Power-Saving in Large-Scale Storage Systems with Data Migration

Koji Hasebe, Tatsuya Niwa, Akiyoshi Sugiki, and Kazuhiko Kato
Graduate School of Systems and Information Engineering
University of Tsukuba,
1-1-1 Tennodai, Tsukuba 305-8573, Japan
hasebe@iit.tsukuba.ac.jp, niwa@osss.cs.tsukuba.ac.jp, {sugiki,kato}@cs.tsukuba.ac.jp

***Abstract*—We present a power-saving method for large-scale distributed storage systems. The key idea is to use virtual nodes and migrate them dynamically so as to skew the workload towards a small number of disks while not overloading them. Our proposed method consists of two kinds of algorithms, one for gathering or spreading virtual nodes according to the daily variation of workloads so that the active disks are reduced to a minimum, the other for coping with the changes in the popularity of data over a longer period. For this dynamic migration, data stored in virtual nodes are managed by a distributed hash table. Furthermore, to improve the reliability as well as to reduce the migration cost, we also propose an extension of our method by introducing a replication mechanism. The performance of our method is measured both by simulation and a prototype implementation. From the experiments, we observed that our method skews the workload so that the average load for the active physical nodes as a function of the overall capacity is 67%. At the same time, we maintain a preferred response time by setting a suitable maximum workload for each physical node.**

I. Introduction

Power-saving has become a central issue in today's computing systems. In particular, as a high percentage of the total computing system's energy is used by the data storage systems, a number of suggestions for reducing power in storage systems have been proposed, e.g. [2], [6], [7]. Many of these studies have restricted their scope to storage systems with a specific kind of central controller to manage the data access or storage systems consisting of a relatively small number of disks (typically, up to several dozen). However, when considering increasingly large-scale computing systems as typified by cloud computing [1], scalability is of major importance.

In this paper, we propose a power-saving method for large-scale distributed storage systems. To reduce power consumption of storage systems, a commonly-observed technique in the literature is to skew the workload towards a small number of disks, thereby enabling the others to be in a low-power mode. To apply this technique to large-scale distributed storage systems, a possible approach is to use replication; a recent example of which is [4] where they investigated the efficient allocation of replicated data to minimize the number of active disks, while all the data are accessible. An alternative approach, not yet thoroughly studied, is to migrate the stored data dynamically in such a way that all data are gathered into as small a number of disks as possible while not overloading them. The main objective of our research is to explore the power-saving method using the latter approach.

For dynamic migration, in our method, data stored in virtual nodes are managed by a distributed hash table (DHT), which has been adopted in the context of load-balancing for DHTs (e.g., [3]). On the other hand, the migration is controlled by two types of algorithms. One, called the *short-term optimization algorithm*, is used for gathering or spreading virtual nodes according to the daily variation of the workload so that the number of active physical nodes is reduced to a minimum. The other algorithm, called the *long-term optimization algorithm*, is used for coping with changes in the popularity of data over a longer period (for example, a week). This leads to more effective short-term optimization. Furthermore, we propose an extension of our method by introducing a replication mechanism to reduce the migration cost and improve reliability (i.e., fast recovery from disk failure).

The performance of our method is measured both by simulation and with a prototype implementation in terms of the average load, the average response time, and the migration cost. From the experiments with our prototype implementation, we observed that our method skews the workload so that the average load of the active physical nodes as a function of the overall capacity is 67%. This result indicates that our method effectively skews the workload while keeping to a preferred response time (whose overall average is 80msec) by setting a suitable maximum workload (i.e., threshold of migration) for each physical node. Additionally, our simulation shows that the long-term optimization algorithm consistently and continually improves power consumption.

This paper is organized as follows. Section 2 presents related work. Section 3 describes the underlying storage system and the method of data migration. Section 4 introduces our power-saving algorithms. Section 5 proposes a replication mechanism in our method. Section 6 presents the simulation results. Section 7 presents the evaluation by means of a prototype implementation. Finally, Section 8

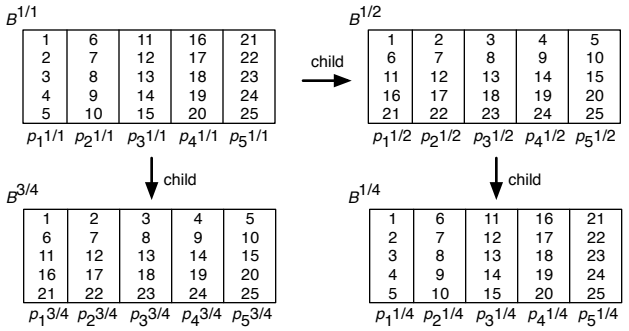

Figure 1. Example of configuration of virtual nodes (For $c = 5$, $d = 4$)

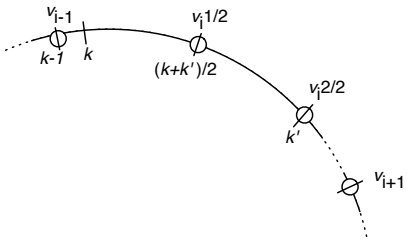

Figure 2. Split of virtual node $v_i^{1/1}$ into $v_i^{1/2}$ and $v_i^{2/2}$

concludes the paper and presents future work.

