# Application Sensitive Energy Management Framework for Storage Systems

Norifumi Nishikawa, Miyuki Nakano, *Member, IEEE*, and Masaru Kitsuregawa, *Fellow, IEEE*

**Abstract**—Rapidly escalating energy and cooling costs, especially those related to the energy consumption of storage systems, have become a concern for data centers, primarily because the amount of digital data that needs storage is increasing daily. In response, a multitude of energy saving approaches that take into account storage-device-level input/output (I/O) behaviors have been proposed. The trouble is that numerous critical applications such as database systems or web commerce applications are in constant operation at data centers, and the conventional approaches that only utilize storage-device-level I/O behaviors do not produce sufficient energy savings. It may be possible to dramatically reduce storage-related energy consumption without degrading application performance levels by utilizing application-level I/O behaviors. However, such behaviors differ from one application to another, and it would be too expensive to tailor methods to individual applications. As a way of solving this problem, we propose a universal storage energy management framework for runtime storage energy savings that can be applied to any type of application. The results of evaluations show that the use of this framework results in substantive energy savings compared with the traditional approaches that are used while applications are running.

**Index Terms**—Application I/O behavior, logical I/O pattern, storage systems, runtime storage energy saving, universal storage energy management framework

## 1 Introduction

The amount of digital data produced by human beings is increasing every day. According to a recent International Data Corporation (IDC) report [1], the amount of information created and stored digitally will exceed 40,000 exabytes by 2020. This *Big Data* must be managed and used in conjunction with data-intensive applications such as sensor data archives, search engines, and web services. Furthermore, since all this data must be properly stored, the capacity of storage systems will increase as well. It has been predicted [2] that data storage capacities in 2017 will need to be 6.5 times as large as those available in 2011.

The energy consumption of information technology (IT) equipment at data centers is growing on a daily basis [3], and a recent report [4] states that storage system energy consumption rates, as a portion of all IT equipment, will increase even further in the near future because the amount of digital data requiring storage will continue to inflate. For example, the paper [5] reports that the power required to handle data storage in large online transaction processing systems (OLTPs) and other data-intensive applications accounts for more than 70 percent of the total power consumption of IT equipment. Therefore, it is clear that storage system energy consumption has to be reduced at large data centers.

There are numerous energy saving approaches that use energy-efficient storage devices, such as large-capacity hard disks (HDDs), 2.5 inch HDDs, or solid state disks (SSDs), or that use space-efficient improvement methods such as data compression or data de-duplication [6]. In contrast with previous energy saving approaches, our paper focuses on enabling dynamic storage energy savings while applications are running.

The storage energy saving methods that have been proposed to date include the one reported in [7], [8], which has a massive arrays of inactive disks (MAID) function that stops HDD rotation when no I/Os are issued to the HDDs [9].

Data centers have various applications in continuous operation, and the I/O behaviors of those applications, such as file servers, OLTPs, decision support systems (DSSs), and web commerce, differ considerably. Traditional energy saving methods are typically implemented inside storage devices, which means their energy saving abilities can only be determined from their storage-level I/O behaviors. There is also the possibility that energy saving methods based on such behaviors could degrade the performance of applications since storage devices have no ability to detect or determine application runtime behaviors.

Our major contribution in this paper is the proposal of a novel energy management framework that can achieve energy savings for storage devices even when different applications are in continuous operation. Furthermore, we introduce a universal framework that utilizes application logical level I/O behaviors without the necessity of implementing a tailored energy management policy for each application. We show that our framework is applicable to a range of systems, from concise compact multiple-HDD systems to large enterprise storage systems, even when various applications are in continuous operation.

Our framework statistically analyzes both application- and storage-device-level I/O behaviors and identifies how the I/O behaviors from each application relate to the

• *The authors are with the Institute of Industrial Science, University of Tokyo, Meguro-ku, Tokyo 153-8505, Japan.*
*E-mail: {norifumi, miyuki, kitsure}@tkl.iis.u-tokyo.ac.jp.*

*Manuscript received 16 Nov. 2013; revised 31 Dec. 2014; accepted 5 Mar. 2015. Date of publication 26 Mar. 2015; date of current version 3 Aug. 2015. Recommended for acceptance by C.-Y. Chan.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TKDE.2015.2416737*

1041-4347 © 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

storage devices. Our framework introduces a *logical I/O pattern* in order to manage application-level I/O behaviors regardless of the application type and to determine runtime energy saving strategies suitable for the logical I/O pattern dynamically. The features of our framework are as follows:

- By introducing the logical I/O patterns, our method has a high level of applicability to all applications, regardless of their I/O behaviors, which allows it to achieve substantial energy savings compared with traditional storage-device-level energy saving methods.
- It uses the logical I/O patterns to enable the execution of more energy-efficient data placement than traditional methods do.

In this paper, we quantitatively evaluate our framework using typical data center applications. To accomplish this, we run actual file server, OLTP, DSS, and web commerce applications on multiple HDD and large storage system environments. We compare our method with two established storage-device-level energy saving methods, popular data concentration (PDC) [10] and dynamic data reorganization (DDR) [11].

Section 2 presents the concepts behind our universal storage energy management framework. Section 3 introduces the logical I/O patterns. Section 4 describes the storage device model and its power status, while Section 5 discusses our monitoring functions. We describe the storage energy management flow in Section 6 and the runtime energy management functions in Section 7. In Section 8, we describe the application of our storage energy management framework to actual systems and analyze the I/O characteristics of four data intensive applications in Section 9. Section 10 shows the results of an evaluation, while Section 11 presents related work. Conclusions are given in Section 12.

## 2 Universal Storage Energy Management Framework Concept

Various critical applications such as OLTP or web commerce are in constant operation at data centers. As discussed in the previous section, the I/O behaviors of different applications vary quite a bit. However, the currently used energy saving approaches for storage devices do not take into consideration the differing characteristics of each application's I/O behaviors. If the characteristics of an application's I/O behaviors are known, it is reasonable to expect that an energy saving method could be designed without degrading the application's performance levels. Unfortunately, it is difficult to coordinate an application's level of I/O behavior with energy saving efforts by utilizing the I/O behaviors of the storage devices alone. In response to this difficulty, we develop a novel universal storage energy management framework that enables the utilization of I/O behaviors from various applications by managing them in a uniform way.

### 2.1 Subjects of Runtime Storage Energy Saving

#### 2.1.1 Problems of Previous Storage Energy Saving

Currently, a number of energy saving methods for storage devices are in use or under consideration, most of which can be roughly classified into two approaches. One approach works to extend the duration of the idle period by controlling the I/O interval to storage devices [12], [13], [14], [15], while the other works to extend the idle period by replacing data amongst storage devices [10], [11], [16], [17], [18], [19], [20].

These methods use fixed energy saving functions and formulate their strategies based on the storage-device I/O behavior levels. However, they do little to reduce energy consumption from the standpoint of runtime application behavior. For example, it is difficult to accurately determine which blocks on a storage device are updated from OLTP applications, or which are being read from E-commerce applications. This is because such applications read or update data blocks from the viewpoint of their databases or files, while storage devices have no ability to take into account such viewpoints. Thus, the traditional energy saving methods have no means of correctly determining which blocks should be kept or pre-loaded into the buffer spaces of storage devices. This indicates that it is necessary to consider how to utilize application I/O behaviors to reduce the energy consumption of storage devices.

#### 2.1.2 Universal Storage Energy Management Framework Based on Application I/O Behaviors

Because I/O behaviors differ significantly among all applications, not simply those concerned with OLTP and DSS, any potentially useful energy saving method will need to be applied differently in order to take those behaviors into account. For OLTPs, continuously writing I/O master table updates into buffers may extend the intervals between actual I/O interactions with storage devices, while in the case of DSS, pre-loading and maintaining the tables that are most commonly accessed may likewise extend the read I/O intervals with storage devices. To achieve significant reductions, the energy saving method should be carefully chosen and applied to storage devices while taking into full consideration the differences in the I/O behaviors of these applications. Our framework is designed to manage storage-device energy consumption in a uniform way, despite I/O application behavior differences.

### 2.2 Framework for Runtime Storage Energy Saving

Critical data center applications cannot be stopped once they have entered service. This means a runtime storage energy saving method is important. Furthermore, as discussed in Section 2.1.1, it is difficult to reduce the energy consumption of storage devices that are accessed frequently from applications such as OLTP or those related to web commerce.

