Multimed Tools Appl (2017) 76:1139–1157
DOI 10.1007/s11042-015-3019-1

Energy-aware and intelligent storage features for multimedia devices in smart classroom

Mehdi Pirahandeh[1] · **Deok-Hwan Kim**[1]

Received: 27 March 2015 / Revised: 4 September 2015 / Accepted: 19 October 2015 /
Published online: 1 December 2015

Abstract With the recent big-data processing in multimedia devices becoming a popular application, a fast and energy efficient storage area network system for smart classroom is required. Traditional storage management system for smart classroom show low performance when small read and write operations are executed. This paper proposes a smart classroom storage management system (SCSMS) which consists of new adaptive chunking and XOR reference matrix based erasure coding techniques for multimedia devices with higher input/output performance and low energy consumption. The SCSI initiator is installed in multimedia devices such as smart TVs, smart phones and personal computers. The proposed adaptive chunking and exclusive-or (XOR) reference matrix-redundant array of inexpensive disks (XRM-RAID)are provided at a target server based on flash array storage, respectively. Adaptive chunking differs from traditional chunking in that it reduces the number of read and write operations by merging small files into a united chunk. XRM-RAID differs from existing RAID in that it reduces the number of XOR operations to generate parity data in the RAID system. This paper provides web based monitoring application of the proposed SCSMS. Experimental results show that the energy consumption of the proposed SCSMS is improved by 32 %, 42 % and 58 % compared to Huang et al., Kim et al. and Scott et al. with respect to file size and buffer size. In terms of the average write throughput, the proposed SCSMS has higher performance by 22 %, 32 % and 56 % compared to Huang et al., Kim et al. and Scott et al. with respect to file size and buffer size.

Keywords Smart classroom · Multimedia storage · Erasure coding · Chunking strategy · Energy consumption

✉ Deok-Hwan Kim
deokhwan@inha.ac.kr

[1] Department of Electronic Engineering, Inha University, Incheon 402-751, South Korea

1 Introduction

Nowadays, data sharing among multimedia devices in smart classroom using storage area network becomes popular [7], and the need for high throughput and low energy consumption increases sharply [3, 8, 9, 18, 19]. The smart classroom storage management system(SCSMS) consists of context-aware data initiator, storage network, chunking strategy, erasure coding/replication mechanism and services management.Types of the storage network, chunking strategy and erasure coding mechanism poses the challenges in term of small input/output(IO) and erasure coding performance to develop storage management system for smart classroom. The proposed SCSMS is compared to existing frameworks in terms of storage network, chunking strategy and redundant array of inexpensive disks(RAID) specifications in Table 1. To process information easier, chunking strategy is used to group large amounts of information into manageable small units in storage management system for smart classroom. Huang et al. [9] and Scott et al. [18] are applied traditional block based chunking strategy(TBC) whereas Kim et al. [19] is applied traditional object based chunking (TOC)strategy. These traditional SCSMSs show low performance when small input/output(IO)operations are executed. Storage network provides a transformation channel to store data from multimedia devices into smart classroom storage system. For example, students and teachers can download data content (e.g. Audio files) with set-top boxes (STBs), and store the content in a storage management system for smart classroom [7, 18]. The stored content can then be accessed by the other multimedia devices (e.g. camera). Huang et al. [9] uses Network-attached storage (NAS) and Scott et al. [18] uses peer to peer(P2P) network whereas Kim et al. [19] uses cloud environment. Storage networks such as NAS and P2P network performance are varied based on their bandwidth size. To provide data redundancy, storage management system for smart classroom employs RAID storage virtualization technology. RAID combines multiple physical disks into a single logical unit. RAID 1 provides replication of data by coping data from original physical disks into backup physical disks. However, traditional RAID 5 and RAID 6 are used in Huang et al. [9] and Kim et al. [19], they provide data redundancy by employing erasure codes(EC) using expensive XOR and multiplication operations between data vector and coding matrix. Erasure coding requires huge dedicated computing processors and memory space.

This paper proposes a new adaptive block based chunking strategy(ABC) and XOR reference matrix (XRM) based erasure coding techniques for smart classroom storage management system. The proposed SCSMS consists of initiator devices, target server with all flash array storages and high-speed storage area network(SAN). Multimedia devices such as smart camera, smart table, smart board, smart TV, smart phone and personal computer are connected to flash array storage server through an Internet Small Computer System Interface (iSCSI) initiator. The proposed chunking technique merges the small files into a

Table 1 The comparison of the developed storage management system for smart classroom

Smart classroom system	Storage network	Chunking strategy	RAID Spec.
Huang et al. [9]	NAS	TBC	EC/Replication
Scott et al. [18]	P2P network	TBC	-
The proposed SCSMS	SAN	ABC	XRM-RAID
Kim et al. [19]	Cloud	TOC	EC/Replication

united block where the size is equal to the buffer size, which decreases the number of physical read and write operations. In addition, exclusive-or (XOR) reference matrix-redundant array of inexpensive disks (XRM-RAID) implemented at the target storage server provides resistance against data failure. XRM-RAID generates parity data by using XOR reference matrix rules, XRM algorithm and an XRM table. It reduces the number of XOR operations to encode and decode data in storage management system for smart classroom.

2 Background

Three major types of the storage management system for smart classroom are used to manage read and write operations of the multimedia devices as shown in Table 1. We discuss the pros and cons of these existing storage management system as follows,

(i) Huang et al. [9] designed a Network-Attached Storage(NAS)which provides an easy way for data sharing and backup among multiple multimedia devices in home networks. This paper provides a write-back policy for home (or classroom) NAS called TESA. TESA considers both temporal and spatial information of the dirty buffers to improve IO performance.

(ii) Scott et al. [18] presented a storage system for smart classroom when various learning devices(eg. camera, personal computer) use dynamic peer to peer(P2P)storage network to share and store data.

(iii) Kim et al. [19] proposed a context-aware learning system such as classroom in a cloud computing environment. Four smarts elastic functions(E4S) such as smart pull, smart prospect, smart content, and smart push are designed to share and store information from various multimedia devices into a cloud storage.

2.1 Effect of chunking strategy in smart classroom storage management system

In the traditional chunking technique used in smart classroom storage management system, an input workload F splits a file directly into a chunk or many chunks [4, 6, 10, 12]. However, if current file f_i has a file size less than the given buffer size, the last part of the chunk is filled with zero bits. The number of read operations using traditional chunking γ_{read} is derived from (1). The symbols m, $last$, and i denote the number of small files, the total number of files and the index file in workload F. The total number of write operations is calculated by summing the numbers for parity data and γ_{read} [1, 2, 11, 13, 15, 21].

$$\gamma_{read} = \left(\sum_{i=m+1}^{last} \frac{sizeoff_i}{buffersize} \right) + m. \tag{1}$$

This paper categorizes data file into small file and large file. If the file size is less than the buffer size, it is called as small file. Otherwise, it is considered as a large file. Figure 1 shows the read performance of small files sized between 44 % and 67 % of large files with respect to four SSD manufacturers. The results also shows write performance of small files sized between 37 % and 61 % of large files with respect to four SSD manufacturers. Flash array storage in terms of life cycle and IO performance is significantly relevant to the number of updates, erase operations, and read and write operations [4–6].

