Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

A reliable power management scheme for consistent hashing based distributed key value storage systems#

Nan-nan ZHAO[1,2], Ji-guang WAN‡[1,2], Jun WANG[3], Chang-sheng XIE[1,2]

([1]*Wuhan National Laboratory for Optoelectronics, Wuhan 430074, China*)

([2]*Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*)

([3]*Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816, USA*)

E-mail: nnzhaocs@hotmail.com; jgwan@mail.hust.edu.cn; jwang@mail.ucf.edu; cs_xie@mail.hust.edu.cn

Received Apr. 13, 2016; Revision accepted June 3, 2016; Crosschecked Sept. 20, 2016

Abstract: Distributed key value storage systems are among the most important types of distributed storage systems currently deployed in data centers. Nowadays, enterprise data centers are facing growing pressure in reducing their power consumption. In this paper, we propose GreenCHT, a reliable power management scheme for consistent hashing based distributed key value storage systems. It consists of a multi-tier replication scheme, a reliable distributed log store, and a predictive power mode scheduler (PMS). Instead of randomly placing replicas of each object on a number of nodes in the consistent hash ring, we arrange the replicas of objects on nonoverlapping tiers of nodes in the ring. This allows the system to fall in various power modes by powering down subsets of servers while not violating data availability. The predictive PMS predicts workloads and adapts to load fluctuation. It cooperates with the multi-tier replication strategy to provide power proportionality for the system. To ensure that the reliability of the system is maintained when replicas are powered down, we distribute the writes to standby replicas to active servers, which ensures failure tolerance of the system. GreenCHT is implemented based on Sheepdog, a distributed key value storage system that uses consistent hashing as an underlying distributed hash table. By replaying 12 typical real workload traces collected from Microsoft, the evaluation results show that GreenCHT can provide significant power savings while maintaining a desired performance. We observe that GreenCHT can reduce power consumption by up to 35%–61%.

Key words: Consistent hash table (CHT), Replication, Power management, Key value storage system, Reliability
http://dx.doi.org/10.1631/FITEE.1601162 **CLC number:** TP316.4

1 Introduction

Server power consumption is an important issue for distributed storage systems because it contributes substantially to a data center's power bills (Thereska *et al.*, 2011). Furthermore, as the size of data increases, a massive amount of storage becomes necessary (Kaushik *et al.*, 2010). Hence, server power consumption has become an urgent priority.

Many schemes (Amur *et al.*, 2010; Thereska *et al.*, 2011) have been proposed for power proportionality: the energy consumed should be proportional to the system load. The literature on data center energy management shows that most workloads in data centers have periodic I/O load patterns with large peak-to-trough ratios (Thereska *et al.*, 2011). A great deal of research has been conducted for power reduction in common distributed file storage systems

‡ Corresponding author

A preliminary version of this paper was presented at the 31st International Conference on Massive Storage Systems and Technology, Santa Clara, CA, USA, June 1–5, 2015

ORCID: Nan-nan ZHAO, http://orcid.org/0000-0002-7219-7856

by powering down storage devices or servers during periods of light loads. These studies have illustrated significant power saving potential (Amur *et al.*, 2010; Kaushik and Bhandarkar, 2010; Kaushik *et al.*, 2010; Thereska *et al.*, 2011). A notable method is to exploit the inherent redundancy of replicas in distributed storage systems (Pinheiro *et al.*, 2006; Harnik *et al.*, 2009; Thereska *et al.*, 2011). This method involves maintaining a primary replica available for access, and powering down the servers hosting redundant replicas to provide a high level of energy conservation (Pinheiro *et al.*, 2006; Thereska *et al.*, 2011).

Our focus is on distributed key value storage systems. Key value storage systems have attracted significant interest over the past few decades (Bhagwan *et al.*, 2003), as they are being broadly used by many popular networking corporations. Modern key value storages often use a consistent hash table (CHT) to distribute data to storage nodes because consistent hashing can adapt to a changing set of nodes with minimal disruption (DeCandia *et al.*, 2007).

In this paper, we study the placement of the data under the consistent hashing algorithm and especially the existing replication strategy for consistent hashing. Generally, consistent hashing maps objects to nodes in a pseudo-random order (Karger *et al.*, 1997; Stoica *et al.*, 2001); thus, assuming that the hash function is 'random enough' and that numerous objects are inserted, powering down a subset of nodes may violate data availability (Harnik *et al.*, 2009). Current replication schemes for CHT do not provide power proportionality by imposing a strong constraint on how many nodes can be powered down to conserve energy without disrupting data availability.

This paper first presents a practical power-proportional replication scheme for consistent hashing based distributed key value storage systems to achieve better power savings while meeting the availability requirements. We use a multi-tier placement of replicas (M-CHT) that assigns the replicas on nonoverlapping tiers of nodes in the consistent hash ring. In this case, we can maintain a number of tiers of nodes that will be made available and power down the remaining tiers of nodes without violating data availability. Moreover, it co-operates a predictive power mode scheduler (PMS) to provide power proportionality for distributed key value storage systems. By powering down different numbers of tiers of nodes, the system can switch to different power modes to meet different performance and power-saving requirements. Accordingly, the performance can be scaled up in a power-proportional manner; that is, the power consumed is proportional to the workload.