With this in mind, we propose a universal storage energy management framework for reducing the energy consumption of storage devices while ensuring that critical applications can continue normal operations. Specifically, by utilizing application-level I/O behaviors for conserving storage device energy, we can introduce a middleware layer monitoring function that works in cooperation with a monitoring function at the storage-device level. Here, we introduce a logical I/O pattern that can easily handle a variety of application types and can analyze the applications' I/O behaviors in a uniform way. By monitoring both application- and storage-device-level I/O behaviors, we can utilize

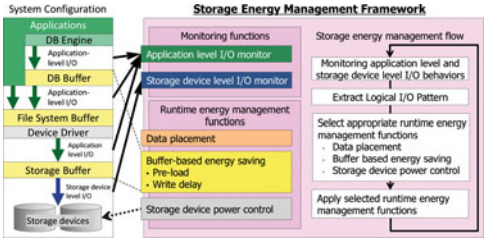



Fig. 1. Universal storage energy management framework.

a buffer in a different layer, such as an application or a storage device, in a uniform way.

Fig. 1 shows an overview of our framework. The left part of the figure shows the system configuration of a database server, or file server, that we suppose. The right part shows the functions and storage-device energy management flow. As can be seen in the figure, our framework has monitoring functions for both application and storage device levels, along with an energy management function that contains a data placement function, a buffer-based energy saving function, and a storage device power control function.

Our framework collects application- and storage-device-level I/O behaviors during operation by utilizing monitoring functions, which it then combines and analyzes to extract logical I/O patterns. The logical I/O patterns are defined in Section 3.

Our framework selects an appropriate energy-reduction function to apply to the storage devices, changes the data placement in the storage devices, and executes the selected strategy to control the storages energy by using a target system buffer. Application I/O behaviors may change when the number of users or frequency of data accesses change. As can be seen from the flow in Fig. 1, our framework captures changes in the I/O behaviors of applications as changes in logical I/O patterns and reselects appropriate functions to reduce storage energy. As a result, it always efficiently reduces energy consumed by storage devices.

## 3 Logical I/O Pattern

In order to identify application I/O behaviors from the storage device energy reduction aspect in a uniform way, we introduce the concepts of data items and logical I/O patterns.

### 3.1 Data Items

A data item is a logical unit of data suitable for efforts to conserve energy. Data items are designed as fragments of application data on a single storage device. These units of data differ from each other on the basis of application type. A unit of data may be a table or a database index in database management system (DBMS) applications such as OLTP or decision support systems. In contrast, a unit of data may be a file in applications that run on file servers. If various pieces of data lie across multiple storage devices, we treat these partitions as different data items. Thus, by splitting data into data items, we can identify the application data I/O behaviors on the storage devices.

### 3.2 Logical I/O Patterns

A logical I/O pattern is a universal indicator of the energy saving potential of a data item. It is also necessary to consider the energy saving potential of each data item. While there are many I/Os issued to files or tables, instance access information is not useful when determining energy savings. Therefore, we need to examine a certain number of I/O access sequences in order to determine whether energy saving efforts are worthwhile. An I/O sequence of data item $A$ is a sequence of I/Os issued from applications to data item $A$ during a sufficiently long monitoring period. We currently classify I/O sequences into four I/O patterns, called logical I/O patterns.

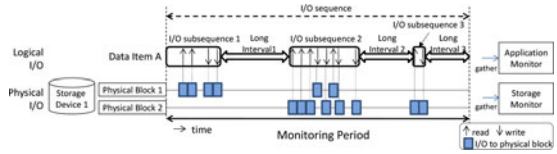



Fig. 2. Long interval and I/O sequence.

Let us introduce the concepts of a long interval and an I/O subsequence that are used to identify logical I/O patterns. A long interval is an I/O interval that is longer than the *break-even time* [21] of a storage device. Here, the break-even time is an indicator of storage power management. Once a storage device is turned off, it requires some energy to be turned on again. However, it consumes some energy to maintain the idle mode when waiting for an I/O request. An I/O subsequence consists of a sequence of read/write I/Os to a data item and I/O interval(s) that are shorter than the break-even time. Fig. 2 shows an example of long intervals and I/O subsequences for data item $A$. As can be seen from the figure, data item $A$ has three long intervals and three I/O subsequences. Long interval #3 ends at the end of the monitoring period, while I/O subsequence #1 starts at the beginning of the monitoring period.

Now let us describe the four logical I/O patterns. The main storage-energy management functions of our framework are: i) data placement, ii) buffer-based energy saving, and iii) turning off storage devices, as shown in Fig. 1. That means that considering how these three storage energy management functions are applied to each data item should provide sufficient information to achieve energy savings, although detailed fine tuning will be necessary later.

The logical I/O pattern definitions are as follows:

- *I/O Pattern P-I.* A P-I I/O pattern is one in which no I/O is issued to a data item during the monitoring period. This pattern has only one long interval and no I/O subsequence. That means there is no I/O issued to a data item during a monitoring period. P-I type data items are candidates for placement in storage devices that are often turned off.
- *I/O Pattern P-R.* A P-R I/O behavior has at least one long interval and at least one I/O subsequence, and the total number of reads in this pattern is larger than 50 percent of the total number of I/Os. P-R data items are read frequently; thus, they are potential candidates for receiving a pre-load function of the buffer-based energy saving functions.
- *I/O Pattern P-W.* A P-W I/O behavior has at least one long interval and at least one I/O subsequence, and the total number of reads in its I/O subsequences is less than 50 percent of the total number of I/Os in

the I/O subsequences. P-W data items are candidates for increasing write I/O intervals by delaying write I/Os to the storage devices.
- *I/O Pattern P-A*. A P-A I/O behavior has only one I/O subsequence. P-A data items are candidates for relocation to storage devices that will not receive a turn-off function because these items have no long intervals.

The first three patterns identify the energy saving potential, while the last pattern identifies those data items that are unsuitable for energy saving efforts.

## 4 Storage Device Model and Its Power Status

Large data centers typically have a variety of storage devices. For example, large data analytic systems running Hadoop or MapReduce are constructed from multiple servers with multiple HDDs, while large OLTP systems often use enterprise storage devices.

We introduced a *storage device model* to treat both HDDs and other enterprise storages devices as energy saving targets within a single framework. The universal storage device model enables our framework to uniformly issue commands, such as power on or off, to its target storage devices. By using this storage device model, our framework can target almost all storage devices, from HDDs to enterprise storage, as potential candidates for energy saving efforts. All storage devices have three power states: active, idle, and power off. Storage devices are powered-on in the active state and I/Os are executed. This mode consumes the most power of the three modes. Storage devices are powered on in the idle state, but no I/Os are executed. The energy consumed in this mode is lower than that in the active mode. Storage devices are turned off in the power-off state. If an I/O is issued to a powered off storage device, there is significant delay because it takes a long time to turn it on, and this process also requires additional energy.

## 5 Monitoring Functions

Monitoring functions are core functions for combining application-level I/O behaviors with storage-device-level I/O behaviors.

### 5.1 Application Level I/O Monitoring Function

In order to leverage the application-level I/O behaviors for energy saving purposes, the application monitor collects two types of information: logical address information and logical I/O statistics.

- *Logical address information*. Logical address information contains information on the relationship between a data item and the volume of the data item on a storage device. Here, a volume is a partition on an HDD or a logically separated area of a RAID group on a disk enclosure in an enterprise storage system.
- *Logical I/O statistics*. The logical I/O statistics contain the I/O read and write numbers of each data item to each volume during each monitoring interval.

Middleware such as DBMSs and web servers manage this logical address information while monitoring the logical I/O statistics. Since this information can easily be obtained from middleware logs and/or profile information, the collection overhead of the information is negligible. Our framework then calculates logical I/O patterns from this information.

### 5.2 Storage Device Level I/O Monitoring Function

Storage device level I/O monitors collect physical address information, physical I/O statistics, and energy consumption information regarding the storage devices.

- *Physical address information*. This information contains data on the relationship between a volume and a storage device.
- *Physical I/O statistics*. The physical I/O statistics contain the time-stamps issued when the I/O statistics are collected, the storage device name, and the number of read and write I/Os handled by the storage device.
- *Energy consumption of the storage device*. This information contains the name of the storage device, the time-stamp issued when energy-consumption information of the storage device was collected, and the energy consumption value of the storage device.

## 6 Storage Energy Management Flow

### 6.1 Overview

Algorithm 1 shows an outline of our universal storage energy management framework.

First, our framework monitors the application- and storage-device-level I/O behaviors over a fixed period called the monitoring period. During the monitoring period, it attempts to reduce the energy consumption of the storage devices by utilizing appropriate runtime energy management functions based on the strategies determined during a previous monitoring period. After the monitoring period is finished, it decides a logical I/O pattern for each data item by referring to a map showing the logical data and storage devices. The map is generated from the obtained application- and storage-device-level I/O behaviors.

Next, our framework determines an energy saving strategy that includes the placement of each data item. Our framework then attempts to extend the storage devices' I/O intervals by applying *pre-load* and *write delay* buffer-based energy saving functions. After determining an energy saving strategy, our framework determines a method of power control for each storage device based on the data placement. Storage devices that only contain P-I, P-R, and P-W data items are configured with a power-off function. In contrast, P-A data item storage will not receive this function because they have no potential for energy savings.