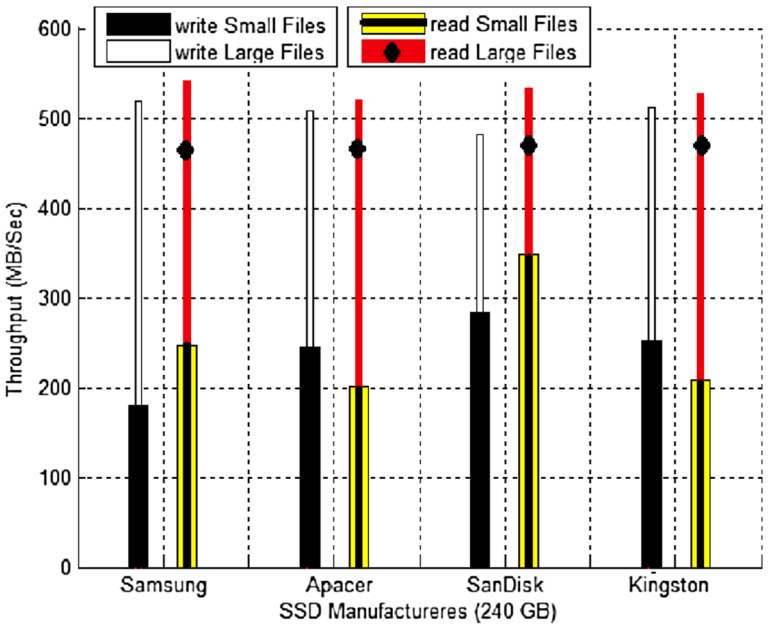

Fig. 1 Read and write performance of small and large files

2.2 Effect of erasure coding in smart classroom storage management system

In Huang et al. [9] and Kim et al. [19], traditional SCSMSs employ various erasure coding schemes called ”parity”. Erasure coding is a widely used technique in information technology to provide fault tolerance for the data. Most schemes use a simple XOR operation, but recent RAID 6 scheme uses two separated parities. These parity elements are generated using addition and multiplication in a particular Galois Field or various type of erasure codes. For instance, RAID 6 requires lower space than data replication in a RAID 1 system. The erasure codes demonstrate a range of performance in terms of encoding time, access to memory, CPU overflow, space efficiency, and resilience to failure. Li, Shu et al. [4] categorized the erasure codes into various types such as Reed-Solomon codes, parity array codes and parity check codes based on their encoding methods. Parity check codes were constructed based on single parity check (SPC) codes [8, 21]. The two-dimensional horizontal and vertical parity check (HVPC) codes are a typical representative of parity check codes [14, 16]. SPC codes are completely based on the number of XOR operations. An HVPC erasure code structure with $m \times n$ data elements [4, 14, 20]. This structure includes m strips, n disks and p is the number of parity disks. All data and parities have an equal element size. The encoding process used in erasure codes is achieved by doing bit matrixivector operations in Galois-Field Arithmetic, where the data elements are defined as the vector, and the parity elements are defined as the erasure codes [3, 4, 8, 17]. Because there are only ones and zeros in the coding matrix for creating the bit matrix, the matrix-vector operations are defined based on exclusive-or operations. Therefore, the performance of the erasure code depends strongly on the number of XOR operations conducted during the encoding execution [5, 20]. The encoding matrix equation is described based on previous research [4, 6],

$(D|C) = DG = D \times (I|H)$, where generator matrix $G = (I|H)$ is composed of I with $wn \times wn$ bits and H with $pw \times wn$ bits. Code words $(D|C)$ consist of wn data bits and wp parity bits.

3 The proposed smart classroom storage management system

Figure 2 shows an overview of the proposed smart classroom storage management system. This system, being supported by underlying storage area network (SAN), can store and retrieve data from multimedia devices such as game console, CCTV, video camera, smart camera, smart table, smart board, audio player, phones, tablets and portable computers. Smart classroom users can also monitor and manage SCSMS storage performance through a web based application. The proposed SCSMS consists of various software components as follows,

Data initiator Figure 3 shows the process of gathering data files from various multimedia devices deployed in smart classroom environments. Data requests from local USB HUB and remote iSCSI initiator will be queued at context aggregator. Then, data files will be passed to the next component through the context aggregator function. This component also provides all data share platform.

Adaptive chunking This is a kind of intelligent processing component. The context or data is defined as a collection of information characterized by a group of people and gathered from multimedia devices in a smart classroom during the specific period of times. The adaptive chunking assorts data files and merges small files into a united chunk and splits large files into several data chunks so that it can decrease the number of read and write operations. Then, data will be passed to the next component through the data element aggregator function.

XOR reference matrix (XRM)-RAID This component provides the fault tolerance against disk failure for data stored from the data element aggregator. This paper proposes XRM-RAID, which strips data into data disks, and generates parity data and strips parity data into the parity disks. XRM-RAID generates parity data by using XRM rules and an XRM table. An XRM rule is used to reduce the search scope of the encoding and decoding

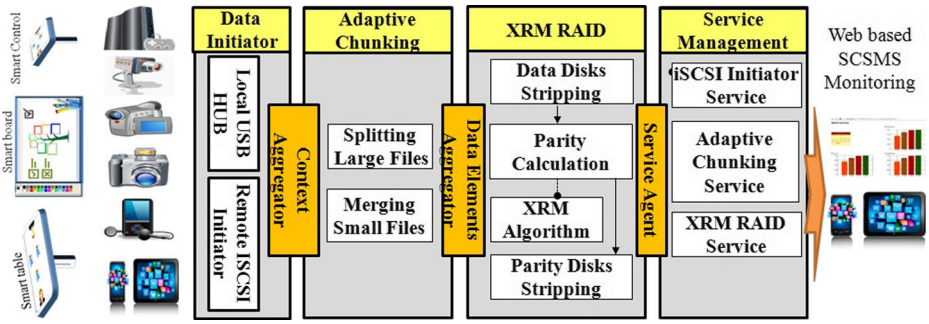

Fig. 2 Overview of the proposed smart classroom storage management system (SCSMS)

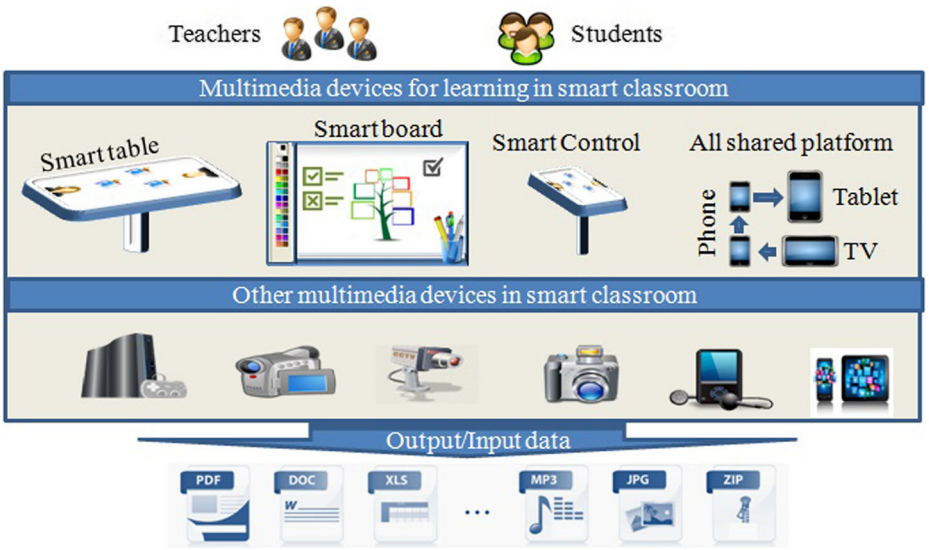

Fig. 3 Data interaction between multimedia devices and users at data initiator for smart classroom

process when the word size is large. Then, the XRM table is applied for parity calculation instead of using real XOR operations. The XRM table stores the results of the XOR operations when the word size is less than four.