II. Related Work

There have been a number of studies to reduce storage power consumption. A commonly-observed feature in many of these techniques is that they skew the workload, and can be classified into the following categories according to the variations in this approach.

The first category, including MAID [2] and PDC [6], focuses on the popularity (i.e., access rate distribution) and concentrates popular data on specific disks. The second, as typified by Pergamum [9], uses NVRAM to extend the low-power mode period by caching data to a write store. The final category considers redundancy (i.e., data replication). In DIV [7], original and redundant data are separated into different disks thereby allowing read/write requests to be concentrated on the disks with the original data. In EERAID [5], RIMAC [12], and eRAID [10], a data access on a disk in low-power mode is transformed into accesses on active disks or caches, and the required data are reconstructed from the parities obtained during these accesses. In Hibernator [13] and PARAID [11], data are collected or spread to adapt to changes in operational loads. Our study mainly focuses on the dynamic migration of storing data.

Although these studies in the literature restrict their scope to storage with a specific kind of central controller to manage the data access, recent work such as Harnik et al., [4] addresses power-saving in large-scale distributed storage. They investigated the efficient allocation of replicated data so as to minimize the number of disks while all the data are accessible. Our main motivation is to explore an efficient technique to skew the workload by means of migration, instead of taking the replication approach.

III. Underlying System and Data Migration

The underlying storage system of our method consists of a set of physical nodes, each of which contains a set of virtual nodes. In our method, the number of virtual nodes in every physical node is fixed at some specific number (denoted by c). The physical nodes are segmented into blocks by c, thus each block contains c^2 virtual nodes. To simplify our discussion, let us consider a usage environment where the daily workload may vary, and five physical nodes are required at off-peak time to maintain a certain expected response time, but may increase up to four-fold. (Throughout this paper, the increasing rate of system workload is denoted by d. Thus, in this case $d = 4$.) In this case, the system configuration is as follows. (See also Fig 1 for a graphical presentation.) First, the required 20 ($= c \cdot d$ with $c = 5$, $d = 4$) physical nodes are segmented into four blocks named $B^{1/1}$, $B^{1/2}$, $B^{1/4}$, and $B^{3/4}$, where the physical nodes in $B^{i/j}$ are called $p_1^{i/j}, \ldots, p_5^{i/j}$ for each block. Next, 25 ($= c^2$) virtual nodes named $v_1^{1/1}, \ldots, v_{25}^{1/1}$ are mapped on $B^{1/1}$ as in Fig 1 where the numbers in block $B^{1/1}$ indicate the allocation of virtual nodes. For example, the second virtual node, $v_2^{1/1}$, is allocated on $p_1^{1/1}$, and $p_1^{1/1}$ contains $v_1^{1/1}, \ldots, v_5^{1/1}$. On the other hand, operations to store/retrieve data are realized using the lookup mechanism of a DHT. That is, 25 virtual nodes are allocated at equal distances in the key space of the DHT.

When the system workload increases, each physical node in $B^{1/1}$ independently checks its own workload, and if it exceeds its capacity, i.e. the maximum workload to maintain a certain expected response time, one of the virtual nodes is moved to another low-loaded active physical node in the next block, $B^{1/2}$. We call such a block a *child*. If there is no such physical node, the leftmost node in a low-power mode is activated. For finer-grained adjustment of the workload, when moving a virtual node it is split as follows. (See also Fig 2 for the graphical presentation. Here, we consider Chord [8] as the underlying DHT.) For example, for the case of $i = 2$ in Fig 2, when the workload of $p_1^{1/1}$ exceeds capacity and virtual node $v_2^{1/1}$ is moved, a new virtual node (called $v_2^{1/2}$) is created and put at the midpoint of the key space covered by $v_2^{1/1}$ (i.e., the ID of $v_2^{1/2}$ is set as $(k + k')/2$), then the data corresponding to the key space $[k, (k+k')/2]$ are copied to $v_2^{1/2}$. Thus, through the built-in mechanism for maintenance of routing tables, the key space $[k, k']$ is shared equally by these nodes. At this moment, the original virtual node is renamed as $v_2^{2/2}$. Then $v_2^{1/2}$ is moved to the point (which is indicated by the corresponding

number in the figure) in the next block (i.e., $B^{1/2}$), while $v_2^{2/2}$ stays at the same point as the original physical node.