Finally, our framework determines the next monitoring period on the basis of the I/O behaviors of the latest monitoring period. Selected runtime energy management functions are executed during the following monitoring period. Note, however, the application behavior or energy consumption may deviate drastically from the estimates produced during the previous monitoring period. If this occurs, our framework terminates monitoring and re-calculates the energy saving strategy on the basis of the I/O behaviors obtained during the latest monitoring period.

**Algorithm 1.** Storage Energy Management Flow

```
Start application monitor and storage monitor;
while applications are running do
  Start selected runtime energy management functions;
  while monitoring period is not finished do
    if I/O interval of hot storage device > break-even time or
     number of powered on cold devices > threshold then
      break;
    end if
  end while
  Extract logical I/O pattern;
  Determine data placement strategy;
  Determine buffer-based energy saving strategy;
  Determine the power control strategy of the storage devices;
  Determine the length of the next monitoring period;
end while
```

### 6.2 Extract Logical I/O Pattern

The steps to extract logical I/O patterns are as follows:

- *Step 1. Find long intervals and I/O subsequences*. For each data item, find long intervals and I/O subsequences in the I/O sequence obtained during the monitoring interval.
- *Step 2. Determine a logical I/O pattern of a data item*. If a data item has no I/O subsequence, the logical I/O pattern of that data item is set as P-I. If a data item has no long intervals, the logical I/O pattern is set as P-A. For the remaining data items, the logical I/O pattern determination step is based on the number counts of the read and write I/Os for each data item. If more than half of the I/Os are read I/Os, then the logical I/O pattern of that data item is set as P-R; otherwise it is set as P-W.
- *Step 3. Extract access order of data items among middleware products*. Some applications, such as web commerce, use multiple middleware products like DBMS or HTTP server. Data items managed by these middleware might need to be accessed in a particular order. For example, when shopping online, users often access image files after accessing a product database in order to obtain more detailed information on a particular product. In this case, there is an access order between two data items. If we utilize this access order when deciding buffer-based energy saving functions, we may be able to reduce the energy consumed by the storage device more efficiently.

  For example, if we learn that the image files in a HTTP server are accessed after a DBMS table is accessed, our framework extracts an access order of DBMS and HTTP server. If a logical I/O pattern of the DBMS data item is P-R, our framework preloads the DBMS table *and* the related image files into a buffer at the same time. As a result, it may conserve more energy than would energy saving measures that only use data items separately.

  The data item extraction steps are as follows:

  **Step A.** Keep track of a pair of a P-R type data item and the data item accessed just before it (these data items are managed by different middleware).

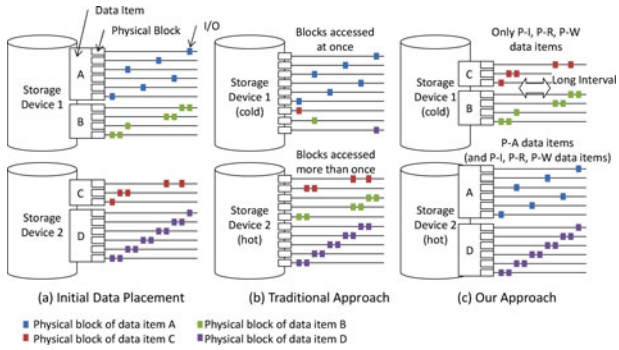



Fig. 3. Data placement comparison.

  **Step B.** Count the frequency of pairs listed up in Step A at the end of a monitoring period.

  **Step C.** Select the top $N$ pairs from the pairs counted in Step B.

  **Step D.** Use the access order of the pairs as the access order of the data items.

  Our framework preloads the data items extracted in these steps into a buffer.

### 6.3 Determine Data Placement Strategy

Effective data placement is one of most important features that enable effective energy conservation. Traditional approaches migrate storage blocks or files on the basis of their physical I/O frequency. But as an energy saving metric, the I/O intervals of logical data are more important than the I/O frequency of the physical data. Our framework determines data item placements on the basis of their I/O intervals.

When attempting to reduce energy consumption by turning off storage devices, at least one long interval is required for a data item. However, we cannot determine whether a data item has at least one long interval from the number of I/Os. Fig. 3 compares a conventional method of data placement and the approach we propose. Fig. 3a outlines the initial data placements of four data items (A, B, C, and D) in two storage devices, along with the access timing of the physical blocks. Here, we assume that data items have been placed on storage devices 1 and 2 to equalize their access frequency and size. Fig. 3b shows a conventional data placement based on the block I/O frequency. Here, frequently accessed blocks, which are those accessed more than once, are placed on storage device 2, while all the other blocks are placed on storage device 1. Data item $B$ blocks are migrated in this case to storage device 2, and data item $C$ blocks are migrated to storage device 1. The figure also shows a case where no long intervals were generated by the migrating blocks on the basis of their I/O frequency. Fig. 3c shows the data placement performed by our method. Here, data items that have at least one long interval are placed on storage device 1, while all the other data items are placed on storage device 2. Data item $A$ is migrated to storage device 2 in this case because there are no long intervals related to data item $A$. Data item $C$ is migrated to storage device 1 because it has one long interval. Our approach increases the probability of long intervals being generated in storage device 1 by removing P-A data items that have no long intervals. Thus, it is clear that proper use of application I/O behaviors

can potentially save a lot of energy. We can further classify storage devices into two groups: *hot* and *cold* storage devices. Hot storage devices are those that handle P-A data items and are never turned off. Cold storage devices handle P-I, P-R, and P-W data items. Cold devices are kept turned off in situations where no I/Os are pending or underway.

Our framework decides the placements of P-A data items on the basis of their current placement. Our framework sorts storage devices by the sizes of the P-A data items they store and designates them with numbers from $one$ to $n$. Here, $n$ is defined as the total number of storage devices in a target system. Our framework then selects the $i$th cold storage device with the smallest sum of P-A data items and checks to see if those P-A data items could be relocated to one of the storage devices numbered from 1st to $i-1$th. Here, it selects a storage device that satisfies the following conditions: (i) the free space in the storage device is larger than that of $t$, and (ii) the input/output per second (IOPS) sum of $t$ and the current IOPS of the storage device does not exceed the maximum IOPS that a single storage device can serve. If two or more storage devices satisfy these conditions, our framework selects the storage device with the smallest IOPS of $t$ and whose current IOPS is minimum. If such a storage device is found, it generates a plan to migrate $t$ to that storage device. If no storage device is found, our framework stops the data placement calculation of the P-A data items.

### 6.4 Determine Buffer-Based Energy Saving Strategy

Data-intensive application systems in todays data centers have large buffers both on their application servers and storage devices. Our framework determines a logical I/O pattern for each data item. Since data items contain application and storage device level access histories, our framework can utilize both sorts of buffers. As a result, it enables the energy saving strategy to be decided in a uniform way when both sorts of buffers are present.

Our framework utilizes its pre-load and write-delay functions when dealing with buffer-based energy saving functions. It gives higher priority to a write-delay function than a pre-load function because we can control the timing used to write updated blocks to storage devices by changing a buffer parameter, such as the dirty page rate. However, pre-load data targets depend on an application's read I/O behavior, and are outside of our control.

#### 6.4.1 Determine Write Delay Strategy

We consider P-W data items as potential write-delay targets because the number of write I/Os of a P-W data item is higher than that of its read I/Os.

Traditional buffers also write updated blocks into storage devices asynchronously when performing application block updates. The traditional methodology calls for writing updated blocks into storage devices at short intervals in order to reduce potential data loss if the system shuts down unexpectedly. For example, a UNIX system writes updated blocks to storage devices every 30 seconds [22], which is why it is difficult to extract a write I/O interval longer than a long interval using traditional methods. In contrast, our write delay function writes updated blocks with write I/O intervals longer than a long interval, which makes it possible to turn off cold storage devices.

#### 6.4.2 Determine Pre-Load Strategy

Since P-R data items have long intervals, and the number of read I/Os is larger than the number of write I/Os, we have restricted the pre-load function targets to P-R data items. We may also extend a read I/O interval by reducing the number of read I/Os.

A common purpose of conventional pre-load functions is the sequential acceleration of read performance. These functions read the next blocks into a buffer when they determine that blocks are being read sequentially. We did not attempt to determine if blocks are being read sequentially based on information from P-R data items. Furthermore, additional blocks are read into a buffer after a read request has been issued. Thus, it is difficult to reduce energy consumption of storage devices by using conventional pre-load functions.

In contrast, our pre-load function reads P-R data items into a buffer at the end of the monitoring period, before the P-R data items are actually read. Furthermore, it also reads any data items that are related with the read P-R data items. As a result, it increases the chances that a read I/O interval will be a long interval, and it can thus be expected to have a higher energy saving potential than the conventional approaches.