Service management Service agent selects the correct storage services based on the current state of multimedia devices and storage system infrastructure for smart classroom.

4 The proposed storage features for multimedia devices in smart classroom

Energy consumption and IO performance in a SAN system highly depend on the number of XOR operations to generate parity data and the number of updates, erases, and read and write operations. The target storage server consists of an iSCSI target, adaptive chunking, XRM-RAID and flash array storage. Figure 4 shows the overall architecture of the proposed SAN system. The proposed SAN system provides storage space for multimedia devices such as smart camera, smart table, smart board, smart TV, tablet, smart phone and personal computer which used in smart classroom.

4.1 Adaptive chunking

The proposed adaptive chunking arranges small files in the separated workload by doing an in-place update. The in-place update requires information such as workload type, file size, chunk size, IO buffer size, file system type and volume size. The flash-based target storage system enhances IO performance and energy efficiency by decreasing the number of updates and read/write operations when the workload has many files that are smaller than the buffer size. Therefore, we propose adaptive chunking to decrease the number of read/write operations by merging small files into a chunk where the size is equal to the buffer size. And

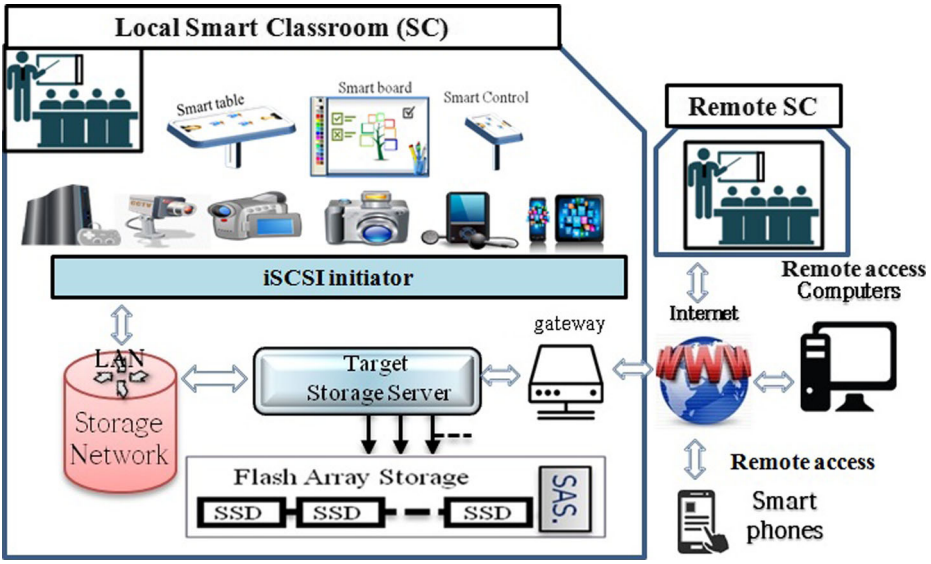

Fig. 4 The proposed SAN for smart classroom

so, adaptive chunking also splits a file into many chunks when the size is equal to or bigger than the buffer size. As a result, the internet small computer system interface (iSCSI) initiator transfers chunks into the iSCSI target server through a storage network appliance. In adaptive chunking, workload F={ f_1, f_2, f_3, ..., f_m, f_{m+1}, ..., f_{last}} is classified into workload F_m={ f_1, f_2, f_3, ..., f_m} and workload F_r={ f_{m+1}, f_{m+2}, ..., f_{last}} ($F = F_m \bigcup F_r$ and $= F_m \bigcap F_r$). In workload F_m, many small files are merged into one chunk using adaptive chunking, and in workload F_r, a file will be split into many chunks. Let us set the chunk size and threshold value as a maximum buffer size. In Fig. 5, the proposed method classifies workload F into F_r and F_m workloads based on the threshold value. And so, the proposed adaptive chunking merges several small files in F_m into a chunk that eliminates the small IOs and splits several large files F_r into many chunks, which prevents the last part of the chunk from being filled with zero bits. This paper also measures the effect of adaptive chunking in the SCSMS application where m denotes the number of files with a size less than the threshold, and r denotes the number of files with a size bigger than the threshold.

$$\gamma'_{read} = \left(\sum_{i=1}^{last} \frac{sizeoff_i}{threshold} \right). \tag{2}$$

Note that, γ'_{read} denotes the number of reads using adaptive chunking in (2). In the proposed method, the number of reads becomes less than the traditional method ($\gamma_{read} \geq \gamma'_{read}$). As a result, for input workload F, we have a condition where, as the m value increases, then γ'_{read} significantly decreases.

4.2 XRM-RAID

iSCSI is an Internet protocol based storage networking standard for connecting data storage equipment. In an iSCSI target, a logical unit number (LUN) is a storage device number addressed by the SCSI protocol in the storage network. Figure 5 also shows the proposed

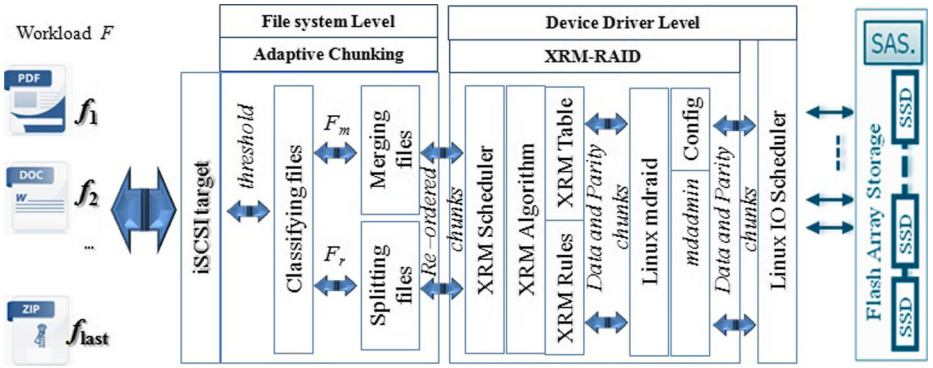

Fig. 5 The proposed iSCSI target storage server architectures

XRM-RAID structure and how it interacts with Linux mdraid device drivers. XRM-RAID is proposed to reduce the execution time of XOR operations and memory requirements when calculating the parity data. The XRM scheduler will write and read information by stripping through Linux mdraid device driver from/to the flash array storage. The first step is to calculate the sequences of XOR operations in HVPC erasure codes. The second step is to avoid performing a real XOR operation for each pair of sequences by applying XRM rules or retrieving data from the XRM table. The complexity of the XRM table calculation increases when word size increases. Therefore, XRM rules are presented to restrict the search scope and overcome the complexity of the XRM table calculation. We proposed XRM rules (1-6) in our previously research [16]. However, XRM rules (7-9) and XRM algorithm are proposed in this paper.