Although recent research and developments have suggested that the most promising approach to saving power in enterprise data centers is to power entire servers down rather than saving power in individual components (Thereska *et al.*, 2011), powering servers down in distributed key value storage systems is challenging since reliability may be compromised. To cope with this problem, we present a distributed log store to maintain reliability: when a subset of servers is powered down, all of the writes to powered-down replicas are distributed to the active successors in a reliable order, which not only ensures reliability but also maintains write parallelism.

In this paper, we present GreenCHT, which includes the above three major components: a power-aware replication strategy, namely a multi-tier replication scheme (M-CHT), a reliable distributed log store, and a predictive PMS. We have implemented GreenCHT on Sheepdog (NTT Group, 2011) as a power-proportional key value storage system. We deploy the GreenCHT system on an Ethernet-based storage cluster. Evaluation results show that GreenCHT can provide significant power savings of up to 35%–61% while maintaining good performance and reliability.

2 Related work

In this section, we first describe power management in both a disk array and a distributed system. In addition, we present various studies on renewable energy solutions.

2.1 Power management in a disk array

Recently, energy management has received considerable attention. There exists a large body of research on reducing power consumption in a disk array. They exploit long idle periods and spin down a subset of disks for energy saving.

Massive array of idle disks (MAID) (Colarelli and Grunwald, 2002) dedicates a small number of drives as cache drives that cache recently accessed blocks. The cache drives would always remain active

to reduce the need to spin up idle data disks. Unfortunately, cache disks result in more energy usage (Zhu *et al.*, 2005). Popular data concentration (PDC) (Pinheiro and Bianchini, 2004) is inspired heavily by data popularity, and migrates frequently accessed data to a few disks, so that the remaining disks can be powered down. However, PDC incurs substantial performance degradation due to load imbalance. Write off-loading (Narayanan *et al.*, 2008) exploits significant idle periods and redirects write requests on spun-down disks to logs stored on persistent storage and reclaimed in the background when the original disks are spun up. However, reads that go to the spun-down disks are affected. Power-aware RAID (PARAID) (Gorini *et al.*, 2007) applies the notion of gears to RAID arrays and uses a skewed striping pattern to adapt to the system load by adjusting the number of active disks and replicating blocks on the active disks. However, PARAID offers less parallelism and bandwidth in low gears. Moreover, maintaining replica consistency wastes both capacity and write bandwidth. Hibernator adapts to load changes by changing the number of disks and their rotational speed in a RAID array (Zhu *et al.*, 2005). However, multi-speed disks are not widely available for most enterprises today.

2.2 Power management in a distributed system

A number of recent works have attempted to reduce power consumption in distributed systems. Sierra (Thereska *et al.*, 2011) exploits the redundancy in distributed systems and powers down the servers hosting inactive replicas. In a three-way replicated system, Sierra can power down $(R-1)/R$ storage servers and ensure that all objects are available. Rabbit (Amur *et al.*, 2010) uses an equal-work data layout, which stores replicas in nodes in a fixing order. Nodes are powered on by creating an expansion chain, and the number of blocks stored on a node is inversely related to the index of the node in the expansion chain. Thus, the performance and the number of nodes that can be powered down would be scaled up at fine granularity. Lightning and GreenHDFS (Kaushik and Bhandarkar, 2010; Kaushik *et al.*, 2010) focus on data classification and separate a cluster into hot and cold zones. A number of servers in the cold zone would be of very low utilization and generate significant periods of idleness. These servers can be powered down to save energy.

2.3 Reducing power consumption using renewable energy sources

Recently, many researchers are interested in integrating emerging renewable energy solutions into computer systems for saving electric power. GreenHadoop (Goiri *et al.*, 2012) and GreenSlot (Goiri *et al.*, 2011) are MapReduce frameworks that predict the amount of solar energy that will be available in the near future and schedule the MapReduce jobs to maximize the green energy consumption and minimize brown energy consumption. iSwitch (Li *et al.*, 2012) explores the design tradeoffs between renewable energy utilization and performance in renewable-energy-driven computer systems by migrating the load between the renewable energy source and the electric power source. Also, iSwitch characterizes wind power behavior to minimize the power tuning overhead and performance degradation. It can mitigate operational overhead while maintaining the desired performance.

3 GreenCHT overview

The key challenges to achieving a high level of energy conservation for consistent hashing based key value storage systems are (1) ensuring data availability and (2) maintaining reliability as well as failure tolerance of the data. This section first details how our design leverages key ideas from the replication scheme under consistent hashing to provide power proportionality without compromising availability, and then describes the log store we use to maintain reliability and failure tolerance of the data. Additionally, we present how PMS predicts the workload and decides the power mode to provide power proportionality in a storage cluster. Finally, we describe the strategies that GreenCHT uses to handle new server failures resulting from power-mode switching.