### 6.5 Determine Storage Device Power Control Strategy

After determining the applicable pre-load data items, our framework determines the power control of the storage devices with the power-off function only applicable to cold storage devices.

### 6.6 Determine Next Monitoring Period Length

Finally, our framework determines the length of the next monitoring period on the basis of the average length of all long intervals for each data item.

The calculation is $I_{new} = average(I_{cur}) \times \alpha$, where $I_{cur}$ is all the measured long intervals in the current monitoring period, and $I_{new}$ is the next monitoring period. The parameter $\alpha$ (greater than one) is introduced in order to increase the monitoring period when the average length of the I/O intervals approaches the length of the monitoring period. The reason that $alpha$ is slightly greater than one is to extract long intervals that are longer than the current monitoring period. Thus, our framework extends the monitoring period in order to reduce the CPU cycles required to calculate the energy management strategy.

Energy savings become large if we can spin up devices at the end of actual long intervals instead of spinning them up at the end of a monitoring interval when the read I/O interval is longer than the monitoring period. This is why we set the parameter $\alpha$ larger than 1. Our simulation results show that $\alpha$ values between 1 and 2 are good for saving storage energy.

## 7 Runtime Energy Management Functions

### 7.1 Data Placement Function

Our data placement function migrates data items among storage devices on the basis of the data item placements

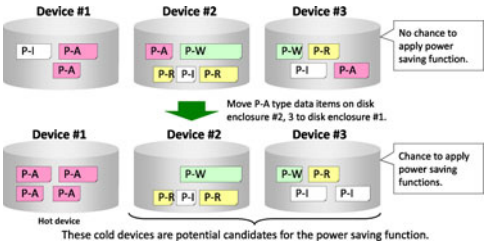



Fig. 4. Effect of data placement.

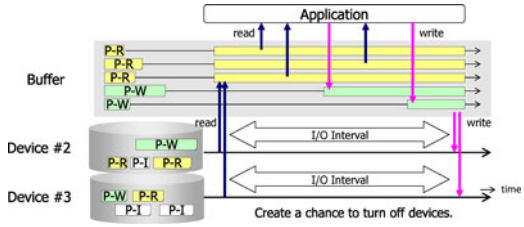



Fig. 5. Effects of pre-loading and write delaying data items.

described in Section 6.3. Data items are moved one at a time in the order determined via data placement strategy described at Section 6.3. The data placement function controls the data transfer rate to ensure they do not influence application performance. Fig. 4 shows the effects of our data placement. It indicates that we can create long intervals at storage devices #2 and #3 by removing the P-A data items from them.

### 7.2 Pre-Load Function

Our pre-load function loads the data items described in Section 6.4.2 into the buffer of the target system. Our pre-load function inspects the buffers and removes data items that are not designated pre-loaded data items if more space for pre-loaded data items is needed, and then it loads newly selected pre-loaded data items into the buffer. The function keeps data items that are already pre-loaded available in the buffer.

If the total size of the pre-loaded target data items is larger than the size of the buffer, our framework first sorts the pre-loaded target data items in descending order of read I/Os per unit size. Then, it pre-loads the data items from the top of the sorted data items. If the free space of the buffer becomes smaller than that of the next pre-loaded target data item, it skips it and tries to pre-load the next data item. If there are no pre-loaded target data items smaller than the free space of the buffer, our framework finishes the pre-load operations. It removes P-R data items from the buffer when the logical I/O patterns of the data items become P-I, P-W, or P-A. It refines the logical I/O pattern of data items at the end of a monitoring period. If a logical I/O pattern of a data item on the pre-loaded buffer is no longer P-R, it removes the data item from the buffer.

The application-level I/O monitoring function described in Section 5.1 to determine whether targets should be removed or not gathers the number of I/Os of data items and keeps this number in the server's memory. This operation is carried out at the end of every monitoring period by the pre-load function of our framework. Our current implementation uses 50 percent of the buffer for the pre-load function. We intend to explore techniques to determine this value in the future.

Fig. 5 shows the effects of pre-loading and write-delaying data items. As the figure indicates, we can extend the read I/O intervals of storage devices #2 and #3 by pre-loading P-R data items (and their related data items) into buffers.

### 7.3 Write Delay Function

Our write delay function first separates the target system buffer into two areas: one for *write-delayed* data items (a write delay buffer), the other for other data items. This function next assigns the write-delayed data items to the write delay area and the remaining data items to the other area. It then assigns a *dirty block rate* to the write delay area in order to extend the write I/O intervals. Here, the dirty block rate is the maximum rate for updating blocks in the buffer. If the number of updated blocks in the write delay buffer area reaches the number of blocks calculated from the dirty block rate, the target system flushes all these updated blocks into storage devices simultaneously. As shown in Fig. 5, we can extend the write I/O intervals of storage devices #2 and #3 by delaying P-W data item writes into the storage devices.

The current implementation of our framework uses 50 percent of the buffer for a write delay function. The value of the dirty block rate is 50 percent, which is the maximum for our enterprise storage's dirty block rate. The dirty block rate for the write delay buffer is a fixed value because the maximum dirty block rate maximizes the write I/O interval. As a result, we can keep the storage in energy saving mode for a long time.

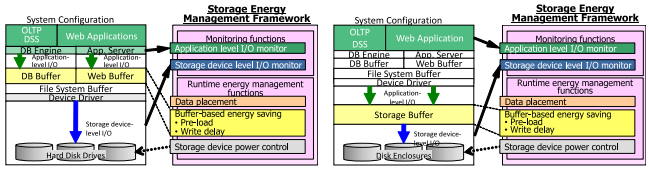
### 7.4 Reassignment of Storage Energy Saving Functions

Our storage energy management flow immediately interrupts the monitoring phase and reselects the energy saving strategy if one of the following conditions is satisfied: i) the service level agreement (SLA) is broken, e.g., the application performance deteriorates by more than 10 percent; ii) the number of power ons for cold devices exceeds $m$ from the end of the previous monitoring period $t_e$ and the current time $t_c$. Here, $m = (t_c - t_e)/l_b$, where $l_b$ is the length of the break-even time. By immediately reselecting the energy saving strategy when such issues arise, our framework maximizes the potential energy conservation even if the I/O behaviors suddenly change during the monitoring period.

## 8 Applying the Universal Energy Management Framework to Actual Systems

Our framework reduces the energy consumption of storage devices by utilizing the target system buffers. However, buffer configurations vary depending on the target application or system configuration. Accordingly, it is necessary to carefully choose the DBMS, file system, or storage buffers that are to be used for energy conservation and ensure they are in accordance with the configuration of the application or target system.

Small systems often use multiple HDDs as storage devices. However, while these HDDs also have buffers, they are unsuitable for energy conservation strategies because it is




(a) Multiple HDDs (b) Enterprise storage systems

Fig. 6. Universal storage energy management framework applied to actual system.

difficult to control the pre-load or write delay of their buffers. For such small systems, file system or DBMS application buffers are potentially more suitable energy saving methods. In our framework, the DBMS cache is used for pre-loading and delaying writing of data items in the DB, while the file system cache is used for pre-loading data items managed by web servers. Fig. 6a shows an implementation of our framework for a system using multiple HDDs.

Large-scale enterprise storage systems use devices that can contain from tens to thousands of HDDs and hundreds of gigabytes of non-volatile buffer area. These storage systems have several functions that can be used for energy conservation efforts, such as pre-load or write delay functions. (For example, the cache residency manager [23] allows administrators to dynamically lock and unlock cache data in real time.) The storage buffer is suitable for energy conservation purposes, and we can implement our pre-loading and write delay functions via the storage systems functions. Fig. 6b shows the implementation of our framework for storage systems.

Memory placements for buffers differ according to the scale of the system. In small systems, our framework uses a main memory in the server as a buffer. In a large enterprise storage system, it uses a cache memory as a buffer. We evaluated both implementations to demonstrate that our framework can reduce the energy consumption of these storage devices. As the placements of buffers differ according to the scale of the system, we think it is difficult to apply small configurations to large ones, and vice versa.

## 9 I/O Characteristics of Data Intensive Applications

Since logical I/O patterns are introduced to treat different applications in a uniform way, we investigated the logical I/O patterns of typical data intensive applications including file servers, OLTP, DSS, and various web applications.

### 9.1 Experimental System

Fig. 7 shows an overview of the system configuration used in the logical I/O behavior measurements.

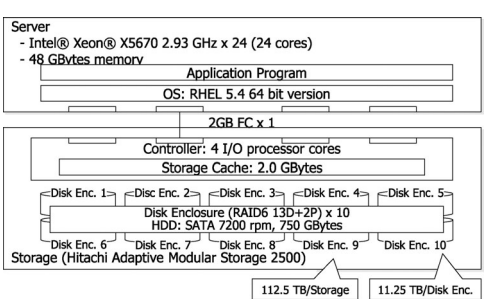



Fig. 7. Experimental system.