XRM rules and XRM table Figure 6 gives an example of the XRM table generated from the possible XOR operations between operands a and b when the word size is equal to four. The properties of the XRM table are analyzed, and these properties are used to adopt encoding processes for various ranges of storage system scales. Let w^2 be the size of a binary data block, where $w = 4, 8, 16, ..., 1024, 2048$. Each w^2 binary block can be represented as an equivalent decimal number a, b and c, where operands a,b and $c \in \{0, 1, ..., w^2 - 1\}$.$a \oplus b = c$, where c or $C_{a,b}$ is the result of an XOR operation between the operands a and b. Based on these general conditions, the XRM rules are extracted as follows:

Rule 1 : if $b = 0$, and $0 < a < 2^w - 1$, then $a \oplus b = a$.
Rule 2 : if $a = 0$, and $0 < b < 2^w - 1$, then $a \oplus b = b$.
Rule 3 : if $a = b$, and $0 < (a,\text{and } b) < 2^w - 1$, then $a \oplus b = 0$.
Rule 4 : if $a + b = 2^w - 1$, and $0 < (a,\text{and } b) < 2^w - 1$, then $a \oplus b = 2^w - 1$.
Rule 5 : if $a = 2^w - 1$, and $0 < b < 2^w - 1$, then $a \oplus b = 2^w - 1 - b$.
Rule 6 : if $b = 2^w - 1$, and $0 < a < 2^w - 1$, then $a \oplus b = 2^w - 1 - a$.

Rule 7 : The result location, $C_{a,b}$, can be searched among four parts of (A), (B), (C) and (D), where if a or b is less than 2^{w-1}, the result location is in parts (B) or (C). On the other hand, if a and b are larger than 2^{w-1}, the result location is in part (D).
Rule 8 : As shown in Fig. 6, the symmetric characteristics divide the XRM table into (A), (B), (C) and (D) parts. Parts (A), (B), (C) and (D) have the same XOR operation

a\b	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	3	2	1	A	7	6	5	4	11	10	9	B	15	14	13	12
4	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	11	10	9	C	15	14	13	12	3	2	1	D	7	6	5	4
12	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fig. 6 XRM table, where $w = 4$

result value. Using this benefit, the size of the XRM table can be reduced to 50 % less than the actual size. The result from part (D) can be relocated to part (A) by applying $a = a - 2^{w-1}$ and $b = b - 2^{w-1}$.

Rule 9 : As shown in Fig. 6, parts (A) and (C) or parts (A) and (B) have a similar pattern. By subtracting an offset, 2^{w-1}, the same result value of their XOR operations can be obtained. Therefore, the search scope of the XRM table can be narrowed down by applying $a = a - 2^{w-1}$ or $b = b - 2^{w-1}$.

- If $C_{a,b}$ is located in part (B) of the XRM table,
  $C_{a,b} = C_{a-2^{w-1},b} + 2^{w-1}$;
  where, $2^{w-1} = C_{a,b} - C_{a-2^{w-1},b}$;

- If $C_{a,b}$ is located in part (C) of the XRM table,
  $C_{a,b} = C_{a,b-2^{w-1}} + 2^{w-1}$;
  where, $2^{w-1} = C_{a,b} - C_{a,b-2^{w-1}}$;

XRM algorithm As shown in Table 2, the XRM algorithm uses symmetric rules 1 to 9 to calculate the parity chunk. The XRM algorithm notations, functions and details are described as follows. Recall that for a given word size, w, and the number of data disks, n, there are n-1 sequences of XOR operations between the data chunks within a horizontal strip consisting of two decimal data blocks, P and D_{j+1}. Note that results of each XOR operation are stored in a parity chunk P, where P denotes a corresponding parity chunk. For each iteration, by applying rules 7, 8 and 9 instead of performing an XOR operation, carry is calculated to set the relocation offset. The XRM algorithm is built on three functions.

Table 2 The pseudo-code of the XRM algorithm

Algorithm: XRM($D[n], n, w$)

Input

$D[n]$: data chunks
n: the number of data disks
w: word size
$carry$: an offset value

Output

P: parity chunk

```
1. carry = 0; P = D_1; ws = w;
2.    for (j = 1; j ≤ n; j++) /* n-1 number of XOR operations */
3.       while ( w ≥ 4)
4.             if( XRULES(P, D_{j+1}, w) is not NULL) /* Rules 1-6*/
5.                   P= XRULES(P, D_{j+1}, w);
6.                   break; /* break the inner loop */
7.             else if (w > 4)
                     /* XRM Rules 7 and 9 */
8.                   if( Location (P, D_{j+1} ) is equal to Part B or Part C )
9.                         D_{j+1} = D_{j+1} − 2^{w−1};
10.                        carry = carry + 2^{w−1};
11.                  P = P + carry;
                     /* XRM Rules 7 and 8 */
12.                  elseif( Location (P, D_{j+1} ) is equal to Part D )
13.                        P = P − 2^{w−1};
14.                        D_{j+1} = D_{j+1} − 2^{w−1};
15.                  w − −; /* Narrow down search scope */
16.            else if (w = 4) P = XTable (P, D_{j+1}, w);
17.      endwhile
18.      w = ws; /* Reset the word size value */
19.   endfor
20. returns P;
```

- XRULES(a, b, w). The function applies rules 1 to 6 for given input decimal values, a, b and word size w. The function outputs the results of the XOR operation between a and b, or NULL if the given input values are not valid.
- XTable(a, b, w). The function searches the XRM Table for given input decimal values, a, b and word size w, and outputs the result of the XOR operation between a and b, or NULL if the rules are not applied to the given input values.
- Location(a, b). The function outputs the location of the XOR operation result between a and b, where it can be in parts (B), (C) or (D) with the corresponding values, B, C or D, for given input decimal values, a, b. Note that this function is based on XRM rule 7.

Table 2 lists the pseudo-code of the XRM algorithm using the rule-based erasure coding mechanism. The algorithm retrieves the parity chunk P from a sequence of XOR operations between the data chunks in a strip. To achieve this, the XRM algorithm uses the function XRULES, the function Location for symmetric rules 7, 8 and 9, and the function XTable

instead of performing real XOR operations. Variables in line 1 are initialized. During the n-1 number of XOR operations among n data chunks, the parity chunk is obtained. For each XOR operation, there are three conditions, of which only one needs to be satisfied.

The first condition in lines 4 and 5 apply rules 1-6 using the XRULES function, and the result of XOR will be stored in parity chunk P. In the second condition located in lines 7-15, XRM symmetric rules 7, 8 and 9 are applied, while word size is more than four. During each cycle of the while iteration, as the value of word size decreases, the search scope of XRM is narrowed until the parity chunk P is obtained. Under this condition, an attempt is made to determine the location of P in the XRM Table from four different parts by applying rule 7 using the Location function. In lines 8-10, P location is relocated using rule 9 by decreasing the P or D_{j+1} value. In lines 12-14, the P location is relocated by applying rule 8. The word size is then reduced, and the value of P is updated.