In contrast, when the system workload is decreasing, split virtual nodes are gradually moved back to the original point. For example, when the workload of $p_1^{1/1}$ becomes low and able to absorb $v_2^{1/2}$, the data stored in this virtual node are merged into $v_2^{2/2}$ and then removed from the key space. At this moment, the merged node is renamed as $v_2^{1/1}$ and put into the same position as $v_2^{2/2}$. Finally, if a physical node has no active virtual node, it enters a low-power mode.

This migration process occurs for each block according to the daily variation of the system workload. For example, after all the virtual nodes in $B^{1/1}$ are migrated to $B^{1/2}$, again, virtual nodes $v_i^{2/2}$ and $v_i^{1/2}$ are split into the pairs $\langle v_i^{3/4}, v_i^{4/4} \rangle$ and $\langle v_i^{1/4}, v_i^{2/4} \rangle$, and then the first-half nodes are moved to $B^{3/4}$ and $B^{1/4}$ in a similar way. This process is realized by using the algorithms, called short-term optimization algorithms. In addition, we also provide an auxiliary algorithm, called the long-term optimization algorithm, to maintain the effectiveness in an environment where the popularity of data varies. The details of these algorithms are explained in the next section.

Here we would like to note that the mapping of the virtual nodes of any two blocks of a parent-child relationship is orthogonal. That is, as indicated in Fig 1, the virtual nodes in a block are allocated in vertical lines if and only if the virtual nodes in its parent as well as its child blocks are allocated in horizontal lines. Because any physical node in a parent block has all the physical nodes in its child block as destinations of migration, the physical nodes in the child block can be efficiently activated from the left. This fact is demonstrated in later sections.

Finally, we comment on the migration cost. To reduce the migration cost, instead of moving the whole data stored in a virtual node, there is an alternative way. That is, when $v_2^{1/2}$ is removed, the stored data remain in $p_2^{1/2}$ and are reused at the time of the next splitting operation. This enables the migration by copying the difference from the previous day. Indeed, the advantage of this technique is the trade-off with the disk space, however if the system has enough space on the disks and can afford to create some redundancy, this technique can be useful for effective migration. In Section 6, we present the introduction of a replication mechanism using this technique.

IV. Power-Saving Algorithms

Our power-saving method relies primarily on two types of algorithm. In this section, we introduce the type named *short-term optimization algorithm*, which copes with the daily variation of the system load. This algorithm consists of two parts; one is used when the workload is increasing and the other when the workload is decreasing, and they are run at regular intervals. We next explain why we require the auxiliary algorithm, named *long-term optimization algorithm*, which is used for load-balancing in each block, and then introduce the algorithm.

Algorithm 1 Short-term optimization (extension)

```
if cap(p) ≤ load(p) then
  for all p' ∈ (On(B_cld), Off(B_cld)) do
    if state(p') = 0 then
      activate p'
    end if
    if cap(p') − (load(p') + Σ_{v''∈queue(p')} load(v''))
    ≥ load(child(v')) then
      queue(p') ← append(queue(p'), v')
    end if
  end for
end if
```

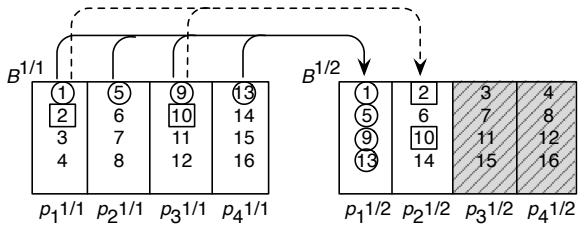

Figure 3. Example of migration (when system workload is increasing)

A. Short-term optimization algorithm for extension

When the system workload is increasing during the nominal period of a day, each active physical node behaves in the following way at regular intervals (e.g., every 30 minutes or one hour), which is called the short-term optimization algorithm for extension, consisting of the following two steps.

1) Each active physical node $p_i^{j/k}$ checks its own workload, and if the workload exceeds its capacity, then this node asks for the current capacity of every physical node in its child block (i.e., $p_i^{2j-1/2k} \in B^{2j-1/2k}$ for $i = 1, \ldots, c$).
2) If there are physical nodes in active mode that do not exceed their capacity, then $p_i^{j/k}$ chooses one of them as the destination of migration of its virtual node. (Note that, due to the manner of allocation, such a virtual node is uniquely determined when the destination is decided.) Otherwise, $p_i^{j/k}$ activates the leftmost among the physical nodes in a low-power mode in the child block, and then migrates the corresponding virtual node.

More formally, this procedure is presented in Algorithm 1. In this algorithm, B_{cld} is the child block of p, and "$queue$" represents the request queue for holding the requests for migration.