3.1 Traditional replication under consistent hashing

Modern key value storage systems such as Dynamo (DeCandia *et al.*, 2007), Chord (Stoica *et al.*, 2001), Cassandra (Lakshman and Malik, 2010), Voldemort (LinkedIn, 2009), and Sheepdog (NTT Group, 2011) often use consistent hash tables to

assign objects to nodes. Consistent hashing maps objects to nodes by first hashing both the nodes and their addresses to the same ring. Then, each object is assigned to the first successor node whose value is equal to or follows its own hash value on the ring. As shown in Fig. 1a, object x is mapped to node A because it is the first node encountered in a clockwise walk (see the solid arrow).

To achieve high availability and durability, modern key value storage systems replicate data on R distinct nodes, where R is a replica-level parameter. As shown in the figure, object x is replicated on three clockwise successor nodes (A, B, and C) in the ring. To achieve load balance, consistent hashing assigns multiple virtual nodes to each physical node. For example, Fig. 1b shows multiple virtual nodes of each physical node. Each object is replicated at R distinct nodes by skipping successor positions to ensure that each replica of the object is stored at a distinct physical node. Thus, object x is replicated on nodes A, B, and D. Since the virtual nodes are permuted in a pseudo-random order by the consistent hashing function, the replicas of each object are randomly distributed around the ring.

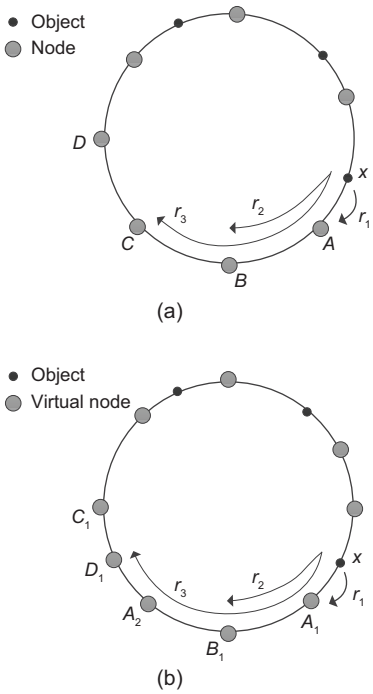

Fig. 1 Traditional replication under consistent hashing without (a) or with (b) virtual nodes

3.2 Multi-tier replication under consistent hashing (M-CHT)

3.2.1 Tiering

To achieve power proportionality under consistent hashing for key value storage systems, M-CHT first assigns nodes to nonoverlapping tiers. Each tier contains only one replica of each object. Thus, a multi-tier layout allows $(R-t)N/R$ nodes to be powered down while keeping t replicas of each object available, where N is the total number of nodes and R is the replica level. By powering down different numbers of tiers, key value storage systems can switch to different power modes to sustain different workload levels.

Fig. 2 shows an example of three power modes and three tiers with a replication level of 3. Each tier contains a subset of servers. Within each tier, the replicas are unique, meaning that the replicas of each object are not stored in the same tier. When the system switches to power mode 1, the first two tiers (tier 0 and tier 1) are powered down. In the lowest power mode (power mode 1), as shown in Fig. 2, two tiers are powered down while there is still one tier (tier 2) available. Apparently, the lowest power mode provides the highest energy savings. When the system switches to power mode 2 from power mode 1, the servers in tier 1 are powered up. In this case, power mode 3 is the highest power consumption mode with three active tiers.

There are many methods to group nodes into tiers. All these methods should satisfy two conditions: first, the number of tiers must be equal to the replica level, which ensures that the replicas of each object are not stored in the same tier, and second, the number of nodes in each tier must not vary greatly, since a large variance among the number of nodes in each tier will cause workload imbalance.

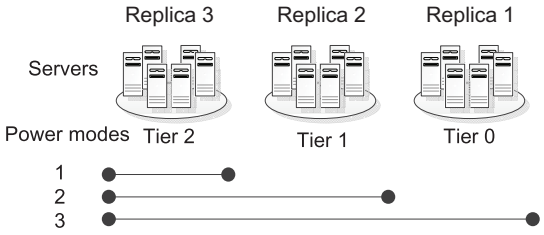

Fig. 2 Tiering and power modes with a replication factor of 3

3.2.2 Allocation of replicas

The replicas of each object are assigned to the first successors in different tiers to ensure that each tier contains only one replica of each object. Fig. 3 shows an example of multi-tier replication under consistent hashing. M-CHT assigns replicas of each object to its first successor in different tiers on the ring. As shown, the first replica of object x, r_1, is associated with virtual node A_1, which is its first successor in tier 0, and mapped to node A. The second replica r_2 is associated with its first successor in tier 1, virtual node B_1. In addition, the third replica is associated with its first successor in tier 2, virtual node C_1, and mapped to node C.

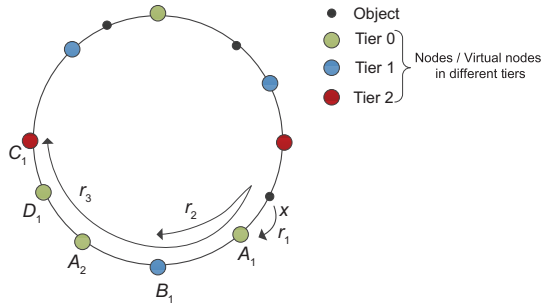

Fig. 3 Multi-tier consistent hash ring and replica layout (M-CHT)

In this case, each tier contains only one replica of each object. M-CHT not only provides power proportionality without disrupting data availability but also maintains load balance, fault tolerance, and scalability properties since it is based on consistent hashing.