TABLE 1
Configuration of Applications of the Enterprise Storage Systems' Environment

| Application | Cache Size | Workload | Data Size |
|---|---|---|---|
| File Server (MSR Trace) (6 hr) | 19,800,000 records | Replay I/O trace using trace reply tool [27] Duration: 6 hr | 2 GB (Storage) |
| OLTP (TPC-C) | 500 GB | # of warehouse: 5,000 # of threads: 1,000 Think time: 0 Duration: 2.0 hr | 25 GB (DBMS) 2 GB (Storage) |
| DSS (TPC-H) | 100 GB | SF = 100 Run Q1 to 22 sequentially Duration: 6 hr | 5 GB (DBMS) 2 GB |
| Web commerce (TPC-W) | 500 GB | # of warehouse: 5,000 # of threads: 1,000 Think time: 0 Duration: 2.0 hr | 25 GB (DBMS) 2 GB (Storage) |

### 9.2 Application Configuration and Load Generation

Table 1 shows the configuration of the applications that were investigated for their logical I/O patterns. We replayed a Microsoft Research I/O trace and executed TPC-C[24], TPC-H[25], and TPC-W[26] on our experimental system (Fig. 7) as the respective loads for file servers, OLTP, DSS, and web commerce applications. TPC-W was created 10 years ago and, as such, contains parameters that are unsuitable for current web services such as online gaming or E-commerce. Therefore, we modified the item record access pattern so that it would follow a Pareto distribution. The distribution function is as follows: $F(x) = 1 - (x_m/x)^a$; $x_m = 1, a = 1$. We also removed the administration transactions from TPC-W because such transactions are rarely issued on actual systems.

For file servers, we created 36 volumes on 12 disk enclosures and assigned the volume MSR traces names in alphabetical order. For TPC-C, we created ten volumes on ten disk enclosures (i.e., each disk enclosure contains one volume) with log data placed on one volume. We then divided the tables and indexes into nine partitions using a DBMS hash function, and placed each table or index partition into one of the nine remaining volumes. For TPC-H, we created nine volumes on nine disk enclosures, with log data placed on one volume. We then divided the tables and indexes into eight partitions using a DBMS hash function, and put each of table or index partition into one of the eight remaining volumes. For TPC-W, we created nine volumes on nine disk enclosures with log data placed on one volume. We then divided the tables and indexes into four partitions using a DBMS hash function, and put each table or index partition into one of the four volumes. It was possible to load the TPC-W image files into the remaining four volumes because the data sizes of each of the volumes are equal.

### 9.3 I/O Pattern of Data Items

Fig. 8 shows the measured results of the logical I/O patterns for applications. The monitoring period is ten times longer than the break-even time. For file servers, 81.2 percent of the data items are P-I, 9.2 percent of the data items are P-R, 6.3 percent of the data items are P-W, and 3.3 percent of the

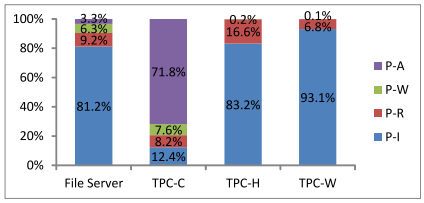



Fig. 8. Logical I/O patterns for data intensive applications.

data items are P-A. For TPC-C, 12.4 percent of the data items are P-I, 8.2 percent of the data items are P-R, 7.6 percent of the data items are P-W, and 71.8 percent of the data items are P-A. Unlike TPC-C, 83.2 percent of the TPC-H data items are P-I, 16.6 percent of the data items are P-R, and 0.2 percent of the data items are P-W. There are no P-A data items. For TPC-W, 93.1 percent of the data items are P-I, 6.8 percent of the data items are P-R, and 0.1 percent of the data items are P-W and P-A. This indicates that there is a need to modify the power saving strategy for each application.

The stability of the I/O patterns was also considered. It was found that the I/O patterns of all applications were stable while the application was running.

## 10 Evaluation

This section describes the evaluation of the energy-saving effects of our framework. In this evaluation, we used one small and one large storage device environment. Three applications (OLTP, DSS, and a web application) were run in the small storage device environment, while four data intensive applications, a file server, OLTP, DSS, and a web application, were chosen for the large storage device environment. For the small storage device, we used an array of five HDDs. For the large storage device, we used a commercial enterprise storage system that contains 150 HDDs.

For the file server, we replayed a Microsoft Research I/O trace [18]. Moreover, we used major benchmark programs for the application type, i.e., TPC-C [24] for OLTP,TPC-H [25] for DSS, and TPC-W [26] for the web application.

### 10.1 Evaluation Using Multiple HDDs

#### 10.1.1 Evaluation Methodology

1) *Comparison targets*. Our framework was compared with two traditional I/O behavior-based energy saving methods.
   - *Popular data concentration*. PDC [10] is a file system level I/O behavior-based data migration used for reducing power consumption of disk enclosures. The data unit is a file, not a data item.
   - *Dynamic data reorganization*. DDR [11] is a storage-device-level I/O behavior-based data migration used for reducing the power consumption of disk enclosures.

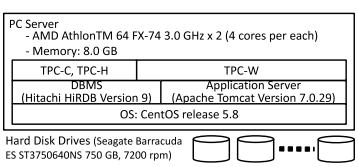



Fig. 9. Multiple HDD system testbed.

TABLE 2
Parameter Values for Evaluation (Multiple HDD Environment)

| Parameter | Value |
|---|---|
| Break-even Time | 24 sec |
| Spin down Time-out | 24 sec (Equal to Break-even Time) |
| Size of Hard Disk Drive | 750 GB |
| Coefficient of Monitoring Period ($\alpha$) | 1.2 |
| Initial monitoring period for our method | 240 sec |
| Monitoring period for PDC | 30 min |
| TargetTH of DDR | 60 IOPS |

2) *Testbed*. Fig. 9 outlines the experimental setup for evaluating our framework on a multiple HDD system. We used a Yokokawa digital power meter WT1600 (error rate was 0.1 percent) to measure the HDDs' energy consumption.
3) *Measurement items*. The energy consumption of the HDDs was measured. The energy consumption figures included expenditures for data item migration, pre-loading data items, delaying the writes of data items, and powering the HDDs on and off, when required. We also measured performance metrics including TPC-C transaction throughput, TPC-H query response time, and the number of web interactions per second (WIPS) for TPC-W. These performance metrics included delays caused by migrating data items, pre-loading data items, delaying data item writes, and powering on HDDs. We ran TPC-C, TPC-H, and TPC-W three times in this evaluation, and assessed the average energy consumption of multiple HDDs and their performance.
4) *Parameter setting*. Table 2 summarizes the parameter values for the proposed method. The break-even time is an actual HDD testbed value. The wait time for powering off an HDD (spin-down time-out) is equal to the break-even time. We used the parameter values from the original articles [10], [11] in order to compare PDC and DDR. We used parameter values for our framework based on the length of the break-even time of the storage devices.

We used a 50 percent as a dirty block rate in this experiment. Fifty percent is the maximum value for the enterprise storage we used. We believe that it is possible to maximize the energy saving effect by using this value. We selected the value of the monitoring period coefficient, alpha, to be slightly larger than the one to extend a long interval if its length was longer than the monitoring period. The length of the initial monitoring period was fixed at 10 times that of the device's break-even time. As described in Section 6.6, however, as the monitoring interval was changed by the logical I/O patterns of data items, an

TABLE 3
Data Item Definitions of Multiple HDD Environment

| Application | Definition of Data Item |
|---|---|
| OLTP (TPC-C) | Table and index |
| DSS (TPC-H) | Table and index |
| Web Application | Table, index and image file |

appropriate length was automatically set for the monitoring intervals.

5) *Workload*. Table 4 shows the configuration of data-intensive applications that were investigated for their logical I/O patterns.
6) *Initial data placement*. For TPC-C and TPC-H, we loaded log data into one HDD, and spread the DB tables and indexes as equally as possible across the other four HDDs. For TPC-W, we loaded log and DB data on two HDDs and distributed the image files onto two other HDDs. Both the DB data and image files were distributed as equally as possible on their assigned HDDs.
7) *Data item definition*. The definitions of the data items are in Table 3.