In the last condition located in line 16, if XRM rules 1-6 are not applied and $w = 4$, the value of P can be retrieved from the XRM table using the XTABLE function. In line 18, after each while loop is done, we need to reset the word size. Finally, in the last line, the parity chunk P is returned.

XRM scheduler The erasure codes protect data from failure in storage systems by reconstructing the lost data. Data failures occur for various reasons, such as disk failure, sector failure and component failure [1, 11, 13, 15]. Encoding is also a calculation of coding information from the actual data and generating parity and data elements [1]. Table 3 describes the pseudo-code of the proposed XRM scheduler at the target storage server. In the first loop, the XRM scheduler reads chunks from the iSCSI target. Note that chunks are arranged using adaptive chunking at the initiator server. In the second iteration, data chunks are written into data SSDs in advance. Then, the XRM algorithm is used to generate parity chunks from data chunks. In particular, the XRM rules and table are applied to reduce energy consumption and the time complexity of erasure coding. Finally, parity chunks are written into parity SSD, respectively. The throughput of such a storage system is typically described by the rate between strip size and encoding time in (3).

$$Throughput_{write} = \frac{workloadsize}{encodingtime}. \tag{3}$$

$$E_{encoding} = t_{waken}e_{waken} + t_{active}e_{active} + t_{idle}e_{idle} + t_{coding}e_{coding}. \tag{4}$$

$$t_{active} = t_{read} + t_{write}. \tag{5}$$

$$E_{decoding} = t_{waken}e_{waken} + t_{active}e_{active} + t_{idle}e_{idle} + t_{coding}e_{coding}. \tag{6}$$

In terms of energy efficiency in the SSD-based IO scheduler, SSDs have various power modes, such as ON, OFF, $sleep$, $active$ and $idle$. The SSD stays in $active$ mode during read and write operations, and during the rest of the time, the SSD will automatically be in $idle$ mode. The proposed technique removes many small random read/write operations and reduces the number of CPU cycles to encode and decode data. The total energy cost of encoding $E_{encoding}$ is calculated in (4) and (5), where t_{waken} , t_{idle} , t_{coding} , and t_{active} are denoted as the time to wake up the SSD, the idle time, the time to generate parity, and active time, respectively. The total energy cost of decoding $E_{decoding}$ is calculated in (6). The power consumption for $waken$, $active$ and $idle$ modes are denoted as e_{waken} , e_{active}, and e_{idle}. And so, the power consumption of coding and decoding (e_{coding},

Table 3 The pseudo-code of the proposed XRM scheduler

Algorithm: XRM Scheduler(Data chunks, $last, w, n$**)**

Input

Data chunks: a data file with a name and type.
$last$: Total number of data chunks after adaptive chunking.
β: The number iteration is equal to $\beta = \frac{last}{n}$.
$CH[\beta][n]$: Two dimensional decimal data chunks;
w: Word size;
n: The number data disks;
$Parity[\beta]$: One-dimensional decimal parity chunks;

```
                    /* Data chunks initialization */
1.   for (i = 1; i ≤ β; i++) /*
2.     for (j = 1; j ≤ n; j++) /*
           /* Adaptive chunking merges and splits data into chunk array
           CH[i][j] with an equal chunk size */
3.         read Data chunks from host and allocate it into CH[i][j].
4.     endfor
5.   endfor
     /* Parity creation using XRM algorithm and IO redirection */
6.   for (i = 1; i ≤ β; i++) /*
7.     for (j = 1; j ≤ n; j++) /*
           /* Stripping data chunk through Linux mdraid device driver */
8.         write CH[i][j] into Data SSD[j].
9.     endfor
10     Parity[i]= XRM(CH[i], w, n). /* Parity creation */
       /* Stripping parity chunk through Linux mdraid device driver */
11.    write Parity[i] into Parity SSD.
12.  endfor
```

$e_{decoding}$) depends on the number of XOR operations and CPU cycles. Calculating e_{waken} and t_{waken} are excluded from ($E_{encoding}$, $E_{decoding}$) because SSD does not stay in awake mode during the encoding and decoding processes.

5 System design and implementation

XRM-RAID is implemented under the CentOS operating system using open source Jearsure code software [11, 13, 15]. Figures 7, 8 and 9 show the detail implementation of SCSMS application. Figure 7 shows screen shot of monitoring application when smart phone initiator is connected to the target server. Figure 8 shows screen shot of monitoring application when the iSCSI target is running successfully. Finally, Fig. 9 shows web based SCSMS application for displaying the read and write performance, and energy consumption at selected date. Figure 10 shows the prototype of the proposed smart classroom using SCSMS application. This prototype initializes various multimedia devices through directed attached USB connection and iSCSI remote connection. Figure 11a shows the target server hardware specifications. Figure 11b shows the SSD specifications for flash array storage.

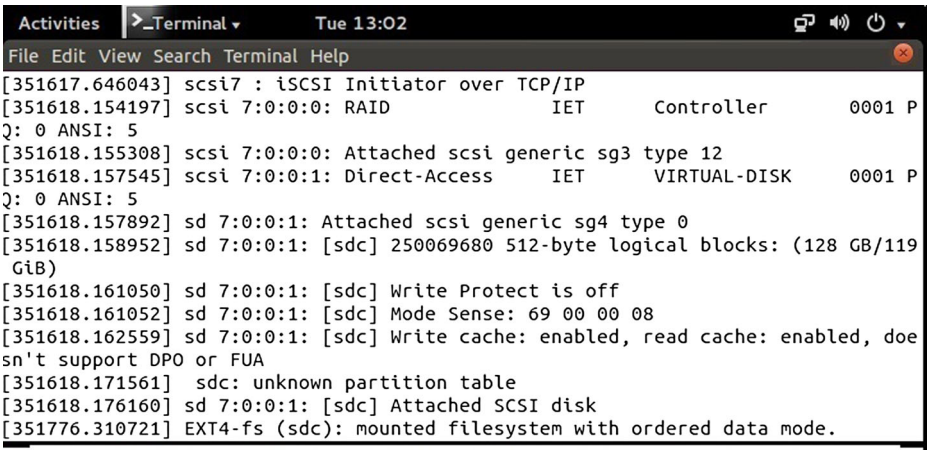

Fig. 7 SCSMS application Initiator server for monitoring IO performance and energy efficiency

Figure 11c lists the VA1, GC2 and SPC1 trace workloads with specific parameters.VA1 is used for smart camera and smart TV. GS2 is used for game console and SPC1 is used for smart phone.