Table 1 shows the allocation of replicas of object x to the three tiers. x has three replicas: r_1, r_2, and r_3. We use the term 'successor i' to refer to the ith distinct successor in a certain tier. r_1 is assigned to x's first successor (i.e., successor 1) in tier 0. r_2 and r_3 are assigned to the first successors in tier 1.

Table 1 Allocation of replicas

Object x	Successor 1
Tier 0	r_1
Tier 1	r_2
Tier 2	r_3

3.3 Distributed log store

The unpredicted failure of servers and network components in large-scale distributed storage systems makes reliability vital, especially during the low-power mode when a certain number of replicas are unavailable. A reliable distributed log store is used to temporarily store the writes to standby replicas. The log store ensures the reliability of the system in a way that it keeps the level of redundancy. All of the writes to standby replicas are distributed/off-loaded to the log regions in the active nodes. We refer to these kinds of writes as log writes and the data created by log writes as log replicas.

We use a small space called the 'log region' in each node to store log writes. The log region of each node holds the log writes that are redistributed to this node. Each node has two regions: an object region and a log region. The object region stores the replicas that are assigned to this node, while the log region holds the log replicas created by log writes.

3.3.1 Allocation of log replicas

To ensure the parallelism of writes (i.e., writes to replicas and log writes) and replica availability, we assign log writes to the active nodes located in the higher tiers. Table 2 shows the allocation of the log replicas of each object to different tiers. We first consider the allocation of log replicas with $R=3$. Each object x has three replicas, r_1, r_2, and r_3, which are assigned to x's first successors in three different tiers. r_1 has two log replicas (log r_1's), which are stored in x's two successors located in two higher tiers, tier 1 and tier 2, respectively. When tier 0 is powered down, the writes to r_1 are forwarded to log r_1, which is stored in x's second successor (successor 2) in tier 1. When both tier 0 and tier 1 are powered down, these writes are forwarded to log r_1, which is stored in x's second successor located in tier 2. r_2 has a single log replica, log r_2, which exists in x's third successor (successor 3) located in tier 2. When both tier 0 and tier 1 are powered down, the writes to r_2 are forwarded to log r_2 and stored in x's third successor located in tier 2.

We then consider the allocation of log replicas with the replica level R. Object x has R replicas, r_1, r_2, ..., r_R. x's first successor in tier $R-1$ stores r_R. x's remaining $R-1$ successors in tier $R-1$ store the log replicas of x (Table 2). When the first $R-1$

Table 2 Allocation of log replicas

Object x	Successor 1	Successor 2	Successor 3	...	Successor R
Tier 0	r_1	–	–	–	–
Tier 1	r_2	Log r_1	–	–	–
Tier 2	r_3	Log r_1	Log r_2	–	–
...	...	...	...	...	–
Tier $R-1$	r_R	Log r_1	Log r_2	...	Log r_{R-1}

tiers are powered down, the writes to the first $R-1$ replicas of x are forwarded to these $R-1$ successors located in tier $R-1$. Since these successors are different, the log store can ensure that the writes to the replicas or log replicas of each object can be performed in parallel.

The reads can be forwarded to available replicas when some replicas are powered down. When the system switches from a low-power mode to a high-power one, a certain number of log writes is reclaimed.

3.3.2 Log region data structure

To improve the random access performance of the log store, the log region that stores the log replicas in each node uses a logical data structure to provide fast lookup, insert, and delete operations (Fig. 4a). The log region stores its log replicas to different directories. Each log directory consists of multiple log files. First, to map a log replica to a log file within a directory, a hash function is applied to the log replica identifier. The n low-order bits of the hash value are used as the file index (i.e., file name), while the remaining bits of the hash value are used as the directory identifier (i.e., directory name). Second, a hash table is used to compute a directory identifier into an array of buckets, from which the desired directory can be found. The elements of each array are directory metadata objects. Within each directory, the file metadata objects are organized into a red-black tree (Park and Park, 2001) structure to provide low-cost lookup, insert, and delete operations, plus efficient sequential and random access. Each metadata object contains the version information to avoid replica divergence during data reclaims.

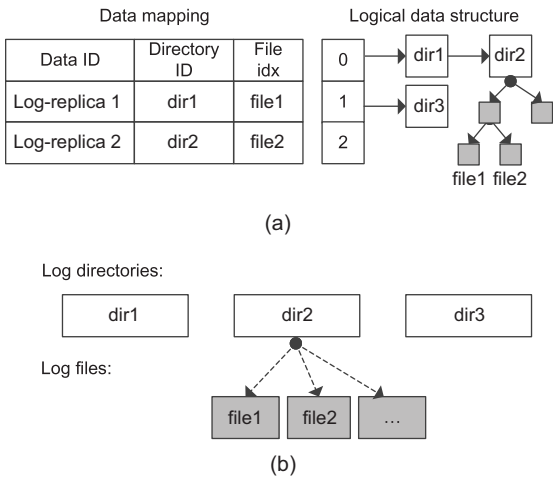

Fig. 4 Logical data structure (a) and physical structure (b) of the log store

Fig. 4b shows an example of a log region with three log directories, dir1, dir2, and dir3. dir2 contains file1, file2, and so on. The left part of Fig. 4a shows the data mapping of log files in a certain directory. Log file 1 is mapped to file 1 within directory dir1. The right part of Fig. 4a shows the directory hash table. Within dir2, the log files are organized into the red-black tree.