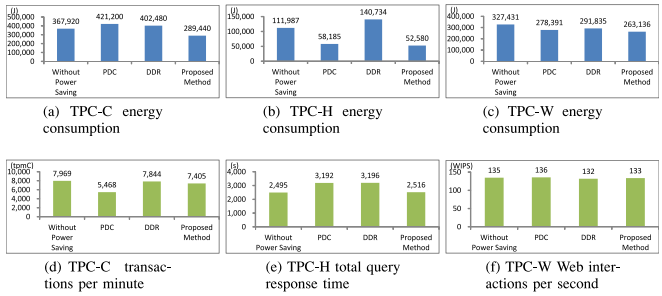
### 10.1.2 Evaluation Results

Figs. 10a, 10b, and 10c show the total energy consumption of five HDDs used by the TPC-C, TPC-H, and TPC-W applications, respectively. As shown in these figures, our framework was able to reduce HDD energy consumption more than could be accomplished with traditional approaches. At approximately 53 percent, the energy savings rate of TPC-H was especially high. Our framework decreased the HDD energy consumption for TPC-C and TPC-W by 21.3, and 19.6 percent, respectively.

Figs. 10d, 10e, and 10f show the transactions per minute of TPC-C, the total query response time of TPC-H, and web interactions per second of TPC-W, respectively. For TPC-C and TPC-W, our framework achieved almost the same application throughput as devices without our framework. On the other hand, a minor query response time degradation was recorded for TPC-H. Despite this, our framework resulted in less application performance degradation in comparison with PDC or DDR, while still providing significant energy savings.

TABLE 4
Configuration of Data Intensive Applications in Multiple HDD Environment

| Application | Data Size | Workload | Cache Size |
|---|---|---|---|
| OLTP (TPC-C) | 4 GB | # of warehouse: 10<br># of threads: 5<br>Think time: 0<br>Duration: 2 hr | 2 GB (DBMS) |
| DSS (TPC-H) | 5 GB | SF = 3<br>Run Q1 to 22 sequentially | 825 MB (DBMS) |
| Web Application (TPC-W) | 1.2 GB (DB)<br>2.7 GB | # of item: 10,000<br># of threads: 5<br>Think time: 0 | 600 MB (DBMS)<br>2 GB |




(a) TPC-C energy consumption (b) TPC-H energy consumption (c) TPC-W energy consumption

(d) TPC-C transactions per minute (e) TPC-H total query response time (f) TPC-W Web interactions per second

Fig. 10. Energy consumption and performance of multiple HDD environment.

## 10.2 Evaluation Using an Enterprise Storage System

### 10.2.1 Evaluation Methodology

1) *Comparison targets*. Our framework was compared with traditional I/O behavior-based energy saving methods as those in the storage system environment.
2) *Test bed*. The experimental system described in Fig. 7 was used as the testbed. We chose the number of HDDs to be 150 on the basis of our experience. A power meter was attached to the controller and disk enclosures to measure the actual power consumption by the storage unit. We used a Hioki 2300 series remote measurement and monitoring system (error rate was 0.3 percent) to measure the energy consumed by the enterprise storage system.
3) *Measurement items*. The total energy consumption and application performance levels were measured. The power consumption included expenditures for migrating data items, pre-loading data items, delaying the writes of data items, and powering the disk enclosures on and off. The application performance metrics were the same as those described in Section 10.1.1. We ran file server, TPC-C, TPC-H, and TPC-W three times in this evaluation, and assessed the average energy consumption of the storage system and its performance.
4) *Parameter setting*. Table 5 shows the parameter values of the proposed method. The break-even time, maximum IOPS, size of disk enclosures, and storage cache size were the actual values of the storage of the testbed. The wait time for powering off a disk enclosure (spin-down time-out) was equal to the break-even time.

   We selected the values of the dirty block rate, monitoring period coefficient alpha and initial monitoring period shown in Table 5 for the same reasons as described in Section 10.1.1 4).
5) *Workload*. During the evaluation process, we ran each application on our testbed. The application settings are described in Table 1.
6) *Initial data placement*. The initial data placement of each application was the same as described in Section 9.2.
7) *Data item definition*. The definitions of the data items are shown in Table 6.

TABLE 5
Parameter Values for Evaluation of Enterprise Storage Environment

| Parameter | Value |
|---|---|
| Break-even Time | 52 sec |
| Spin down Time-out | 52 sec (Equal to Break-even Time) |
| Maximum IOPS of Disk Enclosure | 900 (Random I/O) 2,800 (Sequential I/O) |
| Size of Volumes on Disk Enclosure | 1.7 TB |
| Storage Cache Size | 2 GB |
| Cache Size for Write Delay | 500 MB |
| Cache Size for Preload | 500 MB |
| Dirty Block Rate for Write Delay Cache | 50% |
| Monitoring Period Coefficient ($\alpha$) | 1.2 |
| Initial monitoring period for our method | 520 sec |
| Monitoring period for PDC | 30 min [10] |
| TargetTH of DDR | 450 IOPS [11] |

TABLE 6
Data Item Definitions of Enterprise Storage Environment

| Application | Definition of Data Item |
|---|---|
| File Server | Continuous blocks broken by non-accessed blocks |
| OLTP (TPC-C) | Partition of table and index |
| DSS (TPC-H) | Partition of table and index |
| Web Application | Partition of table and index, image file |

#### 10.2.2 Evaluation Results

Figs. 11a, 11b, 11c, and 11d correspond to the total energy consumption of the storage system used by the file server, TPC-C, TPC-H, and TPC-W applications, wherein our framework reduced the energy consumed by a storage system more than the conventional approaches. The energy savings for TPC-W were especially high, at 66.5 percent. Our framework decreased the storage system's energy consumption by 25.8 percent for the file server application. It decreased the storage system's energy consumption by 14.1 percent for TPC-C, and 48.4 percent for TPC-H.

Figs. 11e, 11f, 11g and 11h show the I/O response time of the file server application, the transactions per minute for TPC-C, the total query response time for TPC-H, and the web interactions per second of TPC-W, respectively. As shown in these figures, for TPC-C and TPC-W, the proposed method imposed only a minor penalty in application throughput relative to that of the HDDs *without the energy saving functions*.

For the file server application, the I/O response time of our framework was the shortest because of the efficiency provided by our pre-load function. For TPC-H, we found that the query response times of all methods increased, but that the increase imposed by our framework was less than that imposed by PDC or DDR.

These two results show that our framework enables significant reductions in storage device energy consumption at the expense of only a minor application performance degradation.

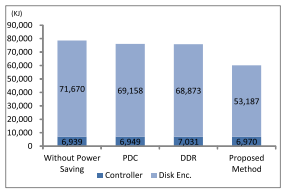
### 10.3 Analysis

As shown above, the our framework can achieve runtime power savings without sacrificing performance. However, the effects of the four logical I/O patterns require further clarification because the proposed method places data items into storage devices based on them. The key points are the storage device I/O intervals, total storage device spin up/down numbers, and data migration costs.

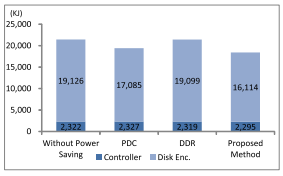
#### 10.3.1 Storage Device Level I/O Interval Lengths

First, we compared I/O interval trends. Fig. 12 shows the relationship between I/O intervals and the total lengths of I/O intervals longer than one second for the file server, TPC-C, TPC-H, and TPC-W applications. The $x$-axis shows the length of the I/O intervals, while the $y$-axis shows the cumulative length of the I/O intervals.

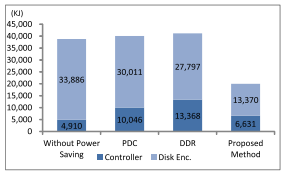
As shown in these figures, the I/O intervals of our framework (except for TPC-H) are longer than those of PDC or



