Using a Potential Game for Power Reduction in Distributed Storage Systems

Koji Hasebe, Takumi Sawada, and Kazuhiko Kato
Department of Computer Science, University of Tsukuba
1-1-1 Tennodai, Tsukuba 305-8573, Japan
hasebe@cs.tsukuba.ac.jp, sawada@osss.cs.tsukuba.ac.jp, kato@cs.tsukuba.ac.jp

***Abstract*—We present a game-theoretic approach for power reduction in large-scale distributed storage systems. The key idea is to use distributed hash tables to dynamically migrate virtual nodes, thus skewing the workload towards a subset of physical disks without overloading them. To realize this idea in an autonomous way (i.e., without any kind of central controller), virtual nodes are considered to be selfish agents playing a game in which each node receives a payoff according to the workload of the disk on which it currently resides. We model this setting as a potential game, where an increase in the payoff to a virtual node reduces the power of the system. This game consists of a pair of global and private utility functions, derived by means of the Wonderful Life Utility technique. The former function evaluates the state of the system, and the latter provides criteria for the migration of each node. The performance of our method is measured by simulations and a prototype implementation. From these evaluations, we find that our method reduces the running time of the disks in active mode by 12.7–18.7%, with an overall average response time of 50–190 ms.**

I. Introduction

With the advent of cloud computing, power reduction in datacenters has become a major concern. In particular, as a high percentage of the total computing system's energy is used by data storage systems, there have been a number of suggestions for reducing the power consumption of massive storage systems (cf. [1] for a comprehensive survey of this research area). To reduce power consumption in storage systems, a commonly observed technique is to skew the workload (i.e., data accesses) towards a small number of disks, thereby enabling the others to be in low-power mode. This idea can be traced back to some prominent early studies, such as MAID [2] and EERAID [7]; however, when considering datacenter-scale computing systems, scalability is of major importance.

In this paper, we propose a power-saving method for large-scale distributed storage systems. As our prime interest is to realize a higher level of scalability, we investigate autonomic control to skew data accesses in a storage array without any kind of central controller. To achieve this objective, we use distributed hash tables (DHTs) to provide a lookup service for data accesses (e.g., Chord [14]). Data are stored in virtual nodes of the underlying DHT, with each node dynamically migrating over physical nodes (disks) depending on the current workload of the resident disk. Here, the criteria for migration are as follows: for each disk, the higher the workload, the better it is for the system, although overload must be avoided. In other words, the optimal state is that in which every disk has as high a workload as possible, while not being overloaded. This idea was originally introduced in our previous work [5]. However, in that study, the migration destination of each virtual node was strictly predetermined in a specific way, and thus could not migrate freely to another disk that may yield a better state. Moreover, owing to this restriction, it was also difficult to change the system configuration by adding/removing disks (when extending the system or when node failure occurs), as well as to deal with changes in the distribution of popularity (i.e., data access frequency).

To overcome the flexibility issue, the key idea is to consider the virtual nodes as selfish agents, playing a game in which each of them receives a payoff according to the workload of the disk on which it currently resides. Here, the payoff is defined by satisfying certain migration criteria, and every virtual node can move to any of the neighboring physical disks. We model this setting as a potential game [9] by introducing global and private utility functions based on the Wonderful Life Utility [4], [8] (cf. [11] for an overview of game theory). In this model, increasing the payoff to any virtual node yields a better system state with respect to power consumption.

The performance of our method is measured using both a simulation and a prototype implementation in terms of the number of active physical nodes, average response time, and migration cost. In the experiments, we consider various changes in the environment that were difficult to deal with in our previous method. From the experiments, we observe that the number of active physical nodes can be reduced by 12.7–18.7% relative to the configuration without any dynamic migration. This result indicates that our method effectively skews the workload while maintaining a preferred response time (with an overall average of 50–190 ms) by setting suitable parameters in the utility functions. In addition, the results confirm that our method improves flexibility.

This paper is organized as follows. Section II discusses related work, and Section III presents our game-theoretic model for power-saving. Sections IV and V present evaluations using simulations and a prototype implementation, respectively. Section VI concludes the paper and presents some ideas for future work.

II. Related Work

There have been a number of suggestions for power-saving in storage systems. As explained in the previous section, these are based on a similar idea, but are classified into the following categories according to variations in their approach.