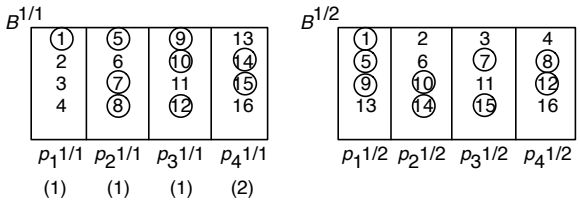

Figure 4. Example of migration (when system workload is decreasing)

To help the reader understand this process, we present a simple example (see also Fig 3 for the graphical presentation, where the nodes in low-power mode are denoted by gray shading). Initially, the workload of every virtual node in block $B^{1/1}$ is 10 and is equally shared by the child nodes when splitting. The capacity of every physical node is equal to 40. All the physical nodes in $B^{1/2}$ are in low-power mode. In this setting, when the workloads of all nodes in $B^{1/1}$ increase, each node migrates one of its virtual nodes according to the algorithm. Here, if at the beginning the workload of $p_1^{1/1}$ exceeds its capacity, then this node activates $p_1^{1/2}$ and $v_1^{1/2}$ is moved to this physical node, and subsequently the remaining nodes from $p_2^{1/1}$ to $p_4^{1/1}$ may choose the same physical node, $p_1^{1/2}$ instead of activating other nodes in $B^{1/2}$. Furthermore, if $p_1^{1/1}$ and $p_3^{1/1}$ become overloaded again, then these nodes can migrate their virtual nodes to $p_2^{1/2}$, so that $p_3^{1/2}$ as well as $p_4^{1/2}$ are still in a low-power mode. Such effective migration is realized by the manner of allocation.

B. Short-term optimization algorithm for reduction

After the peak-time of day and when the system workload is decreasing, reducing power consumption requires gathering the split virtual nodes into their parent blocks again. However, for effectiveness in power-saving, this cannot be realized solely by the reverse process of extension. We explain this by using a simple example. (See Fig 4 for a graphical presentation.) Let us consider the situation where the workload of $B^{1/1}$ becomes low and has room to absorb some of the virtual nodes in $B^{1/2}$. Now assume that the remaining capacity of $p_4^{1/1}$ is 2, while the other in $B^{1/1}$ is 1 (these are presented as parenthetical numbers below the physical nodes), and the virtual nodes at the points of the encircled numbers are migrated to $B^{1/2}$, thus, these can be migrated to $B^{1/1}$. In addition, the workload of every virtual node in B_{cld} is equal to 1. In this situation, the best solution for minimizing the active nodes in $B^{1/2}$ is to migrate $v_7^{1/2}$, $v_{10}^{1/2}$, $v_{14}^{1/2}$, and $v_{15}^{1/2}$, that enables both $p_2^{1/2}$ and $p_3^{1/2}$ to enter the low-power mode. Otherwise fewer nodes can be in a low-power mode or at least one physical node in $B^{1/1}$ is overloaded. This problem can be formally stated as follows.

Problem: For a given pair of blocks, B_{prt} and B_{cld}, with a parent-child relationship, find a maximal set S of the physical node in B_{cld} such that all the virtual nodes in S can be merged with their corresponding virtual nodes in B_{prt} without overloading any physical nodes in B_{prt}.

Algorithm 2 Short-term optimization (reduction)

```
function findSolution(M_1, M_2, ..., M_c)
N ← {M_1}, T ← φ
for all M_i ∈ {M_2, ..., M_c} do
  for all M ∈ M_i do
    for all M' ∈ N do
      M'' ← M ∩ M'
      if not is_include(T, M'') then
        T ← append(T, M'')
      end if
    end for
    N ← T, T ← φ
  end for
end for
return one of the maximum elements from N

function is_include(T, M)
for all S ∈ T do
  if M ⊆ S then
    return true
  end if
end for
return false
end function
```

The solution of this problem cannot be achieved when every node behaves independently. That is, our algorithm, called the short-term optimization algorithm for reduction, gathers the information of the workload at regular intervals and then calculates the best solution among all possible combinations of migration. This algorithm consists of the following four steps.

1) Each $p'_i \in B_{cld}$ sends the information of $load(v')$ for all $v' \in p'_i$ to all the physical nodes in B_{prt}.
2) Each $p_i \in B_{prt}$ lists all possible combinations of a subset of physical nodes in B_{cld} such that p_i can absorb their virtual nodes without exceeding its capacity.
3) All the results obtained in the previous step are gathered into a specific single node in B_{prt}, and the solution is found using Algorithm 2 below. Here $\mathcal{M}_i$ represents the result calculated by p_i.
4) The solution is announced to all the physical nodes in B_{cld}, and then the virtual nodes are gathered into B_{prt} according to this solution.