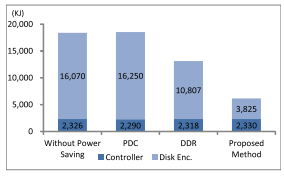

(a) File server energy consumption



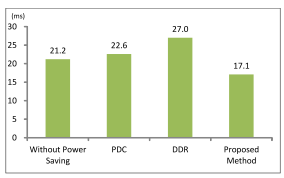

(b) TPC-C energy consumption



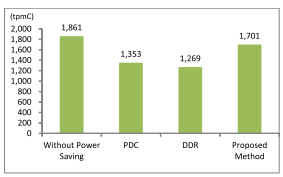

(c) TPC-H energy consumption



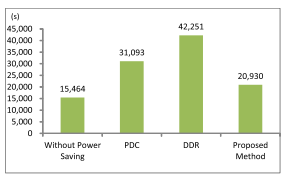

(d) TPC-W energy consumption



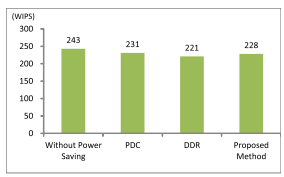

(e) File Server I/O response time




(f) TPC-C transactions per minute

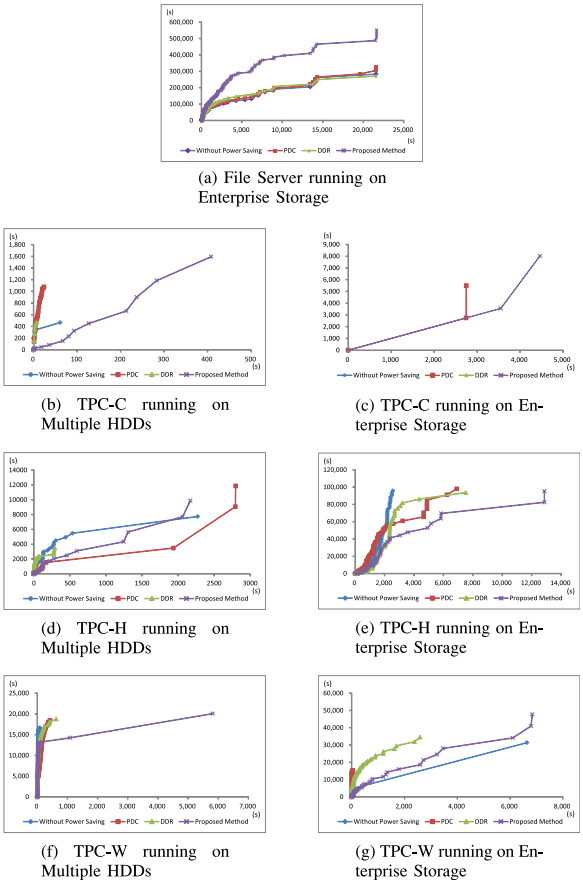



(g) TPC-H total query response time




(h) TPC-W Web interactions per second

Fig. 11. Energy consumption and performance of a storage system environment.




(a) File Server running on Enterprise Storage

(b) TPC-C running on Multiple HDDs

(c) TPC-C running on Enterprise Storage

(d) TPC-H running on Multiple HDDs

(e) TPC-H running on Enterprise Storage

(f) TPC-W running on Multiple HDDs

(g) TPC-W running on Enterprise Storage

Fig. 12. I/O Intervals of file server, TPC-C, TPC-H, and TPC-W.

DDR in multiple HDDs environments. It is evident that the PDC I/O intervals lengthened because the wait count for spinning up PDC HDDs is much higher than other methods, and it prevented PDC from reducing total HDD energy consumption despite the long I/O intervals.

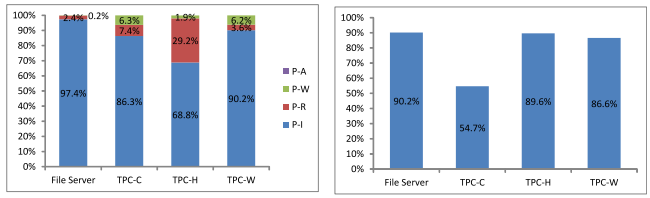
### 10.3.2 Data Migration Costs

We compared the total sizes of data migrated during the measurement period (Table 7).

As shown in this table, the migrated data size of our framework is the smallest for the multiple HDD environment. For the enterprise storage environment, the migrated data size of our framework was smaller than that of PDC. This is because our framework migrates only P-A data items. The migrated data size of DDR in the enterprise storage system environment was smaller

TABLE 7
Migrated Data Size (GB)

| Appl. | Env. | PDC | DDR | Proposed Method |
|---|---|---|---|---|
| File Server | Storage | 3,238 | 1 | 23 |
| TPC-C | HDD | 1.1 | 4.8 | 0.1 |
| | Storage | 1,082 | 1 | 30 |
| TPC-H | HDD | 7.6 | 36.8 | 4.8 |
| | Storage | 100 | 2 | 61 |
| TPC-W | HDD | 9.5 | 2.34 | 1.4 |
| | Storage | 97 | 2 | 12 |




(a) Block-based logical I/O patterns

(b) Storage energy consumption

Fig. 13. Effect of data items compared with physical block-based emthod.

than that of our method because the IOPS of the disk enclosures is higher than the LowTH.[1] of DDR (225, half of the TargetTH). Therefore, as is shown in Fig. 11d, DDR could not reduce the energy consumption of the enterprise storage.

## 10.4 Effect of Data Items

We calculated the logical I/O patterns of physical blocks to demonstrate the advantages of our approach. Fig. 13a presents the results.

The logical I/O patterns of blocks are similar for different applications, as can be seen in Fig. 13a. The block-based method could not find the P-A data items of logical I/O patterns that were are not suitable for energy savings, especially in the case of TPC-C. The block-based method selected blocks that had little effect on the pre-load function as P-R in TPC-H. This means that it is difficult for block-based monitoring methods to save energy by using the I/O behaviors of applications.

We also conducted simulations comparing the energy consumed by block-level buffer management with the energy consumed by our method. The results are presented in Fig. 13b. The parameters for the simulation are listed in Table 8.

Fig. 13b compares the energy consumed by our approach and by block-based buffer management. The energy consumed by block-based buffer management is 100 percent. As we can see from the graph, our method reduced energy consumption by 10 percent for the file server and TPC-H, 13 percent for TPC-W, and 45 percent for TPC-C. These results indicate that data items are a suitable way of establishing a universal energy saving framework for storage.

## 10.5 Effect of Each Energy Saving Strategy Used by Our Framework

We compared energy consumed by our approach and that of the buffer-only and migration-only energy saving strategies by using actual I/O traces captured during our experiments. Energy consumption of a buffer-only storage energy saving for File Server, TPC-C, TPC-H and TPC-W are 110.9, 182.9, 100.0 and 115.4 percent respectively compared with our framework. Energy consumption of a migration-only storage energy saving for File Server, TPC-C, TPC-H and TPC-W are 100.1, 110.3, 109.2 and 100,1 percent respectively compared with our framework. These results demonstrate

1. LowTH is an indicator of the disk enclosures in DDR. LowTH means the minimum IOPS of the disk enclosures that are considered to be cold in DDR.

TABLE 8
Simulation Parameters

| Item | Value |
|---|---|
| Monitoring Interval | 520 (s) |
| Break Even Time | 52 (s) |
| Power off time out | 104 (s) |
| Power consumption of active/idle state | 287.5 (W) |
| Power consumption of power off state | 0.0 (W) |
| Energy consumption of spin up | 1,4950.0 (J) |

that a combination of buffer-based and migration strategies is useful for saving energy.

## 11 Related Work

There are numerous energy saving methods for storage devices.

1) *I/O Interval Control using cache memory*. I/O Interval Control controls the I/O timing of an application by using a cache memory in order to increase the probability that storage devices are in an energy saving mode [12], [13], [14], [28], [29], [30]. Our method extracts application level logical I/O patterns that are deemed suitable for storage energy conservation and then selects appropriate energy conservation methods, such as data placement control or cache-based I/O interval extensions. Therefore, our method is more capable of reducing storage energy consumption than the traditional methods used only device level I/O patterns are.
2) *Data placement control*. Data placement control seeks to reduce HDD power consumption by controlling the data placement in these drives [10], [11], [16], [17], [18], [19], [20], [31], [32], [33]. These physical I/O behavior-based approaches are unable to find energy saving potentials obtained by combining physical and logical I/O behaviors. Furthermore, a logical I/O behavior-based approach cannot recognize disk enclosures (units of power on and off) in large storage environments because these storage units are highly virtualized. Therefore, logical I/O-based data placement may not work well.
3) *Application-level energy saving*. Another paper in the literature [34] describes traditional hardware-based energy saving methods as part of the solution and notes that data management software is important for power conservation for large data centers.This paper [34] did not discuss energy saving efficiency or the impact on DBMS performance. Some papers [35], [36], [37], [38] show the energy efficiency of data warehouse systems. These reports identify a number of points for energy proportion of data warehouse systems. However, they avoid discussion of energy saving after the data center is implemented, or in relation to other applications such as OLTP or web commerce.

## 12 Conclusion

In this paper, we proposed a universal framework for reducing storage device energy consumption while critical applications are running. We introduced a logical I/O pattern in order to easily handle a variety of application types, and analyzed the I/O behaviors of each application in a uniform way. Furthermore, we measured the I/O behaviors and energy consumption of multiple HDDs and an enterprise storage system with applications such as a file server, OLTP, DSS, and web commerce. We then compared the method based on our framework with those of conventional PDC and DDR storage energy saving methods. For all applications, we found that the energy savings of our method were equal to or greater than those of PDC and DDR. Additionally, application performance levels when using our method did not degrade significantly in comparison with devices without energy saving functions. The results of our evaluations show that our universal storage energy saving framework can make significant contributions to data center energy conservation efforts.

## Acknowledgments

Norifumi Nishikawa is the corresponding author.