The first category, which includes MAID [2] and PDC [12], focuses on popularity by concentrating popular data on specific disks. MAID provides specific disks to act as a cache to store frequently accessed data, thereby reducing accesses to the other disks. PDC periodically reallocates data in the storage array according to the latest access frequencies.

The second category uses non-volatile random-access memory (NVRAM) to extend the standby mode period by caching data to a write store. A typical example is Pergamum [15], which uses NVRAM to buffer write accesses and store data signatures, reducing direct accesses to the disks.

The final category considers redundancy (i.e., data replication). In DIV [13], original and redundant data are separated onto different disks, thereby allowing read/write requests to be concentrated on those disks that contain the original data. Hibernator [20] applies the idea of PDC to RAID and dynamic rotations per minute (DRPM) systems. RIMAC [19] provides two-layered caches, one for storing storage data and the other for parity conservation. PARAID [18], which is also a power-saving technique for RAID, allocates the replica in a specific way, meaning that data are collected/spread to adapt to changes in operational workloads.

Although the above studies restricted their scope to storage systems consisting of a relatively small number of disks (typically up to several dozen), in recent years the target of this research area has shifted to datacenter-scale systems. Kaushik et al. [6] proposed the idea of dividing disks in Hadoop distributed file systems (HDFS) into hot and cold zones. Verma et al. [16] developed sample-replicate-consolidate mapping (SRCMap), which gathers accesses to the replicas on active disks, while Vrbsky et al. [17] proposed a replication approach called the sliding window replica strategy (SWIN). Finally, Hasebe et al. [5] reported a power-saving method based on the DHT technique that skews the workload by dynamically migrating virtual nodes in the storage array.

The motivation for this research follows that of the recent studies mentioned above. In particular, this study can be considered as a direct successor to [5]. However, our main aim is to increase scalability and flexibility by means of game-theoretic autonomic control, thereby making it possible to deal with various changes in the system environment.

III. System Design

A. Underlying System

Our proposed method targets storage systems consisting of hundreds of physical disks with unique IDs. The basic lookup service is provided by an underlying DHT mechanism, and the stored data are managed by collaboration between virtual nodes. Owing to the DHT mechanism, clients may access data by sending a request to any of the virtual nodes, regardless of their location. In this setting, to reduce power consumption, each virtual node can migrate to any of the neighboring physical disks depending on their current workload. (Note that a similar idea for the dynamic migration of virtual nodes can be found in [3] in the context of load balancing.) In the next subsection, we introduce our modeling of this mechanism as a potential game.

B. System Model as a Potential Game

Preliminaries. As preliminaries, we first introduce some notation and functions. In the following, $\mathbb{N}$ is used to denote the set of natural numbers. Let $P = \{p_1, p_2, \dots, p_n\}$ be the set of physical nodes and $V = \{v_1, v_2, \dots, v_m\}$ be the set of virtual nodes. Time progress is represented as steps indicated by a natural number $t = 0, 1, 2, \dots \in T$, where 0 indicates the time of the initial state. Every virtual node is stored on a physical node, with the allocation determined by the function $place : V \times T \to P$. Intuitively, $place_t(v) = p$ means that virtual node v is on p at time t. We also use this function to denote the set of virtual nodes $V' \subseteq V$ that are on physical node p, i.e., $place_t(p) = V'$. In addition, the notation $v \in P$ is used to denote this relation. For each physical node, the capacities of workload and volume are respectively determined by the functions $cap_{load} : P \to \mathbb{N}$ and $cap_{vol} : P \to \mathbb{N}$. The current workload and total volume of stored data of a virtual node at any time are respectively determined by the functions $load : V \times T \to \mathbb{N}$ and $size : V \times T \to \mathbb{N}$. These functions are also used for physical nodes according to the following definitions: $load(p,t) = \sum_{v \in place_t(p)} load(v,t)$ and $size(p,t) = \sum_{v \in place_t(p)} size(v,t)$, respectively. Note that no physical node is allowed to have virtual nodes whose total stored data volume exceeds its volume capacity (i.e., $cap_{vol}(p) \leq \sum_{v \in place_t(p)} size(v_i,t)$ for any t). In addition, p is considered to be overloaded if $load(p,t) > cap_{load}(p)$.

The central idea. Before giving a formal definition of our game-theoretic model, we briefly describe the central idea behind our modeling approach. Our model can be regarded as a potential game, a kind of strategic game in which the incentive for any player to change their strategy can be expressed by a single global function. More specifically, our model is developed according to the following steps, in conjunction with the Wonderful Life Utility technique.