More formally, this procedure is presented in Algorithm 2. Intuitively, each element in $\mathcal{M}_i$ indicates one of the largest sets of physical nodes which can be in a low-power mode under the best situation possible for p_i. Thus,

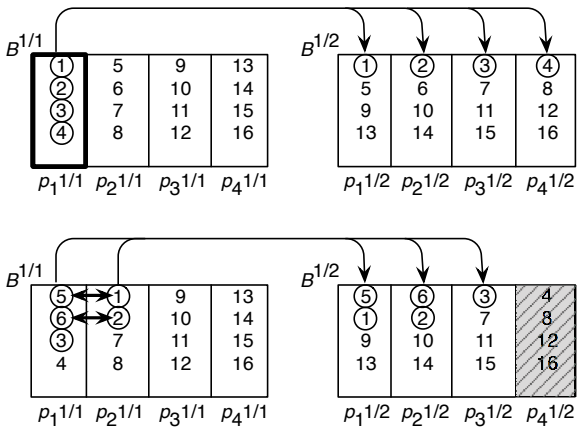

Figure 5. Example of inefficient migration and reconfiguration

every intersection of elements chosen singly from each $\mathcal{M}_i$ indicates a possible solution if all the physical nodes absorb as many of their virtual nodes as possible. Therefore, any of the largest intersections among all possible combinations is a solution of the problem.

We comment here on the computational complexity of this algorithm. Generally, for a given c of the number of physical nodes in a child block, the computational complexity to find the best solution is $(c-1)\cdot({}_cC_{\lfloor \frac{1}{2}c \rfloor})^3$, although the number of all possible combinations becomes 2^{c^2}. Indeed, as far as our experiments are concerned, we were able to find a solution in a feasible time for the condition $c \leq 15$. Note that, according to the value of 2^{c^2} for the input c, we were able to find a solution in a feasible time for the condition around $c \leq 5$, if we checked the results of all possible combinations one by one without any strategy.

In consideration of this result of computational complexity, for larger systems, we propose to separate a child block into several sub-blocks which consist of about 10 physical nodes, and apply the algorithm to every distinct sub-block. As we shall show in Section 6, we considered the condition that $c = 100$ by taking this approach in our simulation, and observed that it still effectively skews workload without search-space explosion.

C. Long-term optimization algorithm

So far we have explained how our method reduces power by moving virtual nodes according to the daily variation of the workload. However, to maintain effective power-saving, it requires load-balancing in each block. We explain this by using a simple example. (See Fig 5 for a graphical presentation.) In the upper two blocks, $B^{1/1}$ is the parent block of $B^{1/2}$, each of which consists of four physical nodes. The capacity of each node equals 10. We assume that due to changes in popularity of stored data, the workload of $v_i^{1/1} \in p_1^{1/1}$ (for $i = 1, \ldots, 4$) becomes 5 while the workload of the others is 2, so the physical node in $B^{1/1}$ is much busier than in the others. In such a case, this physical node should migrate its virtual nodes earlier than the others, and consequently all the physical nodes in $B^{1/2}$ should become active while their workload is low.

To avoid this problem, our method provides an auxiliary algorithm named the long-term optimization algorithm, which is intended for load-balancing in each block at regular relatively longer periods (such as, once a week). For example, in the case of Fig 5, $p^{1/1}$ detects the imbalance, and exchanges some of its virtual nodes (say, $v_1^{1/1}$ and $v_2^{1/1}$) with the low-loaded nodes on the same row (say, $v_5^{1/1}$ and $v_6^{1/1}$) in the same block. (The result is shown as the lower two blocks in the figure.) At that point, the allocation of all the other blocks is also changed in the same way. (Note that this migration requires the exchange of all data stored in each pair of virtual nodes between the different physical nodes. However, if the pair is on the same row, in half the blocks the corresponding pair is in the same physical node, hence we reduce the migration cost.) In this case, by this reconfiguration, the activation of $p_4^{1/2}$ is delayed, so we can reduce the active nodes in $B^{1/2}$.

However, in a more realistic situation, it is possible that the requirement for reconfiguration in one block conflicts with the requirement of another. In this paper, this problem is not thoroughly investigated and one of the future work.

V. Management of Replicas

In this section we explain how to introduce a mechanism for data replication into our proposed system for improving the migration cost and reliability. This can be realized by the following two additional operations when splitting or merging a virtual node. (See also Fig 6 for the graphical presentation. In this figure, the boxed items indicate the data kept in the original node, while the encircled items indicate replicas which are updated after the migration.) Here $\rho < c$ is the number of replicas.

- Split: When the virtual node $v^{j/k}$ on $p \in B_{prt}$ is split into $\langle v^{2j-1/2k}, v^{2j/2k} \rangle$ and $v^{2j-1/2k}$ is moved into the child block B_{cld}, the data stored in the key space of $v^{2j-1/2k}$ are kept in p. Also, after the split, the updated difference of stored data by $v^{2j/2k}$ is saved in ρ physical nodes in B_{cld}.
- Merge: When the virtual node $v^{2j-1/2k}$ on physical node $p' \in B_{cld}$ is merged with $v^{2j/2k}$ in the parent block B_{prt}, the data stored by $v^{2j-1/2k}$ are kept in p'. Also, after the merge, the updated difference of stored data by $v^{j/k}$ is saved in ρ physical nodes in B_{prt}.