3.4 Predictive power-mode scheduler

In this section, we describe how PMS predicts the workload and decides the power mode for the next epoch. Since a server takes several seconds to come out of standby (Thereska *et al.*, 2011), GreenCHT switches power modes on a timescale of hours. PMS running on the manager aggregates load measurements from the local server and every other remote server periodically, and computes the power mode for the next hour based on the system load (detailed in Section 4.2).

3.4.1 Calculating the power mode

To choose the power mode with enough active servers to sustain a given load, we define three metrics: node metric, tier metric, and load metric. The node metric and tier metric (L_{tier}) are measured in MB/s by using the random-access I/O stream. The load metric is defined by using the aggregated reads and writes over all of the servers. Note that

the write rate is weighted by a factor of R since writes are replicated R times (Thereska *et al.*, 2011). Thus, the load can be computed in units of servers. We choose to compute the load in units of tiers rather than servers since PMS switches in units of tiers.

To track the load, the load is measured in 1-s intervals at each server and aggregated for each hour. Then, the scheduler predicts the load L_{predict} for the next hour by using a predictor based on an existing ARMAX model (Brockwell and Davis, 1991). To compute the power mode P in the next hour, we determine whether the predicted load for the next hour exceeds P tier loads. Then, we power down the remaining tiers, meaning that we choose the power mode P that contains P tiers to sustain the load:

$$P = \left\lceil \frac{L_{\text{predict}}}{L_{\text{tier}}} \right\rceil, \tag{1}$$

where L_{tier} denotes the tier metric, which is the performance of a single tier of servers. P denotes the power mode we choose, which is proportional to the system load. In this case, PMS co-operating with the multi-tier replication strategy provides power proportionality, since the power consumed in the GreenCHT system is proportional to system load.

3.4.2 Predicting load based on the ARMAX model

We establish an auto-regressive moving average with the exogenous output (ARMAX(p, q)) model (Box and Jenkins, 1990) to predict system load based on the observation that the measurements of the system load are discrete and equispaced at intervals of hours. The prediction algorithm can be described as follows: First, we use the Box-Jenkins principle (Brockwell and Davis, 1991) to obtain the correct model that fits the measured dataset. Second, we use the Akaike information criterion (AIC) to estimate the parameters of the ARMAX model and then obtain the two parameters p and q (Box and Jenkins, 1990). Since our dataset is limited to a week of data, we have the maximum values for both p and q (e.g., 12 and 12). We then deal with all possible subsets of these lags and calculate the AIC for each of them. Third, we choose the ARMAX model with the smallest AIC as our prediction model.

3.5 Metadata

To store the metadata concerning virtual nodes and node power states, GreenCHT maintains two lists, a node list and a virtual node list (Fig. 5). As shown, the node list consists of a number of node buckets. GreenCHT generates a unique identifier (64-bit) for each node when the node joins the storage cluster by using a consistent hash function. Each node bucket maintains the virtual nodes for which this node is responsible, using virtual node pointers (Fig. 5). Additionally, each node bucket contains its tier number, power state, and load. Fig. 5 summarizes the fields included in the node buckets. The metadata of each virtual node includes its identifier, tier number, and power state. Note that, if a node is active, the power state of its virtual nodes is active.

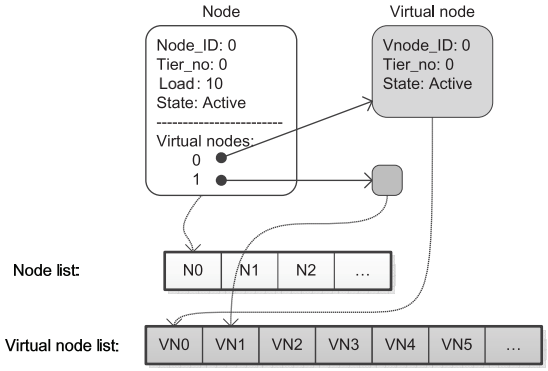

Fig. 5 Metadata concerning the multi-tier consistent hash table (M-CHT)

Typically, most key value storage systems (DeCandia *et al.*, 2007; LinkedIn, 2009; Lakshman and Malik, 2010; NTT Group, 2011) allow multiple virtual nodes per node. We use the term 'num vnodes' to refer to the number of virtual nodes randomly assigned to a node on the ring. num vnodes is an important factor of the metadata size as shown in Fig. 5. Apparently, the more the virtual nodes, the larger the proportion of the dataset that the node stores. Assuming that all nodes have equal capacity, the recommended or default value for num vnodes in Sheepdog is 64, which is sufficient to ensure good load balance for a small to median storage cluster. Usually, the number of servers in an Ethernet-based storage cluster is small (say, less than 1000 servers). Hence, we use the default of 64 virtual nodes empirically in our implementation. In our experiments, the

metadata size is approximately 50 KB.