In our model, virtual nodes are considered to be selfish players who select their strategies to remain on the current physical disk or move to another at regular intervals. After the decision, each virtual node receives a payoff, and thus every virtual node looks for a better location that will yield a higher payoff. In this setting, we first introduce the global function for evaluating the state of the system with respect to power-saving and overloading. Next, using the Wonderful Life Utility technique, we derive a local function, which determines the payoff to virtual nodes, from the global function. This local function satisfies the following criteria: if the workload of the disk on which it resides increases, the payoff to the node also increases, although the payoff is cut drastically if the resident disk becomes overloaded. Moreover, the relationship between the global and local functions can be regarded as a potential game; that is, increasing the payoff to any virtual node causes an increase in the value of the global function.

Formal definition of the global function. We first introduce the global function G to evaluate the state of the system. However, to respond to rapid changes in the system workload, it estimates the expected state at some later step provided that the rate of change in the workload during the past j steps is the same in the future. The formal definition of the global function is as follows:

$$G(place_t) = \sum_{i=0}^{s} \sum_{p \in P} (load_{pred}^2(p,i,t) - c \cdot over_{pred}^2(p,i,t)).$$

Here, the function $load_{pred}(p,i,t)$ indicates the expected workload of p at time $i+t$ (i.e., i steps later), and is defined as follows:

$$load_{pred}(p,i,t) = load(p,t) + \frac{i}{j}(load(p,t) - load(p,t-j)).$$

The function $over_{pred}(p,i,t)$ indicates the level of overload for p, and is defined as follows:

$$over_{pred}(p,i,t) = \beta \cdot cap_{load}(p) - load_{pred}(p,i,t),$$

where β (with $0 < \beta \leq 1$) is a coefficient indicating the proportion of capacity that may be devoted to migration. For example, $\beta = 0.6$ means that the physical node may spare 60% of its capacity to respond to data access requests, while the remainder (i.e., 40% of its capacity) is kept for migration. c is a weighting constant with $c > (s+1)\sum_{p \in P} cap_{load}^2(p)$.

In the definition of G, $load_{pred}^2(p,i,t)$ gives a positive evaluation for effectively skewing the workload, whereas $-c \cdot over_{pred}^2(p,i,t)$ gives a negative evaluation according to the level of overloading. We note that the argument of G (i.e., $place_t$) determines the allocation of virtual nodes at time t. Thus, in game-theoretic terminology, it is a strategy profile of the virtual nodes.

Formal definition of the local function. Next, in terms of the Wonderful Life Utility technique, the global function defined above can be used to derive a local function L_v for node $v \in V$ as follows.

$$\begin{aligned} L_v(place_t) \quad = \quad & \sum_{i=0}^{s} [(load_{pred}^2(place_t(v),i,t) \\ & -load_{pred}'^2(place_t(v),i,t)) \\ & -(c \cdot over_{pred}^2(place_t(v),i,t) \\ & -c \cdot over_{pred}'^2(place_t(v),i,t))]. \end{aligned}$$

Here, $load'_{pred}(p,i,t)$ and $over'_{pred}(p,i,t)$ denote the values obtained from $load_{pred}(p,i,t)$ and $over_{pred}(p,i,t)$, respectively, by removing v from p.

C. Migration algorithm

In our proposed method, every virtual node independently checks its current payoff at regular intervals, and migrates to another physical disk if it yields a better payoff. Because the model introduced above is a potential game, any migration that improves a virtual node's payoff increases the value of the global function. However, owing to the limitation of network bandwidth in a system, it is difficult to allow all virtual nodes to migrate as they wish. Thus, we introduce a priority order according to the degree of contribution to improving the value of the global function (i.e., the amount by which the value of the global function increases).

As the degree of contribution, we consider the increase in the payoff of migration divided by the time required to complete the migration. The formal definition is as follows:

$$Mig_{eval}(v, p_{from}, p_{to}) = \frac{L_v(place') - L_v(place)}{Mig_{time}(v, p_{from}, p_{to})},$$

where p_{from} and p_{to} ($\in P$) denote the source and destination of migration, respectively, and Mig_{time} represents the time required to migrate v from p_{from} to p_{to}. This function is defined as follows:

$$Mig_{time}(v, p_{from}, p_{to}) = \frac{diff(v, p_{from}, p_{to})}{\min\{bw(p_{from}), bw(p_{to})\}},$$

where *diff* denotes the total amount of data that must be transferred for the migration of v from p_{from} to p_{to}, and bw represents the bandwidth of p, defined as $bw(p) = (1-\beta)cap_{load}(p)$ (with $0 < \beta \leq 1$). Note that β is the same predefined constant as in the global function.