If the system has enough space in its storage disks, this redundancy of data enables a reduction in data migration. That is, when splitting virtual node $v^{j/k}$, the data in $v^{2j-1/2k}$ saved at the previous merge operation can be reused, and it

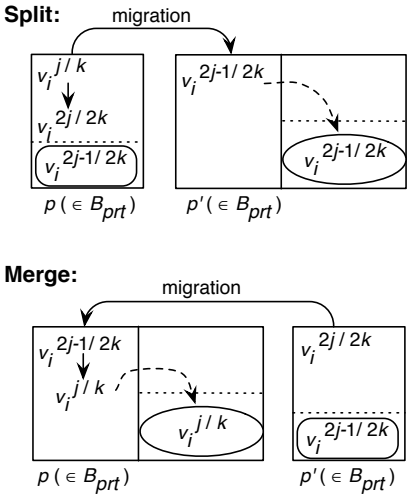

Figure 6. Data replication (with $\rho = 1$)

is necessary to migrate only the updated difference from the previous merge operation. Similarly, when merging virtual node $v^{2j-1/2k}$ with $v^{2j/2k}$, the data in the key space of $v^{2j-1/2k}$ saved at the previous split operation can also be reused.

Our proposed replication mechanism is also useful for data recovery from disk failure. That is, for a given number of replicas, ρ, any data in the system is accessible at any time against ρ disk failures.

VI. Simulation Results

To understand the effectiveness of our method for storage systems consisting of hundreds of physical nodes, we evaluated the average load (i.e., the ratio of workload to capacity) of the active physical nodes and the impact of the long-term optimization algorithm using simulations.

Parameters and settings. In the evaluation of this section, we considered the following system. Each block consisted of 100 physical nodes and the number of virtual nodes in each physical node is 100, thus each block included 10,000 virtual nodes. In the intended usage environment, we assumed that the system workload varies in a day. During the course of a day (that is modeled by discrete time intervals $t = 0, \ldots, 23$), the workload of all virtual nodes was initially at its lowest, and increased until the middle of the day then decreased until the end, where the gap was sixfold. We modeled workloads by natural numbers and assumed that the capacity of every physical node was equal to 100. In addition, due to the popularity of stored data, we considered two groups of virtual nodes with different workloads: group G_1 of 2000 nodes (initially allocated in $p_1^{1/1}, \ldots, p_{20}^{1/1}$) were the busier ones, while group G_2 of 8000 (in $p_{21}^{1/1}, \ldots, p_{100}^{1/1}$) were the normal nodes. In each group, the workloads of all virtual nodes were the same. The ratio of workloads of a node in G_1 to a node in G_2 was denoted by α, and

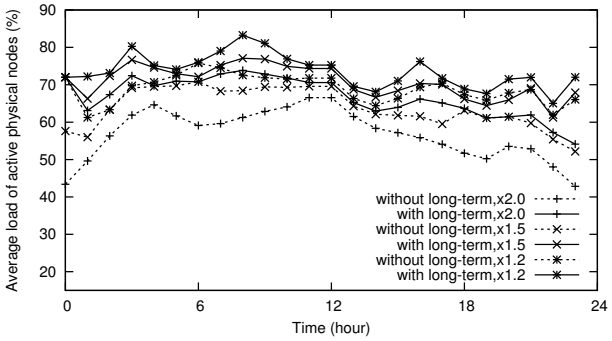

Figure 7. Average load of active physical nodes (with replication)

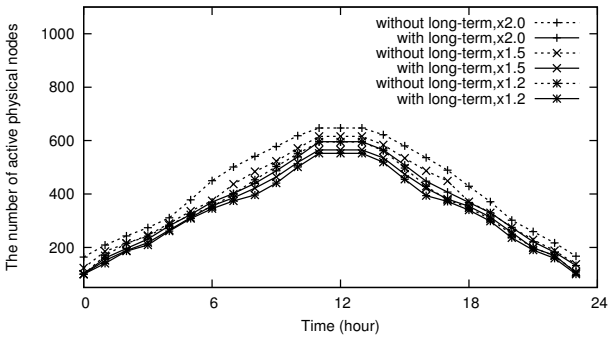

Figure 8. Number of active physical nodes (with replication)

we considered the cases that $\alpha = 1.2$, 1.5, and 2. In each case, we set the initial workload of the system to be 60% of its capacity, i.e. for $\alpha = 1.2$, 1.5, and 2, the initial workloads of a node in G_1 were respectively 0.69, 0.82, and 1, while in G_2 were respectively 0.58, 0.55, and 0.5. In the following simulations, we evaluated cases with the long-term optimization (which totally improves the bias of workload in each block) and without one. Due to space limitations, we only present the results for the case of single replication (i.e., $\rho = 1$).