3.6 Failure tolerance and recovery

3.6.1 Failures in low-power mode

Replica unavailability occurs when a node fails in the lowest power mode. For example, when server n fails while the servers that store its object replicas are powered down, these objects are unavailable. GreenCHT will switch to a higher power mode when replicas are unavailable. For example, if a server in tier 2 fails in power mode 1, the GreenCHT system will switch to power mode 2 by powering up the servers in tier 1.

However, since powering up a server takes several seconds or minutes, there will be a latency for requests accessing the temporarily unavailable objects during the power transition period (Thereska *et al.*, 2011). We define two policies: a 'high power savings' policy ($P_{\min} = 1$) where the system is required to keep at least one active replica for each object, and a 'high reliability' policy ($P_{\min} \geq 2$) where at least two replicas of each object are kept active to provide high reliability and availability. While the system with a high power saving policy will achieve significant energy conservation, the high reliability policy can ensure a better availability and reliability even in the lowest power mode. For example, in a distributed storage system with a replica level $R > 3$, if $P_{\min} \geq 2$, the temporarily unavailable replica will be avoided when a server fails in the lowest power mode $P_{\min}$.

3.6.2 Log file failure

When server n in a higher tier fails, the log files stored by server n will be unavailable since each log file has only a single replica. In this case, when a server m whose object is logged on server n is powered up, the replicas stored on server m will be updated from the active servers that store replicas with server m. We do not replicate the log file since it will eventually be reclaimed and deleted.

3.6.3 Replica recovery

Replica recovery begins when a server enters or leaves the system. Note that we do not treat server power-ups or power-downs in the same way as we treat a server entering or leaving the system. According to consistent hashing, when a server n joins or leaves the system, certain objects will be migrated between server n and its successor. The objects will be reassigned to server n's successor in the same tier as server n. For example, when server n in tier 0 leaves the system, the objects that were previously assigned to server n will be reassigned to server n's first successor in tier 0. When server n located in a higher tier leaves the system, the log files or log replicas also need to be migrated between server n and its successor. For example, when server n in tier 1 leaves the system, the objects and log replicas that were previously assigned to server n will be assigned to server n's first successor in tier 1.

4 GreenCHT implementation

GreenCHT was prototyped on Sheepdog (NTT Group, 2011), which was chosen for its open-source code and its consistent hashing based data distribution and replication mechanism.

4.1 Sheepdog

Sheepdog is a distributed object based storage that provides block-level storage volumes attached to QEMU/KVM virtual machines (NTT Group, 2011). Sheepdog's architecture provides an object storage. It assigns a 64-bit unique ID to each object. In Sheepdog's object storage, the location of an object is calculated based on a consistent hashing algorithm, and each object is replicated to multiple nodes to avoid data loss as mentioned in Section 3.1. Then, the object storage executes I/O request operations to the local disks. The Sheepdog system supports write to multiple replicas simultaneously, and the writes are sent to all the replicas of each object. While the read operation is randomly sent to one of the R replicas, and if a replica is successfully read, then read operation is considered successful.

4.2 GreenCHT prototype

We implement our data distribution and replication module, M-CHT, on Sheepdog by modifying its original data distribution and replication algorithm. In addition, PMS runs in the user space to schedule nodes to be powered down and powered up. As shown in Fig. 6, PMS includes a workload monitor and a predictor. The workload monitor captures

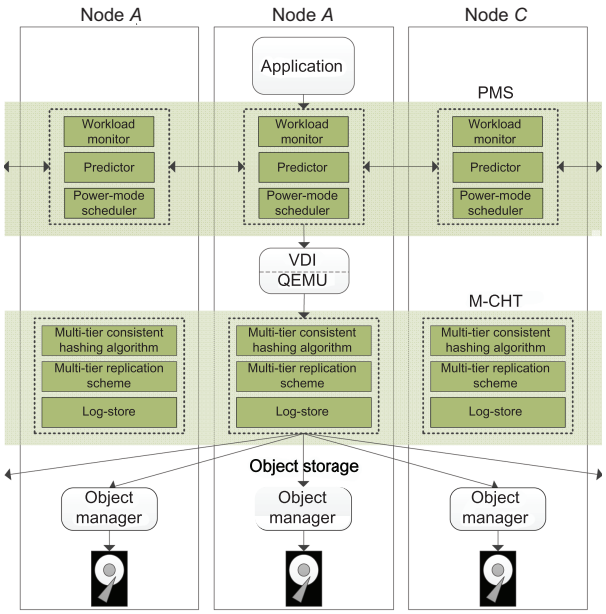

Fig. 6 GreenCHT system architecture

local load information (such as the number of I/O requests or bandwidth) within each hour. The predictor predicts the workload for the next hour.