In a real system, the evaluation of payoffs and migration are autonomously controlled by the physical nodes on the basis that the virtual nodes behave as selfish agents. Roughly speaking, every physical node (denoted by p) independently collects information to evaluate the payoffs from its neighboring nodes. (As we shall see in Section IV, the set of neighbors for each physical node is predetermined. This set consists of 100 neighbors in our simulation.) Then, p selects the best migration, and sends a request to the destination. The destination node decides whether to accept or reject the request. If it is accepted, the migration starts immediately; otherwise, it is rejected. Note that, to reduce the migration cost, physical nodes are allowed to send or receive one virtual node at a time.

More precisely, every physical node (denoted by $p \in P$) independently executes the following steps at regular intervals. (Here, the current time is indicated by $t \in T$.)

1) p obtains the following information from each of its neighboring nodes (denoted by $p' \in P$).
   - Estimated workload: $load_{pred}(p',i,t)$;
   - Amount of free space: $cap_{vol}(p') - size(p')$;
   - The value of the migration that p is executing: $Mig_{eval}(v, p_{from}, p_{to})$, where v is the virtual node that is currently migrating from/to p'.
2) Among all possible migrations for the virtual nodes in p and the migrations previously requested to p, find the best migration (whose virtual node is denoted by v) yielding the largest $Mig_{eval}(v,p,p')$. If there is no such v (i.e., $Mig_{eval}(v,p,p') \leq 0$ for any v), then p quits and returns to Step 1 in the next time interval. Note that the currently migrating virtual node is also included as a candidate.
3) Execute one of (3-a)–(3-d) according to the current state:

(3-a) If v is currently migrating from p to p', then the migration continues;

(3-b) If $v \in p$ and p is asking its neighbor to receive v, but this has not yet been approved, then p continues to wait for a response;

(3-c) If $v \in p$ and p has not asked its neighbor to receive v, then p drops its current request that was previously sent to a neighbor, and asks the destination to receive v.

(3-d) If $v \notin p$, then p drops the request that was previously sent by p to a neighbor and the migration currently being executed, and then approves the migration of v;

4) Reject all other requests from the neighbors.

In closing this section, we comment on the migration cost. Instead of moving all data stored in a virtual node, there is an alternative that reduces the volume of data to be migrated. That is, when a virtual node is removed from a disk, the stored data remain in the disk and are reused at the time of the next migration. This enables the migration to proceed by copying only those data that are different from the previous residence at that physical node. Indeed, the advantage of this technique is the trade-off with the disk space. However, if the system has enough space on the disk and can afford to create some redundancy, this technique can be useful for effective migration. We adopt this technique in our simulation and experiments, as reported in later sections.

IV. Simulation Results

To understand the scalability of our method for storage systems consisting of thousands of physical nodes, we conducted a series of simulations in various environments. These evaluated the performance by observing the change in the number of active physical nodes, average response time, and cumulative number of migrations.

A. Parameters and Settings

In the evaluation, we considered a system consisting of 1,000 physical nodes whose volume and workload (i.e., maximum transfer rate of data access) capacities are 500 GB and 55 MB/s, respectively. As per the suggestion in [14], the number of virtual nodes is 10 times the number of physical disks. The system stores 200 million files, each of size 500 KB, and these are randomly allocated between the virtual nodes. In the intended usage environment, we assumed that the system workload varies over a day.