Average load of active physical nodes. Fig 7 indicates the change in the average load of active physical nodes during a day. This figure shows that in the case of long-term optimization, for the cases of $\alpha = 1.2$, 1.5, and 2, the daily average load is 74%, 71%, and 67%. On the other hand, in the case of no long-term optimization, the daily average load is 69%, 64%, and 57%. The results of this simulation show that in our method the physical nodes run effectively, coping with the daily variation of workload. Moreover, compared with the case of no long-term optimization, this optimization algorithm improves the average load as expected.

Impact of long-term optimization. Fig 8 indicates the change in the number of active physical nodes during a day. This figure shows that long-term optimization saves on average 7%, 10%, and 14%, and up to 17%, 23%, and

39% in the cases that $\alpha = 1.2$, 1.5, and 2, compared with the case without long-term optimization. Moreover, this optimization improves the power consumption consistently and continually.

VII. Experiments with implementation

We conducted experiments with the current prototype implementation of our proposed system to evaluate the applicability of our method to real systems. In this paper, we measured the average load of the active physical nodes, the response time, the number of active nodes, and the number of migrations of virtual nodes in an environment where the system workload varies.

Our prototype consisted of 40 PC servers housed in five enclosures, each of which was equipped with Dual Xeon 3.60GHZ CPUs, 1.2–1.8GB memory, and a single 36GB SCSI disk. Migrations of virtual nodes were managed by the short-term optimization algorithms with the current workload (i.e., the number of requests for data access per unit time) monitored by each server. Here, the migration speed was saved so as to avoid the increase of response time. In our prototype, DHT for data management was not implemented, thus data were accessed randomly so that the data access frequency to each virtual node was the intended value. In addition, due to the limitation of our experimental environment, i.e. the bandwidth of different enclosures, we evaluated response time of data access by measuring time from sending a request until the data were loaded into memory in the server.

Parameters and settings. Servers (i.e., physical nodes) were logically grouped into four blocks, and each server had initially 10 virtual nodes. Besides these 40 servers, we used a single client of data access. Every virtual node stored 600 files whose size was equal to 1MB. In our experiments we did not consider data replication. As in the simulations, we considered two gruops of virtual nodes with different workloads: group G_1 of 20 nodes were the busier ones whose initial workloads were equal to 100 data access requests per minute, while group G_2 of 80 nodes were the normal ones whose initial workloads were equal to 50 requests per minute. The experiment duration was 24 hours and the workload of all virtual nodes was initially at its lowest, and increased until the middle then decreased until the end at regular 1-hour intervals, where the gap was sixfold. By some preliminary experiments, we observed that the capacity of each physical node is 1000 requests per minute to keep an expected response time, and in consideration of the migration cost, we set 90% and 60% of this capacity as the thresholds of migration for extension and reduction, respectively. In our experiments, we assumed that the amount of daily difference of data from the previous day is 10%, i.e. migration of virtual nodes was done by moving only this updating part, and compared it to the case that the whole data were migrated.

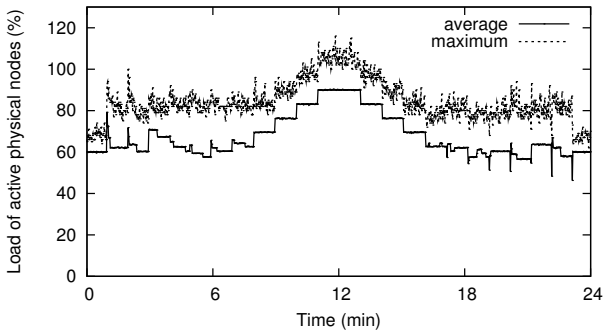

Figure 9. Average load of active physical nodes

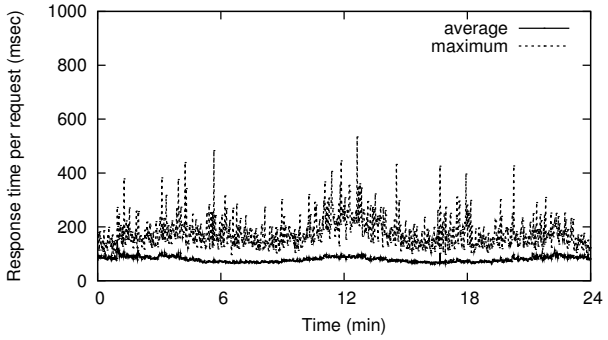

Figure 10. Average and maximum response time

Average load and response time. Figs 9 and 10 indicate the change in the average load of active physical nodes and the response time. Fig 9 shows that the overall average load is 67% of the capacity, although workloads of some nodes exceed the capacity in the first half part of this experiment. From another experiment, we observed that the overall average load was 68% when the whole data were migrated. Fig 10 shows that the overall average response time is 80msec, which is mostly as intended, although some responses are delayed. (Throughout this experiment, the maximun response time is 534msec.) In the case that the whole data were migrated, the overall average and maximum response time was respectively 158msec and 1436msec. The results of this experiment show that our optimization algorithms effectively skew the workload while keeping the intended response time in the real system.