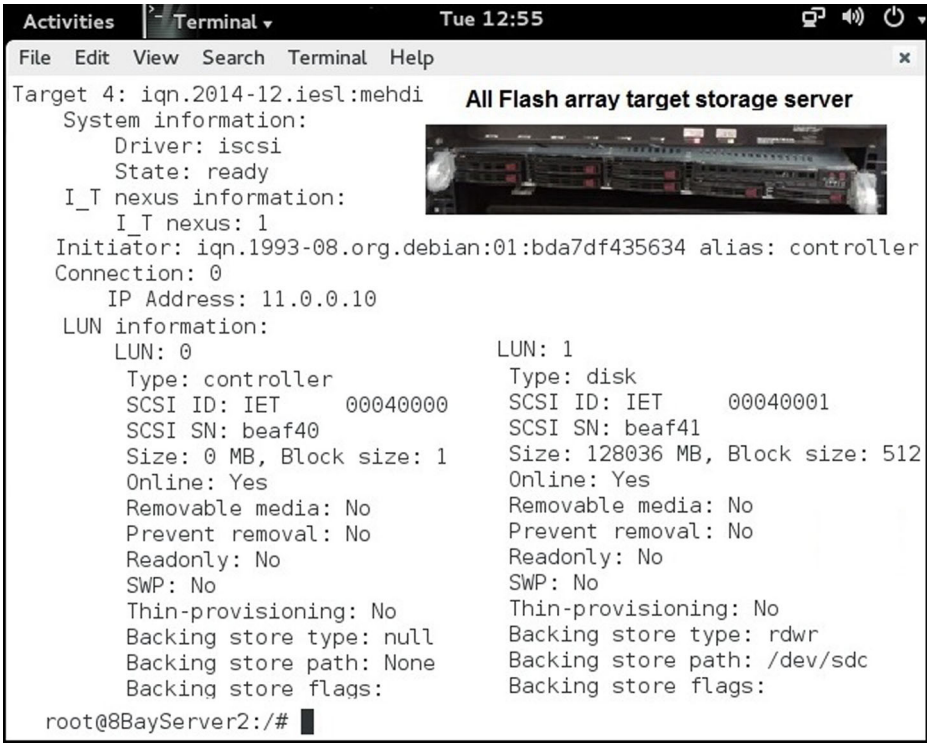

Fig. 8 SCSMS application using target server for monitoring IO performance and energy efficiency

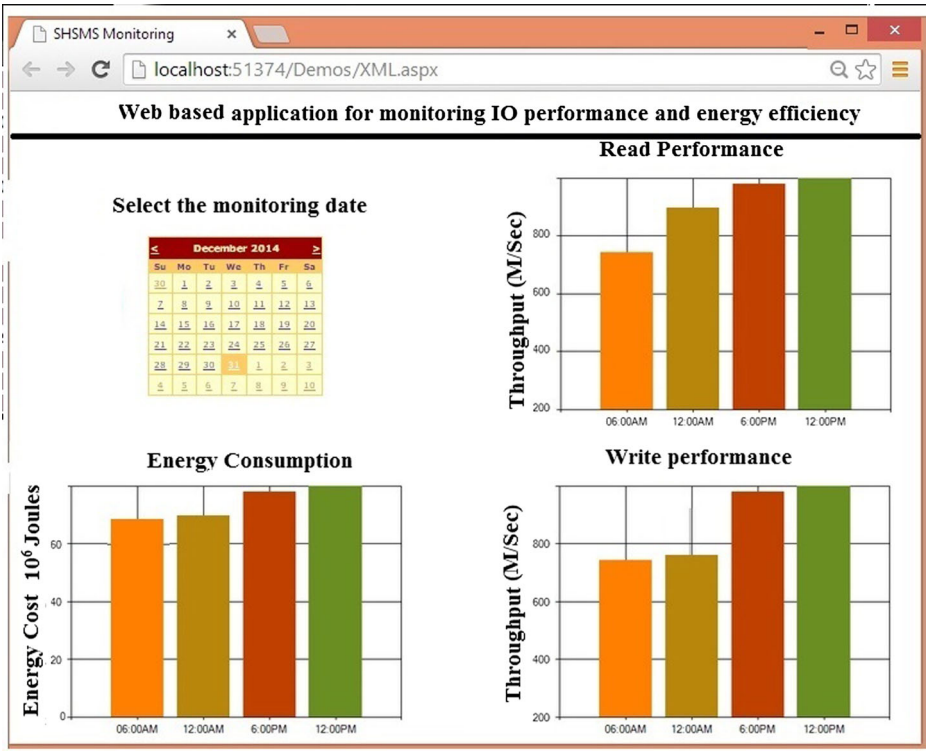

Fig. 9 Web based SCSMS application for monitoring IO performance and energy efficiency

6 Experimental results

We have implemented a prototype of smart classroom using SCSMS application. The performance evaluation is conducted on a platform of target storage server withan Inter Xeon 2,0 GHz processor, 32 GB DDR memory and six 64 GB SSDs and two 128 GB SSDs. For given buffer sizes (1 MB, 5 MB and 10 MB), we evaluate read/write performance for various SCSMSs applications. The experimental environment for storage management system for

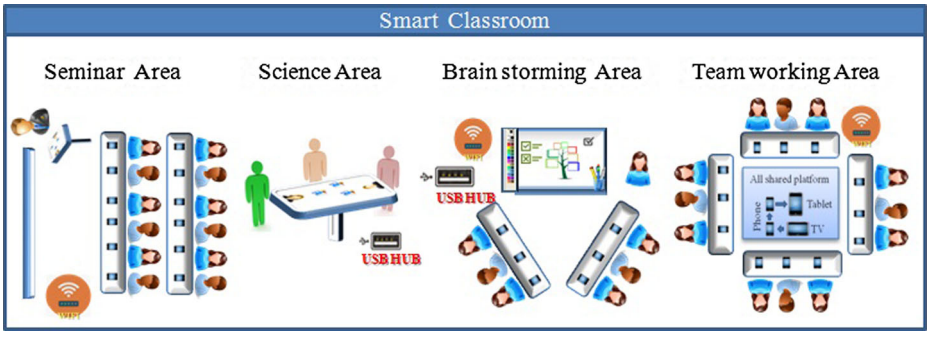

Fig. 10 The prototype of smart classroom using SCSMS application

(a)
TARGET SERVER SPECIFICATIONS

	OS	Memory	CPU
Target Storage Server	Cent OS 6.2	DDR3 32GB	Intel Xeon 2.0 GHz

(b)
SSD SPECIFICATIONS

Manufacturer	e_{active}	e_{idle}	Capacity	Max read	Max write
Samsung 830	0.15 watts	0.08 watts	64 GB	520 MB/s	320 MB/s
Samsung 840	0.07 watts	0.05 watts	128 GB	530 MB/s	390 MB/s

(c)
TRACE LIST WITH RESPECT TO VARIOUS WORKLOADS

Name	Description	LUNs	Total size	Device type
VA1	Video and Audio workload	4	1 GB	Video Camera
GC2	Game Console workload	6	2 GB	Game Console
SPC1	Storage benchmark	2	1 GB	Smart Phone

Fig. 11 SCSMS application experimental environment

smart classroom is simulated by applying SCSMS application into our target and initiator hardware infrastructure as shown in Figs. 7 and 8. As shown in Table 1, detail experimental environment specifications of existing and the proposed SCSMSs are described as follows,