During the course of a day, as modeled by discrete time intervals t (s) with $0 \le t \le 3{,}600 * 24$, the workload of all virtual nodes is initially at its lowest, then increases until the middle of the day, before decreasing until the end, where the gap is 4-fold. As suggested by many studies (e.g., [3], [12], [10]), we assumed that the data access frequency conforms to a Zipf distribution [21]. According to this law, the access frequency of the r-th most popular file is determined by the following formula:

$$load_r(t) = load_{sys}(t)\frac{\frac{1}{r^\alpha}}{\sum_{i=1}^{F}\frac{1}{i^\alpha}},$$

where F is the total number of files, $load_{sys}$ indicates the total number of data accesses, and α is a coefficient. In our simulation, we consider the case where $\alpha = 0.7$. Thus, the workload of the system is determined by the following formula:

$$load_{sys}(t) = \begin{cases} 22,000 * (\frac{3t}{12*3,600} + 1) & (0 \le t < 12*3,600) \\ 22,000 * (-\frac{3t}{12*3,600} + 7) & (12*3,600 \le t < 24*3,600). \end{cases}$$

For example, the total number of accesses is 22,000 during the off-peak period, rising to 88,000 at peak time. Note that if we consider a static configuration (i.e., without our method), it is necessary to make 800 physical nodes active to deal with the workload in this setting.

As mentioned in Section III, after migrating a virtual node, the stored data remain at the source physical disk unless it overflows. Thus, we reuse the remaining unchanged data when the virtual node returns to its original position. We assumed that 10% of files are rewritten by write accesses during the course of a day.

Finally, the remaining parameters were as follows: In the global function defined in Section III-B, we set $s = 30 * 60$, $\beta = 0.9$, $j = 15$. The neighbors for each physical node p_i are $\{p_j \mid i - 50 \le j \le i - 1, i + 1 \le j \le i + 50\}$. Note that the future workload was estimated using only the workloads of 15 and 30 minutes ago.

B. Number of Active Physical Nodes

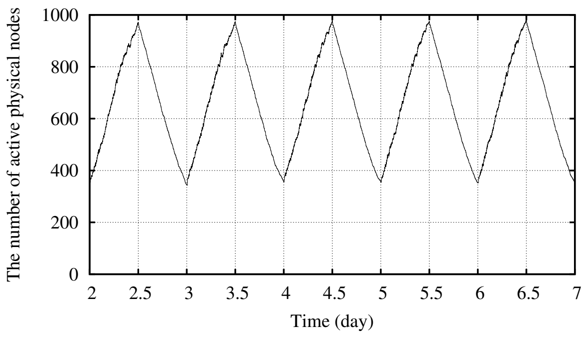

Fig. 1. Number of active physical nodes

Fig. 1 indicates the change in the number of active physical nodes for the 5 days following the initial configuration. (Throughout this section, we omit results from the first 2 days, because from the 3rd day, reusable data are spread across the system by the migrations in the first 2 days.) Fig. 1 shows that the number of active physical nodes, indicating the total running time, was reduced by an average of 17.3% relative to the static configuration.

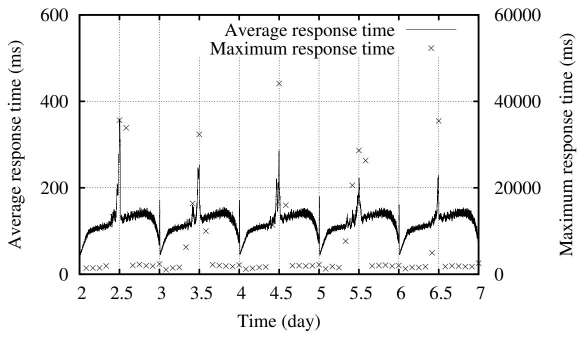

Fig. 2. Response time

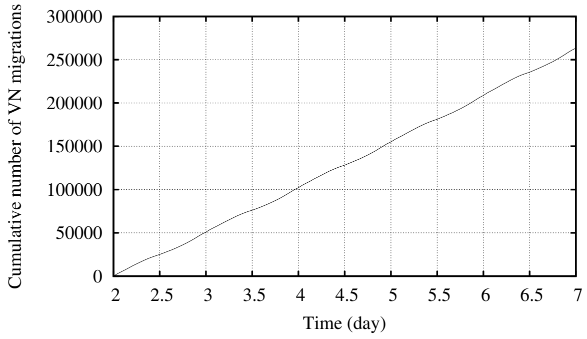

Fig. 3. Cumulative number of migrations

TABLE I
SIMULATION RESULTS FOR CASES 1–4

	Num. of Nodes [%]	Ave. Res. [ms]	Max Res. [ms]
Case 1	14.4	142	79,638
Case 2	12.7	190	159,358
Case 3	18.7	130	157,345
Case 4	16.9	128	56,124

C. Response Time and Migration Cost

Figs. 2 and 3 indicate the change in the overall average and maximum response time and the cumulative number of migrations during the 5 days, respectively. Fig. 2 shows that the overall average response time was 121 ms, although the maximum was 44,134 ms. The results of this experiment show that roughly the same number of virtual nodes were migrating regardless of the system workload. For each virtual node, there were an average of 5.27 migrations. In our previous work [5], the number of migrations required of each virtual node was about 2 in a similar environment, which is fewer than half the number required using this method.