## References

[1] J. Gantz and D. Reinsel. (2012). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east [Online]. Available: http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf
[2] J. Chang, R. W. Cox, S. Deshpande, A. Kros, M. Suzuki, S. Low, A. Munglani, and A. Kim, "Forecast: External controller-based disk storage, worldwide, all countries, 2011-1027, 1q13 update," 2013.
[3] A. G. Yoder, "Green storage technologies, CAPEX and OPEX," in *Proc. Conf. SNW Fall*, [Online]. Available: http://www.snia.org/sites/default/education/tutorials/2010/fall/green/AlanYoder_Green_Storage_Technologies-2.pdf, 2010.
[4] S. Worth. (2012). Green storage—the big picture [Online]. Available: http://www.snia.org/sites/default/education/tutorials/2012/fall/green/SW_ Green Storage_Big_Picture_9-12.pdf
[5] M. Poess and R. O. Nambiar, "Energy cost, the key challenge of today's data centers: A power consumption analysis of TPC-C results," *Proc. VLDB Endowment*, 2008, vol. 1, no. 2, pp. 1229–1240.
[6] M. Kitsuregawa, *Storage Networking*. Ohmsha, 2011.
[7] F. Moore and A. Guha, "Introducing COPAN systems' MAID architecture," 2004.
[8] "Hitachi adaptable modular storage 2500," *HITACHI, Ltd.*, 2011.
[9] D. Colarelli, D. Grunwald, and M. Neufeld, "The case for massive arrays of idle disks (MAID)," in *Proc. Conf. File Storage Technol.*, 2002, pp. 1–6.
[10] E. Pinheiro and R. Bianchini, "Energy conservation techniques for disk array-based servers," in *Proc. 18th Annu. Int. Conf. Supercomput.*, 2004, pp. 68–78.
[11] E. Otoo, D. Rotem, and S.-C. Tsao, "Dynamic data reorganization for energy savings," in *Proc. 22nd Intl. Conf. Sci. Statist. Database Manage.*, 2010, pp. 322–341.
[12] A. E. Papathanasiou and M. L. Scott, "Energy efficient prefetching and caching," in *Proc. USENIX Annu. Tech. Conf.*, 2004, p. 22.
[13] D. Li and J. Wang, "EERAID: Energy efficient redundant and inexpensive disk array," in *Proc. 11th ACM SIGOPS Eur. Workshop*, article 29, 2004.
[14] X. Yao and J. Wang, "RIMAC: A novel redundancy-based hierarchical cache architecture for energy efficient , high performance storage system," in *Proc. 1st ACM SIGOPS/EuroSys*, 2006, pp. 249–262.
[15] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini, "Application transformations for energy and performance-aware device management," in *Proc. Int. Conf. Parallel Archit. Compilation Tech.*, 2002, pp. 121–130.

[16] D. Colarelli and D. Grunwald, "Massive arrays of idle disks for storage archives," in *Proc. ACM/IEEE Conf. Supercomput.*, 2002, p. 47.
[17] C. Weddle, M. Oldham, J. Qian, A.-I. A. Wang, P. Reiher, and G. Kuenning, "PARAID: A gear-shifting power-aware raid," in *Proc. 5th USENIX Conf. FAST*, Oct. 2007, vol. 3, no. 3, pp. 245–260.
[18] A. Donnelly and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," in *Proc. 6th USENIX Conf. 6th USENIX Conf. File Storage Technol.*, 2008, pp. 253–267.
[19] A. Verma, R. Koller, L. Useche, and R. Rangaswami, "SRCMap: Energy proportional storage using dynamic consolidation," in *Proc. 8th USENIX Conf. File Storage Technol.*, 2010, p. 20.
[20] J. Guerra, H. Pucha, J. Glider, W. Belluomini, and R. Rangaswami, "Cost effective storage using extent based dynamic tiering multi-tiering: Design choices," in *Proc. 9th USENIX Conf. File Storage Technol.*, 2011, pp. 1–14.
[21] G. De Micheli, "Comparing system level power management policies," *IEEE Des. Test Comput.*, vol. 18, no. 2, pp. 10–19, Mar. 2001.
[22] S. Carson and S. Setia, "Analysis of the periodic update write policy for disk cache," *IEEE Trans. Softw. Eng.*, vol. 18, no. 1, pp. 44–54, Jan. 1992.
[23] Hitachi virtual storage platform architecture guide. (2011) [Online]. Available: http://www.hds.com/assets/pdf/hitachi-architecture-guide-virtual-storage-platform.pdf
[24] TPC benchmark C standard specification revision 5.11, *Trans. Process. Perform. Council*, 2010.
[25] TPC benchmark H (decision support) standard specification revision 2.14.2, *Trans. Process. Perform. Council*, 2011.
[26] TPC benchmark W (web commerce) specification version 1.8, *Trans. Process. Performance Council*, 2002.
[27] A. Brunelle, "Btrecord and btreplay user guide," 2010.
[28] D. Li, H. Cai, and X. Yao, "Exploiting redundancy to construct energy-efficient, high-performance RAIDs," Comput. Sci. Eng. Dept., Univ. Nebraska Lincoln, Lincoln, NE, USA, Tech. Rep. TR-05-07-04, pp. 1–20, 2005.
[29] Q. Zhu and Y. Zhou, "Power-aware storage cache management," *IEEE Trans. Comput.*, vol. 54, no. 5, pp. 587–602, May 2005.
[30] F. David and C. Devaraj, "Reducing energy consumption of disk storage using power-aware cache management," in *Proc. 10th Int. Symp. High Perform. Comput. Archit.*, 2008, pp. 118–118.
[31] E. Otoo, D. Rotem, and S.-C. Tsao, "Workload-adaptive management of energy-smart disk storage systems," in *Proc. IEEE Int. Conf. Cluster Comput. Workshops*, 2009, pp. 1–11.
[32] J. Guerra, W. Belluomini, J. Glider, K. Gupta, and H. Pucha, "Energy proportionality for storage: Impact and feasibility," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 35–39, 2010.
[33] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan, "Robust and flexible power-proportional storage," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, p. 217.
[34] M. A. Shah, S. Harizopoulos, and J. Meza, "Energy efficiency: The new holy grail of data management systems research," in *Proc. 4th Biennial Conf. Innovative Data Syst. Res.*, [Online]. Avairable: http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_112.pdf, 2009.
[35] J. Meza, M. A. Shah, P. Ranganathan, M. Fitzner, and J. Veazey, "Tracking the power in an enterprise decision support system," in *Proc. 14th ACM/IEEE Int. Symp. Low Power Electron. Des.*, 2009, p. 261.
[36] M. Poess and R. Nambiar, "Tuning servers, storage and database for power efficient data warehouse," in *Proc. 26th IEEE Int. Conf. Data Eng.*, 2010, pp. 1006–1017.
[37] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah, "Analyzing the energy efficiency of a database server," in *Proc. Int. Conf. Manage. Data*, 2010, pp. 231–242.
[38] W. Lang and S. Harizopoulos, "Towards energy-efficient database cluster design," *Proc. VLDB Endowment*, vol. 5, no. 11, pp. 1684–1695, Jul. 2012.

**Norifumi Nishikawa** received the BFA degree from the Department of Measurement Engineering, Kobe University, in 1989, and the MFA degree from the Department of Instrumental Engineering, Graduate School of Engineering, Kobe University, in 1991. He received the PhD degree from the Department of Electronics and Information, Graduate School of Information Science and Technology, University of Tokyo, in 2012. He joined Hitachi, Ltd., in 1991. He is a member of IPSJ and the Database Society of Japan.

**Miyuki Nakano** received the BS and PhD degrees from the University of Tokyo, respectively. In 1981, she was at Fujitsu Corporation and in 1985, she began working at the University of Tokyo. In 2004, she became an associate researcher at the Institute of Industrial Science, University of Tokyo. In 2009, she became an associate professor at the same institute. She is currently a professor at the Shibaura Institute of Technology. Her interests include the study of database systems and storage systems. She is currently engaged in engineering data research. She is a member of the IEEE, IEICE, IPSJ, ACM, and the Database Society of Japan.

**Masaru Kitsuregawa** received the BS, MS, and PhD degrees from the University of Tokyo, respectively. He is currently the director general at National Institute of Informatics (NII) in Japan and is also a professor and the director in the Center for Information Fusion, Institute of Industrial Science, and the executive director in the Earth Observation Data Integration and Fusion Research Initiative (EDI-TORIA), University of Tokyo. His research interests include high-performance database engineering, and big data system. He is running earth environmental digital earth of more than 10PB. He serves as a chair of the steering committee of International Conference on Data Engineering, and served as a trustee of the VLDB Endowment. He received the ACM SIGMOD E.F. Codd Innovation Award in 2009. He was serving as a science advisor to the Ministry of Education, Culture, Sports, Science and Technology in Japan. He is a fellow of the ACM and the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.