm{P}_2, ..., \mathrm{P}_{\mathrm{r}+\Delta\mathrm{r}}\}$, which can be exclusively buffered in data nodes $\{\mathrm{DN}_1, \mathrm{DN}_2, ..., \mathrm{DN}_{\mathrm{k}}\}$. The active groups can achieve $\lceil (r+1)/2 \rceil$-fault tolerance. After synchronizing parity blocks $\{\mathrm{P}_1, \mathrm{P}_2, ..., \mathrm{P}_{\mathrm{r}}\}$ to the corresponding parity nodes $\{\mathrm{PN}_1, \mathrm{PN}_2, ..., \mathrm{PN}_{\mathrm{r}}\}$, all $k + r$ nodes can achieve r-fault tolerance. Figure 16 shows the data flow.

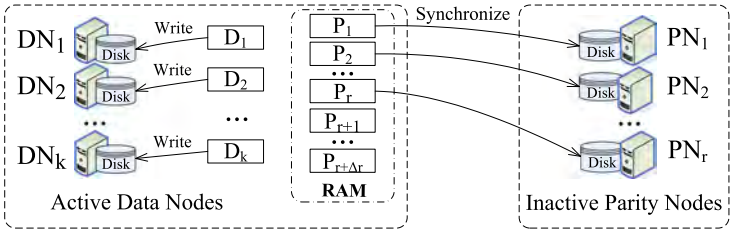


Fig. 16. Data flow of power-efficient (k + r, r)RS-coded storage cluster with Δr extra parity blocks, with $\lceil (r+1)/2 \rceil$-fault tolerance for k active nodes and r-fault tolerance for $k + r$ nodes.

From Formula (21), it is seen that an extra parity block can make the active group 2-fault tolerant when r = 3.

ACKNOWLEDGMENTS

The authors would like to sincerely thank all the anonymous reviewers for their constructive comments in reviewing this article. The authors also appreciate Shuaijun Han and Shenjie Cui for their assistance in the development of the experimental platform for this study.

REFERENCES

Aguilera, M., Janakiraman, R., and Xu, L. 2005. Using erasure codes efficiently for storage in a distributed system. In *Proceedings of IEEE International Conference on Dependable Systems and Networks (DSN)*. 336–345.

Amur, H., Cipar, J., Gupta, V., Ganger, G., Kozuch, M., and Schwan, K. 2010. Robust and flexible power-proportional storage. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. 217–228.

Application, O. 2007. I/O and search engine I/O. umass trace repository. http://traces.cs.umass.edu/.

Bairavasundaram, L., Goodson, G., Pasupathy, S., and Schindler, J. 2007. An analysis of latent sector errors in disk drives. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. 289–300.

Blaum, M., Brady, J., Bruck, J., and Menon, J. 1995. Evenodd: An efficient scheme for tolerating double disk failures in raid architectures. *IEEE Trans. Comput. 44,* 2, 192–202.

Borthakur, D. 2008. Hdfs architecture guide. http://hadoop.apache.org/hdfs.

Borthakur, D. 2010. Hdfs and erasure codes (hdfs-raid).

Chen, F., Jiang, S., and Zhang, X. 2006. Smartsaver: Turning flash drive into a disk energy saver for mobile computers. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 412–417.

Colarelli, D. and Grunwald, D. 2002. Massive arrays of idle disks for storage archives. In *Proceedings of the ACM/IEEE Conference on Supercomputing*. 1–11.

Corbett, P., English, B., Goel, A., Grcanac, T., Kleiman, S., Leong, J., and Sankar, S. 2004. Row-diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST)*. 1–14.

Fan, B., Tantisiriroj, W., Xiao, L., and Gibson, G. 2009. Diskreduce: Raid for data-intensive scalable computing. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage*. 6–10.

Ganesh, L., Weatherspoon, H., Balakrishnan, M., and Birman, K. 2007. Optimizing power consumption in large scale storage systems. In *Proceedings of the 11th USENIX Workshop On Hot Topics In Operating Systems*.

Ghemawat, S., Gobioff, H., and Leung, S. 2003. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. 29–43.

Greenan, K., Long, D., Miller, E., Schwarz, S., and Wylie, J. 2008. A spin-up saved is energy earned: achieving power-efficient, erasure-coded storage. In *Proceedings of the 4th USENIX Conference on Hot Topics in System Principles*. 4–10.

Hafner, J. 2005. Weaver codes: Highly fault tolerant erasure codes for storage systems. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies*. 211–224.

Hafner, J. and Rao, K. 2006. Notes on reliability models for non-MDS erasure codes. IBM Res. rep. RJ10391.

Hafner, J., Pandey, P., and Thakur, T. 2010. Read-modify-write protocol for maintaining parity coherency in a write-back distributed redundancy data storage system. US Patent App. 12/710,123.

Holland, M., Gibson, G., and Siewiorek, D. 1994. Architectures and algorithms for on-line failure recovery in redundant disk arrays. *Distrib. Par. Datab. 2*, 3, 295–335.

Jiang, W., Hu, C., Zhou, Y., and Kanevsky, A. 2008. Are disks the dominant contributor for storage failures?: a comprehensive study of storage subsystem failure characteristics. *ACM Trans. Storage 4,* 3, 7.

Kavalanekar, S., Worthington, B., Zhang, Q., and Sharda, V. 2008. Characterization of storage workload traces from production windows servers. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. 119–128.

Lang, W. and Patel, J. 2010. Energy management for mapreduce clusters. *Proc. VLDB Endow. 3*, 1–2, 129–139.