Our method has room for improvement in the response time at peak times, as well as in reducing the number of migrations. On the whole, however, our method effectively skews the workload while keeping a preferred overall average response time in a large-scale system.

D. Flexibility toward Various Environments

To evaluate our method in various environments, we considered the following cases, which cannot be treated using our previous method [5]:

Case 1: Peak times of some virtual nodes are different from others.
Case 2: Distribution of popularity changes over time.
Case 3: Some physical nodes are added to the system.
Case 4: Some physical nodes are removed from the system.

For Case 1, we considered the environment where 80% of the virtual nodes encounter peak time at noon, while for the others this occurs at midnight. For Case 2, we changed the difference in workload between peak and off-peak times to be 8-fold for 10% of the virtual nodes, and to be a factor of 32/9 for the other 90%. For Cases 3 and 4, we added 50 and removed 30 physical nodes, respectively.

The results for each case are summarized in Table I. (Here "Num. of Nodes", "Ave. Res.", "Max Res." denote the results of reducing the number of active physical nodes relative to the static configuration, overall average response time, and maximum response time, respectively.) Overall, although in some cases the performance worsened, our method was able to effectively skew the workload, adapting to various changes in environment.

V. IMPLEMENTATION EXPERIMENTS

We conducted experiments with the current prototype implementation to evaluate the applicability of our method to real systems. In the experiments, we measured the change in the number of active physical nodes and the response time in an environment with a variable system workload.

Our prototype consisted of 28 PC servers, each of which was equipped with Intel Core i7 2.67 GHz $\times$ 8 CPUs, 11.8 GB memory, and a single 500 GB ATA disk. In this implementation, we omitted the DHT mechanism.

A. Parameters and Settings

The system consisted of 300 virtual nodes, each of which stored 5.6 million files of size 500 KB. In this experiment, the distribution of data access frequency was also assumed to conform to Zipf's law. At off-peak times, the system workload was 1,209,600 accesses/s, and this increased to 4,838,400 accesses/s at peak times, similarly to the simulations.

For this setting, if we consider a static configuration, it is necessary to make 23 physical nodes active to deal with the workload.

B. Number of Active Physical Nodes

Applying this configuration, we observed the performance for 24 h. Fig. 4 indicates the change in the number of active physical nodes. The figure shows that our method reduced this number by an average of 14.1% relative to the static configuration. This result shows that our method adjusts the number of physical nodes to changes in workload, and reduces power consumption effectively.

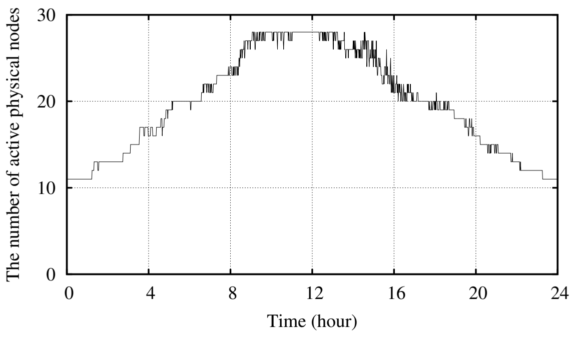

Fig. 4. Number of active physical nodes

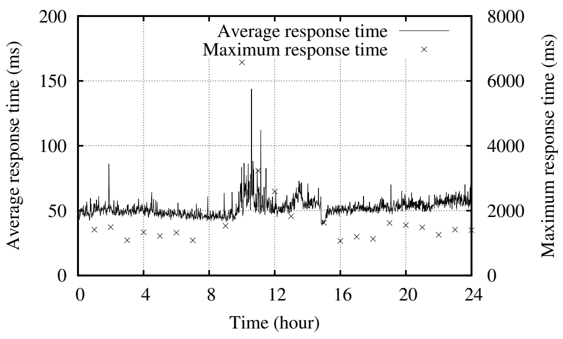

Fig. 5. Response time

C. Response Time

Fig. 5 indicates the change in the average and maximum response times. This figure shows that the overall average response time was 50 ms, while the maximum was 6,573 ms. We observed some long delays in responses at peak time, but on the whole, our method retains an intended latency in the real system.