As shown in Fig. 6, we use QEMU (NTT Group, 2011) to create virtual disk images (VDIs). Each I/O requests access to the virtual disk image, which corresponds to an object in an object-based storage device (OSD). As shown, the M-CHT component receives I/O requests and calculates the target nodes. In addition, the M-CHT component consists of a multi-tier consistent hashing algorithm, a multi-tier replication scheme, and a log store.

4.3 Power controlling in key value storage systems

GreenCHT is deployed in an Ethernet-based storage cluster. The GreenCHT system consists of multiple OSDs. A manager who is responsible for powering down or up a subset of tiers of nodes based on the global knowledge of the server load is selected from the OSDs. The metadata (M-CHT) is cached locally at each OSD. As shown in Fig. 7, all the OSDs (including the manager) consist of two major modules: M-CHT and PMS. As mentioned, the M-CHT module is used to distribute object replicas to OSDs, while PMS is responsible for deciding which OSD needs to be powered down or powered up in the next epoch.

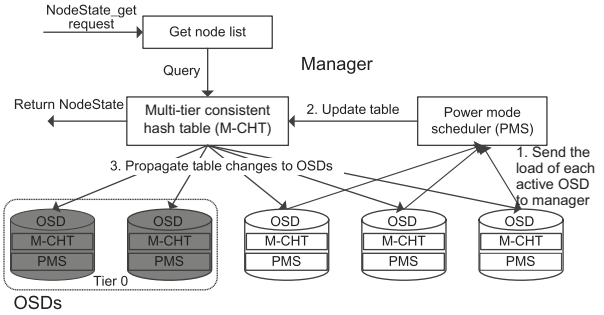

Fig. 7 Power control (the system switches from power mode 2 to power mode 3)

Fig. 7 shows an example of the power controlling process when a system switches from power mode 2 to power mode 3 in the GreenCHT system. As shown, the OSDs in tier 0 are powered down in power mode 2. First, each active OSD sends its load measurements periodically to the PMS running on the manager. Second, the PMS running on the manager predicts the workload and computes the power mode for the next epoch, namely power mode 3. Before powering up the OSDs in tier 0, the PMS first updates the power state of these OSDs in M-CHT. Then, the manager propagates the table modifications (or power mode schedules) to all active OSDs. Eventually, all active OSDs will update their metadata (M-CHT).

5 Evaluation

We evaluated GreenCHT with 12 real enterprise data center workloads by using trace-driven evaluation. The 12 traces used in our evaluation were taken at the block level and collected from Microsoft Cambridge servers (Microsoft Research Ltd., 2014). Each of the 12 workloads consisted of 7 d of data. Table 3 describes the parameters of the traces.

5.1 Experiment setup

All experiments were run on a 51-node Linux-based storage cluster. Each node was equipped with a 2.13 GHz Intel® Xeon® CPU (E5506) and 16 GB memory (Hynix HMT 151 R7AFP4C-H9 DDR3 REG 4 GB × 4). All nodes were installed with Fedora 14 with a Linux 2.6.35.6-45.fc14.x86 64-kernel and the Ext4 file system. Each node was equipped with one disk (Western Digital RE4 ITB), and each disk provided 1 TB storage.

In our experiments, we evaluated GreenCHT by

Table 3 Workload features

Trace	Average request size (KB)	Read/Write
usr	47.39	7.08
proj	17.44	5.56
hm	8.94	0.77
rsrch	8.35	0.19
prxy	12.01	1.57
src	38.21	3.02
stg	26.77	0.68
web	42.27	3.75
mds	36.57	1.41
prn	16.91	1.17
wdev	9.08	0.20
ts	9.12	0.21

comparing it with a baseline system based on Sheepdog (NTT Group, 2011), which is designed and deployed without any power management policy. In the baseline system, Sheepdog, a consistent hashing based data distribution policy, was used to distribute objects to OSDs (Section 3.1). As mentioned in Section 4, GreenCHT is implemented on the Sheepdog storage cluster by modifying Sheepdog's original data distribution and replication strategy. Hence, the unmodified Sheepdog system can be treated as a baseline system. Additionally, the consistent hash function used is the FNV-1 hash function (Cisco Systems, 2012) which is used by both Voldemort (LinkedIn, 2009) and Sheepdog (NTT Group, 2011). Initially, the Sheepdog storage cluster was formatted to use a default replication factor of 3.

During our experiments, we mapped the I/O requests to the system by using multiple virtual disk images. We used the QEMU block driver (NTT Group, 2011) to store the objects (4 MB) of the virtual disk image on the OSDs. Each I/O request accessed the virtual disk image, corresponding to a unique 64-bit object ID in OSDs. The virtual disk image object was then stored in OSDs as a data object corresponding to logical extents within the disk. We also used the QEMU block driver to export the virtual disk image: the virtual disk image was connected to a network block device (NBD). We used Blktrace (Open Source and Linux Organization, 2007) to replay the I/O requests to the NBD device. In addition, we used the Wake-on-LAN technique to power down or up the servers.

We first measured the power savings of GreenCHT and Sheepdog (Section 5.2). Second, we evaluated the performance of the two schemes (Section 5.3). We then evaluated the performance and power savings under various reliability requirements, and the performance of log store as well as the recovery rate (Section 5.4).