Numbers of active physical nodes and migrations. Fig 11 indicates the change in the number of active physical nodes and the number of migrations. This figure shows that the migration is done on average 0.14 virtual nodes and up to 20.0 virtual nodes. In the case that the whole data were migrated, these values were 2.3 and 19.0, respectively. The result of this experiment shows that our system adjusts the number of physical nodes to the variation of workloads and reduces power effectively.

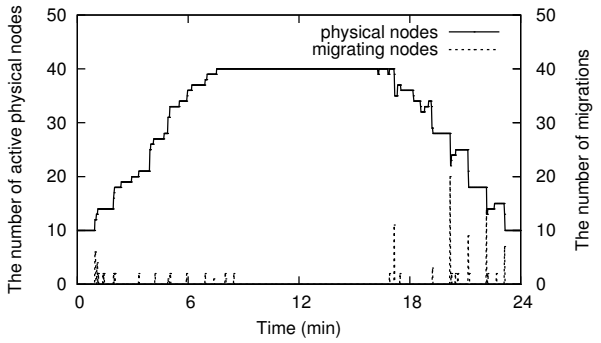

Figure 11. Number of active physical nodes

In closing this section, we would like to stress that the average response time can be reduced further by changing the threshold of migration for extension to smaller in exchange for the increase of power consumption.

VIII. Conclusions and future work

In this paper, we presented a power-saving method for large-scale distributed storage systems. To reduce power consumption, we introduced two kinds of optimization algorithms, thereby effectively skewing the workload towards a small number of physical nodes. We also introduced a method of data replication for both reducing migration costs and improving the reliability. Finally, the performance of our system was evaluated both by simulations and a prototype implementation. The simulation results showed that our method kept the workload of active physical nodes in terms of total capacity, on average, 67–74% for the case with single replication. The results of experiments with our implementation showed that the overall average load was 67%. At the same time, we maintained a preferred response time (whose overall average was 80msec) by setting a suitable capacity for each physical node.

In future work, we will further investigate to refine our prototype implementation. Especially, we will implement our proposed replication mechanism to improve response time, load balance, and reliability.

Acknowledgments

We would like to thank Munetoshi Ishikawa for his valuable comments. This study is partially supported through the SCOPE program by the Ministry of Internal Affairs and Communications of Japan. This study is also supported in part by the Grant-in-Aid for the Global COE Program on “Cybernics: fusion of human, machine, and information systems” at the University of Tsukuba.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. *Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley*, 2009.

[2] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *Proc. ACM/IEEE Conf. on Supercomputing*, pp.1-11, 2002.

[3] P. B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in dynamic structured p2p systems. In *Proc. IEEE INFOCOM*, 2004.

[4] D. Harnik, D. Naor, and I. Segall. Low power mode in cloud storage systems. *Parallel and Distributed Processing Symposium, International*, pp.1-8, 2009.

[5] D. Li and J. Wang. EERAID: energy efficient redundant and inexpensivedisk array. In *Proc. ACM SIGOPS European Workshop*, 6 pages, 2004.

[6] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *Proc. Int'l Conf. on Supercomputing*, pp.68-78, 2004.

[7] E. Pinheiro, R. Bianchini, and C. Dubnicki. Exploiting redundancy to conserve energy in storage systems. In *Proc. ACM SIGMETRICS Conf. on Measurement and modeling of computer systems*, pp.15-26, 2006.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM*, pp.149-160, 2001.

[9] M. Storer, K. Greenan, E. Miller, and K. Voruganti. Pergamum: Replacing tape with energy efficient reliable, disk-based archival storage. In *Proc. USENIX Conf. on File and Storage Technologies*, pp.1-16, 2008.

[10] J. Wang, H. Zhu, and D. Li. eRAID: Conserving energy in conventional disk-based raid system. *IEEE Trans. on Computers*, vol.57(3), pp.359-374, 2008.

[11] C. Weddle, M. Oldham, J. Qian, A. Wang, P. Reiher, and G. Kuenning. PARAID: A gear-shifting power-aware RAID. In *Proc. USENIX Conf. on File and Storage Technologies*, pp.245-260, 2007.

[12] X. Yao and J. Wang. RIMAC: a novel redundancy-based hierarchical cache architecture for energy efficient, high performance storage systems. In *Proc. ACM SIGOPS/EuroSys European Conf. on Computer Systems*, pp.249-262, 2006.

[13] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: helping disk arrays sleep through the winter. In *Proc. ACM symposium on Operating systems principles*, pp.177-190, 2005.