(i) Huang et al. [9] : For given flash array storage, a NAS infrastructure is build up and RAID 6 is implemented using Linux mdraid 6 driver kernel. This SCSMS chunks data into multiple blocks using traditional chunking strategy;
(ii) Kim et al. [19] : Ceph cloud storage is build up using flash array storage. Ceph supports erasure coding and replication mechanism where Ceph chunks data into multiple objects using traditional chunking strategy;
(iii) Scott et al. [18] : A P2P network is build up using flash array storage. Each multimedia device is considered as peer where this SCSMS does not support erasure coding mechanism. It chunks data into multiple blocks using traditional chunking strategy;
(iv) The proposed SCSMS : For given flash array storage, a SAN infrastructure is build up using XRM-RAID and XRM algorithm where it chunks data into multiple blocks using adaptive chunking strategy;

6.1 IO performance results

Figure 12 shows the read performance of various storage management system for smart classroom using given buffer size(1 MB,5 MB or 10 MB) with respect to file sizes(4 KB,16 KB,64 KB,256 KB,1 MB). The average read performance of the proposed SCSMS is improved by 32 %, 45 % and 58 % compared to Huang et al., Kim et al. and Scott et al. with respect to file size and buffer size. The average read performance of the proposed SCSMS for given buffer size(10MB) is improved by 24 % and 52 % compared to those for other buffer sizes(5 MB and 1 MB). This is because the proposed SCSMS can merge many small files into a block as the buffer size increases. However, the average read performance of the traditional SCSMSs (Scott et al. [18], Huang et al. [9] and Kim et al. [19]) for given buffer size(10MB) is improved by 17 % and 32 % compared to those for other buffer sizes(5 MB and 1 MB) due to dedicating larger memory space for buffer.

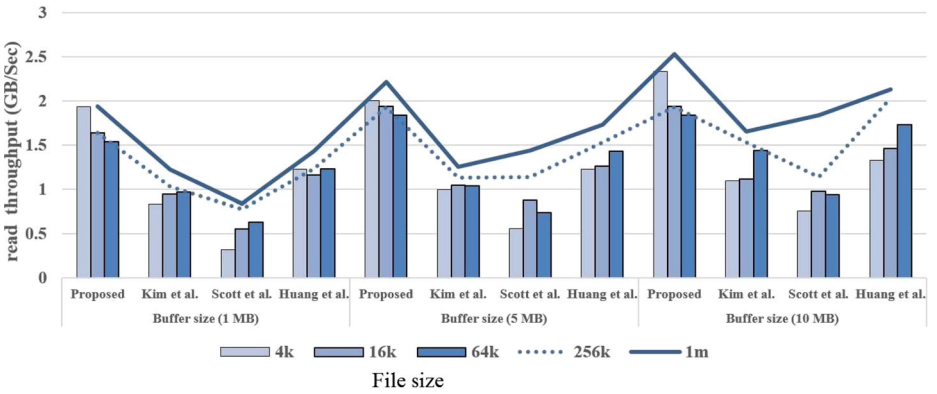

Fig. 12 The read performance for SCSMS applications

Figure 13 shows the write performance of various storage management system for smart classroom using given buffer size(1 MB,5 MB or 10 MB) with respect to file sizes(4 KB,16 KB,64 KB,256 KB,1 MB). The average write performance of the proposed SCSMS is improved by 22 %, 32 % and 56 % compared to Huang et al., Kim et al. and Scott et al. with respect to file size and buffer size. The average write performance of the proposed SCSMS for given buffer size(10MB) is improved by 21 % and 43 % compared to those for other buffer sizes(5 MB and 1 MB)due to increasing ratio of merging small chunks into a larger block. However, the average read performance of the traditional SCSMSs (Scott et al. [18], Huang et al. [9] and Kim et al. [19]) for given buffer size (10MB) is improved by 18 % and 31 % compared to those for other buffer sizes(5 MB and 1 MB) due to larger dedicated memory space.

6.2 Physical IOPS performance results

Figure 14 shows the average number of physical read and write operations per second (IOPS) of various storage management system for smart classroom using given buffer size(1 MB,5 MB or 10 MB) with respect to file sizes(4 KB,16 KB,64 KB,256 KB,1 MB). The average number of physical read and write operations per second (IOPS) of the proposed SCSMS is improved by 68 %, 72 % and 74 % compared to Huang et al., Kim et al. and

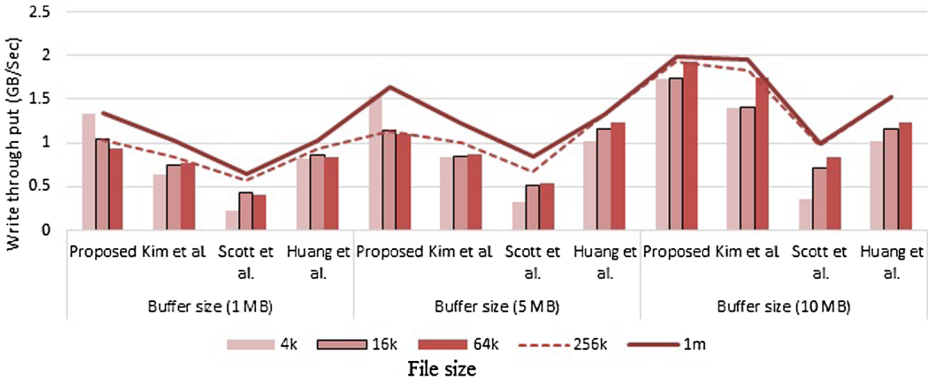

Fig. 13 The write performance for SCSMS applications

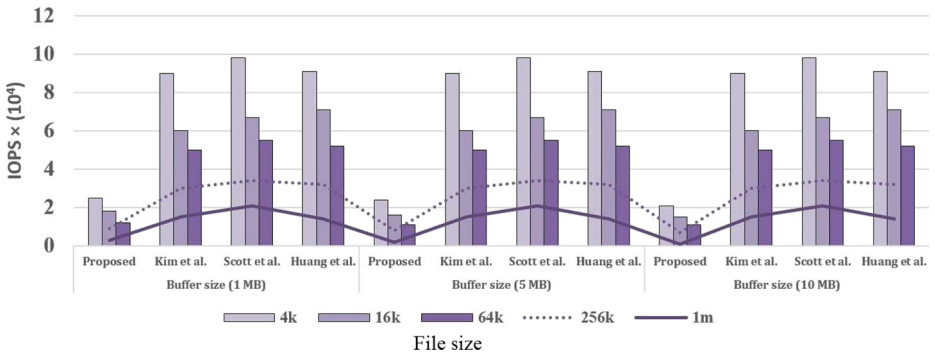

Fig. 14 The physical IOPS performance for SCSMS applications

Scott et al. with respect to file size and buffer size. The average number of physical read and write operations of the proposed SCSMS for given buffer size(10MB) is reduced by 16 % and 48 % compared to for other buffer sizes(5 MB and 1 MB). This is because the proposed SCSMS eliminates small physical read and write operations by merging several small files into a larger chunk. However, the average number of physical read and write operations of the traditional SCSMSs (Scott et al. [18], Huang et al. [9] and Kim et al. [19]) buffer size (10MB) is 14 % and 21 % lower compared to those for other buffer sizes(5 MB and 1 MB). Note that the effect of buffer size is limited in traditional SCSMSs when workload composes of many small files.