Lang, W., Patel, J., and Naughton, J. 2010. On energy management, load balancing and replication. *ACM SIGMOD Rec. 38,* 4, 35–42.

Leverich, J. and Kozyrakis, C. 2010. On the energy (in) efficiency of hadoop clusters. *ACM SIGOPS Oper. Syst. Rev. 44,* 1, 61–65.

Li, D. and Wang, J. 2004. Eeraid: Energy efficient redundant and inexpensive disk array. In *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*.

Lumb, C., Schindler, J., Ganger, G., Nagle, D., and Riedel, E. 2000. Towards higher disk head utilization: extracting free bandwidth from busy disk drives. In *Proceedings of the 4th USENIX Conference on Symposium on Operating System Design & Implementation-Volume 4*.

Meisner, D., Gold, B., and Wenisch, T. 2009. Powernap: Eliminating server idle power. *ACM SIGPLAN Notices 44*, 3, 205–216.

Moore, R., D'Aoust, J., McDonald, R., and Minor, D. 2007. Disk and tape storage cost models. http://users.sdsc.edu/∼mcdonald/content/papers/dt cost.pdf.

Narayanan, D., Donnelly, A., and Rowstron, A. 2008. Write off-loading: Practical power management for enterprise storage. *ACM Trans. Storage 4,* 3, 10.

Ongaro, D., Rumble, S., Stutsman, R., Ousterhout, J., and Rosenblum, M. 2011. Fast crash recovery in ramcloud. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*. 29–41.

Phanishayee, A., Krevat, E., Vasudevan, V., Andersen, D., Ganger, G., Gibson, G., and Seshan, S. 2008. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*.

Pinheiro, E. and Bianchini, R. 2004. Energy conservation techniques for disk array-based servers. In *Proceedings of the 18th Annual International Conference on Supercomputing (ICS)*. 68–78.

Pinheiro, E., Bianchini, R., and Dubnicki, C. 2006. Exploiting redundancy to conserve energy in storage systems. *ACM SIGMETRICS Perf. Eval. Rev. 34,* 1, 15–26.

Plank, J. 2005. Erasure codes for storage applications. In *Proceedings of 4th USENIX Conference on File and Storage Technologies (FAST)*–Tutorial slides.

Plank, J. et al. 1997. A tutorial on Reed-Solomon coding for fault-tolerance in raid-like systems. *Softw. Pract. Exper. 27*, 9, 995–1012.

Plank, J., Luo, J., Schuman, C., Xu, L., and Wilcox-O'Hearn, Z. 2009. A performance evaluation and examination of open-source erasure coding libraries for storage. In *Proccedings of the 7th Conference on File and Storage Technologies (FAST)*. 253–265.

Rao, K., Hafner, J., and Golding, R. 2011. Reliability for networked storage nodes. *IEEE Trans. Dependable Sec. Comput. 8,* 3, 404–418.

Reed, I. and Solomon, G. 1960. Polynomial codes over certain finite fields. *J. Soc. Indus. Appl. Math. 8,* 2, 300–304.

Samsung. 2008. Data sheet of Samsung ddr2 sdram. http://www.samsung.com/.

Seagate. 2011. Data sheet of seagate disk drive. http://www.seagate.com/docs/pdf/datasheet/.

Storer, M., Greenan, K., Miller, E., and Voruganti, K. 2008. Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*.

Tsirogiannis, D., Harizopoulos, S., and Shah, M. 2010. Analyzing the energy efficiency of a database server. In *Proceedings of the International Conference on Management of Data*. 231–242.

Wang, J., Zhu, H., and Li, D. 2007. eRAID: Conserving energy in conventional disk-based raid system. *IEEE Trans. Comput.*, 359–374.

Weatherspoon, H. and Kubiatowicz, J. 2002. Erasure coding vs. replication: A quantitative comparison. In *Peer-to-Peer Systems*, 328–337.

Weddle, C., Oldham, M., Qian, J., Wang, A., Reiher, P., and Kuenning, G. 2007. Paraid: A gear-shifting power-aware raid. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*.

Wilcox-O'Hearn, Z. 2011. Zfec 1.4.2. open source code distribution. http://pypi.python.org/pypi/zfec.

Xin, Q., Miller, E., Schwarz, T., Long, D., Brandt, S., and Litwin, W. 2003. Reliability mechanisms for very large storage systems. In *Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST)*. 146–156.

Yang, L., Li, X., and Zhang, X. 2011. Research on energy consumption of general storage network system. *Comput. Eng. (China) 37*, 18, 53–58.

Yao, X. and Wang, J. 2006. Rimac: A novel redundancy-based hierarchical cache architecture for energy efficient, high performance storage systems. *ACM SIGOPS Oper. Syst. Rev. 40*, 4, 249–262.

Zhang, Z., Deshpande, A., Ma, X., Thereska, E., and Narayanan, D. 2010. Does erasure coding have a role to play in my data center? Tech. rep. MSR-TR-2010-52, Microsoft.

Zhu, Q., David, F., Devaraj, C., Li, Z., Zhou, Y., and Cao, P. 2004. Reducing energy consumption of disk storage using power-aware cache management. *IEE Proc. Softw.,* 118–118.

Zhu, Q., Chen, Z., Tan, L., Zhou, Y., Keeton, K., and Wilkes, J. 2005. Hibernator: Helping disk arrays sleep through the winter. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles*. 177–190.

Zyhd. 2010. Zh-101 portable electric power fault recorder and analyzer. http://www.zyhd.com.cn/cN/Bs_Product.asp.

Received May 2012; revised October 2012; accepted January 2013