VI. Conclusions and Future Work

In this paper, we presented a game theoretic approach for reducing power consumption in datacenter-scale storage systems. To enhance the scalability in a commonly-observed technique, our method was based on DHTs to dynamically migrate virtual nodes, thereby skewing the workload and avoiding overloads. Moreover, to improve the flexibility problem in the method of [5], we investigated a technique that allowed virtual nodes to freely migrate to any neighboring disks, thereby enabling autonomous adaptation to various changes of environment, such as changes in the popularity distribution, extension of a system, and node failures. For this objective, the idea behind our method was to consider virtual nodes as selfish agents, and play a game in which each node receives a payoff according to the workload of the disk on which it currently resides. This idea was modeled as a potential game using the Wonderful Life Utility technique. We also evaluated the performance of our method using simulations and a prototype implementation. The results showed that our method effectively skewed the workload while maintaining a preferred response time in a large-scale distributed environment.

In future work, we will improve the migration algorithm so as to reduce the latency at peak-time. We are also interested in an evolutional mechanism to allow virtual nodes to autonomously change the parameters of local payoff functions according to the current environment, with the aim of achieving a better optimization.

References

[1] T. Bostoen, S. Mullender and Y. Berbers. Power-reduction techniques for data-center storage systems. *ACM Computing Surveys*, vol.45(3), 38 pages, 2013 (to appear).

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

[3] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp and I. Stoica. Load balancing in dynamic structured p2p systems. *INFOCOM*, vol.4, pp.2253-2262, 2004.

[4] R. Gopalakrishnan, J. R. Marden and A. Wierman. An architectural view of game theoretic control. *SIGMETRICS Perform. Eval. Rev.*, vol.38, no.3, pp.31-36, 2010.

[5] K. Hasebe, T. Niwa, A. Sugiki and K. Kato. Power-saving in large-scale storage systems with data migration. *IEEE International Conference on Cloud Computing Technology and Science (CloudCom'10)*, pp.266-273, 2010.

[6] R. T. Kaushik and M. Bhandarkar. GreenHDFS: towards an energy-conserving, storage-efficient, hybrid Hadoop compute cluster. *2010 international conference on Power aware computing and systems (HotPower'10)*, pp.1-9, 2010.

[7] D. Li and J. Wang. EERAID: energy efficient redundant and inexpensive disk arrays. *11th ACM SIGOPS European Workshop*, Article no.29, 6 pages, 2004.

[8] J. R. Marden and A. Wierman. Distributed welfare games. *Operations Research*, vol.61, no.1, pp.155-168, 2013.

[9] D. Monderer and L. S. Shapley. Potential games. *Games and Economic Behavior*, vol.14, no.1, pp.124-143, 1996.

[10] J. Okoshi, K. Hasebe, and K. Kato. Power-Saving in Storage Systems for Internet Hosting Services with Data Access Prediction. *4th International Green Computing Conference (IGCC 2013)*, 10 pages, 2013.

[11] M. J. Osborne and A. Rubinstein. A course in game theory. *The MIT Press*, 1994.

[12] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. *International Conference on Supercomputing*, pp.68-78, 2004.

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

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

[15] M. Storer, K. Greenan, E. Miller and K. Voruganti. Pergamum: replacing tape with energy efficient reliable, disk-based archival storage. *USENIX Conference on File and Storage Technologies (FAST'08)*, pp.1-16, 2008.

[16] A. Verma, R. Koller, L. Useche and R. Rangaswami. SRCMap: energy proportional storage using dynamic consolidation. *8th USENIX Conference on File and Storage Technologies (FAST'10)*, pp.154-168, 2010.

[17] S. V. Vrbsky, M. Lei, K. Smith and J. Byrd. Data replication and power consumption in data grids. *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom'10)*, pp.288-295, 2010.

[18] C. Weddle, M. Oldham, J. Qian, A. Wang, P. Reiher and G. Kuenning. PARAID: a gear-shifting power-aware RAID. *USENIX Conference on File and Storage Technologies (FAST'07)*, pp.245-260, 2007.

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

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

[21] G. K. Zipf. *Human behavior and the principle of least effort*. Addison-Wesley Press, 1949.