6.3 Energy consumption results

Figure 15 shows the energy consumption of various storage management system for smart classroom using given buffer size(1 MB,5 MB or 10 MB) with respect to file sizes(4 KB,16 KB,64 KB,256 KB,1 MB). The energy consumption of the proposed SCSMS is reduced by 32 %, 42 % and 58 % compared to Huang et al., Kim et al. and Scott et al. with respect to file size and buffer size. The energy consumption of the proposed SCSMS for given buffer size(10MB) is reduced by 8 % and 23 % compared to those for other buffer sizes(5 MB and 1 MB). This is because the proposed SCSMS has steady energy consumption regardless of file size. In XRM-RAID 6, t_{active} decreases as the number of data chunks decreases, and t_{coding} decreases as the number of XOR operations decreases.

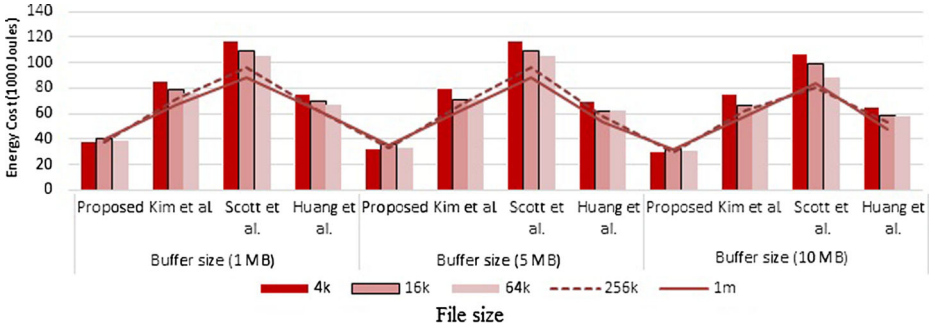

Fig. 15 Energy consumption for SCSMS applications

7 Conclusion

Smart classroom requires lower energy consumption and faster performance storage area network to store data which are created from various multimedia devices. This paper builds up a smart classroom storage management system using flash array in a classroom SAN and presents an adaptive chunking and XRM-RAID technique for various multimedia devices. Adaptive chunking removes many small read/write operations to encode and decode data. In the proposed SCSMS, XRM-RAID reduces the number of XOR operations by providing an XRM scheduler to generate parity data and also to break down the XOR complexity of Linux mdraid for SCSMS application. Experimental results show that the energy consumption of the proposed SCSMS is improved by 32 %, 42 % and 58 % compared to Huang et al., Kim et al. and Scott et al. with respect to file size and buffer size. In terms of the average read throughput, the proposed SCSMS has higher performance by 32 %, 45 % and 58 % compared to Huang et al., Kim et al. and Scott et al. with respect to file size and buffer size.

Acknowledgments This work was supported in part by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ICT/SW Creative Research program (NIPA-2014-H0502-14-3002) supervised by the NIPA(National IT Industry Promotion Agency) and in part by the National Research Foundation of Korea(NRF) Grant funded by the Korean Government(MOE) (2013R1A1A2006912) and in part by Inha University Research Grant.

References

1. Blaum M, Brady J, Bruck J, Menon J (1995) EVENODD: An Efficient Scheme for Tolerating Double Disk Failure in RAID Architectures. IEEE Trans Comput 2:44
2. Cooley JA, Mineweaser JL, Servi LD, Tsung ET (2003) Software-based erasure codes for scalable distributed storage. In: Proc. 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies. (MSST 2003), pp 157–164
3. Gill K, Yang S-H, Yao F, Lu X (2009) A zigbee-based home automation system. IEEE Trans Consum Electron 55(2):422–430
4. Greenan KM, Li X, Wylie JJ (2010) Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. In: IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp 1–14
5. Hafner JL (2005) WEAVER codes: highly fault tolerant erasure codes for storage systems. In: Proc. of the USENIX Conference on File and Storage Technologies (FAST'05), vol. 4. CA, USA, p 16
6. Hafner JL, Deenadhayalan V, Rao KK, Tomlin A (2005) Matrix methods for lost data reconstruction in erasure codes. In: Proc. of the USENIX Conference on File and Storage Technologies (FAST'05), San Francisco, USA, pp 183–196
7. Han D-M, Lim J-H (2010) Smart home energy management system using IEEE 802.15.4 and zigbee. IEEE Trans Consum Electron 56(3):1403–1410
8. Han D-M, Lim J-H (2010) Design and implementation of smart home energy management systems based on zigbee. IEEE Trans Consum Electron 56(3):1417–1425
9. Huang T-C, Chang D-W (2013) TESA: a temporal and spatial information aware writeback policy for home network-attached storage devices. IEEE Trans Consum Electron 11(1):122–129
10. Jiang S, Ding X, Chen F, Tan E, Zhang X (2005) DULO: an effective buffer cache management scheme to exploit both temporal and spatial locality. In: Proc. of the USENIX Conference on File and Storage Technologies(FAST'05), vol. 4. CA, USA, pp 8–8
11. Kenchammana D, He D, Hafer JL (2005) REO: A generic RAID Engine and Optimizer. In: Proc. of the USENIX Conference on File and Storage Technologies(FAST 07). San Francisco, USA, pp 261–276
12. Khan O, Burns R, Plank J, Pierce W, Huang C (2012) Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads. In: Proc. of the USENIX Conference on File and Storage Technologies, San Jose, CA
13. Kim J, Oh Y, Kim E, Choi J, Lee D, Noh SH (2009) Disk schedulers for solid state drivers. In: Proc. of ACM international conference on Embedded software (EMSOFT '09). NY, USA, pp 295–304

14. Li Y et al. (2013) Energy-Aware Storage. In: Proc of the USENIX Conference on File and Storage Technologies
15. Luo J, Xu L, Plank JS (2009) An efficient XOR-scheduling algorithm for erasure codes encoding. In: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '09), pp 504–513
16. Pirahandeh M, Kim D-H (2012) Adopted erasure code for SSD based RAID-6 System. In: Proc. of the ITC-CSCC conference. Sapporo, Japan, pp 81–85
17. Plank S (2011) XORs Lower Bounds and MDS Codes for Storage. IEEE Information Theory Workshop, Brazil, pp 529–551
18. Scott K et al. (2010) Context-Aware Services writeback policy for home network-attached storage devices. IEEE Trans Learn Technol 3(3):214–227
19. Svetlana K, Song S-M, Yoon Y-I (2011) Smart Learning Services for Smart Learning Spaces. IEEE Sensors J 11:7835–7850
20. Won Y et al. (2007) Energy-aware disk scheduling for soft real-time I/O requests, vol 13. Springer, Multimedia System, pp 409–428
21. Xie T (2008) SEA: A Striping-Based Energy-Aware Strategy for Data Placement in RAID-Structured Storage Systems. IEEE Trans Comput 57(6):748–761

Mehdi Pirahandeh studied BS in computer and system science, Boras University in Sweden. He is an integrated M.S. and PhD student at Inha University in South Korea. His research interests include embedded systems, cloud storage systems, social networking and e-health systems.

Deok-Hwan Kim received a M.S. and PhD from the Korea Advanced Institute of Science and Technology. He is a professor at Inha University in Korea. His research interests include embedded systems, storage systems, cloud systems, multimedia systems and brain computer interfaces.