5.2 Power savings

In this section, we first show the accuracy of the predictor used by PMS. Second, we show how the power modes are scheduled based on PMS's decisions by replaying trace usr. Third, the power savings of GreenCHT and sheepdog are measured.

5.2.1 Load prediction

Fig. 8 shows the accuracy of our load predictor from the 12 workloads used. We applied a time-series analysis model, namely the ARAMX model, to predict workload. We see that during approximately 90% of the time the prediction error was less than 1, and during 70% of the time it was less than 0.3. The maximum prediction accuracy was prxy. Ninety percent of the errors of prxy were less than 0.15. The minimum prediction accuracy was wdev. Seventy percent of the errors of wdev were less than 0.3. Fig. 9 shows the power mode schedule. We observed that during 90% of the time the power modes matched the workloads. So, the predictor is correct most of the time.

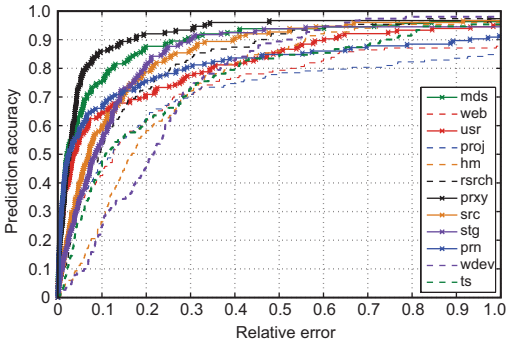

Fig. 8 The accuracy of our load predictor from the 12 workloads used

5.2.2 Power mode scheduler

Fig. 9 shows the load metric for each hour and the number of active tiers (i.e., power mode) chosen by PMS for each hour of replaying usr. The load metric used was the number of servers needed to support a workload for each hour. Power mode was defined as the number of tiers chosen to support the workload for each hour. We made two observations. First,

during 90% of the time the power modes were correct. The number of active tiers chosen matched the workload for each hour. Second, the results showed that 51 servers were sufficient for meeting the performance requirement. Only a few hours of load were overhead. For example, the workload during the period of 60th–64th hour cannot be sustained even with all tiers powered up.

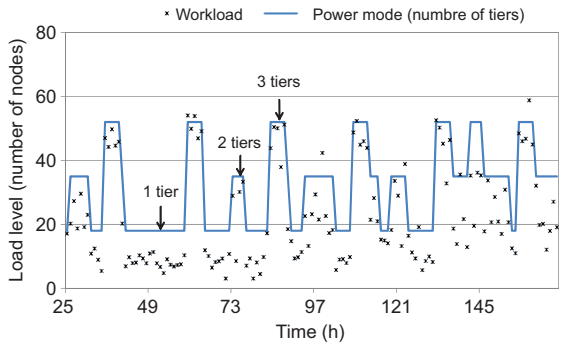

Fig. 9 Power mode schedule for the GreenCHT system: high-power mode with three active tiers, median-power mode with two active tiers, and low-power mode with one active tier

5.2.3 Power savings

Fig. 10 shows the time spent on different power modes as a percentage of the whole replaying time using the power scheduler's decisions by replaying 12 traces. Variance was measured across traces. For the high-power mode (with three active tiers), the maximum time was 64% (prxy) and the minimum was 2% (wdev). For the low-power mode (with one active tier), the maximum time was 53% (wdev) and the minimum was 14% (stg). For the median-power mode, the maximum time was 46% (proj) and the minimum was 14% (prxy).

The nodes used in our experiments were homogeneous, and the total power consumption of the storage cluster was nearly proportional to the number of active nodes. Thus, the power consumption is computed based on the number of active nodes that are switched on:

$$E = \frac{T_{p1}N_{p1} + T_{p2}N_{p2} + \ldots + T_{pn}N_{pn}}{T_{p1} + T_{p2} + \ldots + T_{pn}}, \quad (2)$$

where E denotes the average number of active nodes that are switched on during the whole trace replay time. We use E to represent the power consumption of the whole cluster. T_{pi} denotes the time spent on power mode i, while N_{pi} denotes the number of active nodes in power mode i. Fig. 11 shows the power consumptions of GreenCHT and Sheepdog. As shown, GreenCHT consumes much lower power than Sheepdog. Sheepdog's power consumptions for different workloads are the same, and the average number of active nodes that are switched on is 51. This is because Sheepdog does not use a power management strategy.

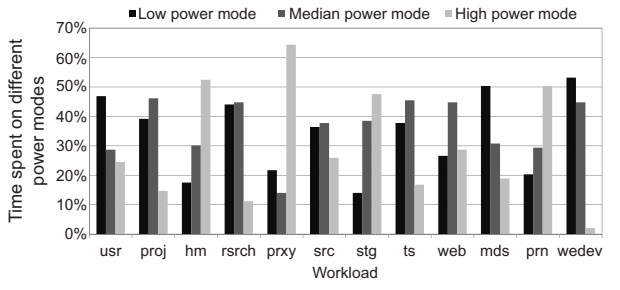

Fig. 10 Time spent on different power modes for GreenCHT

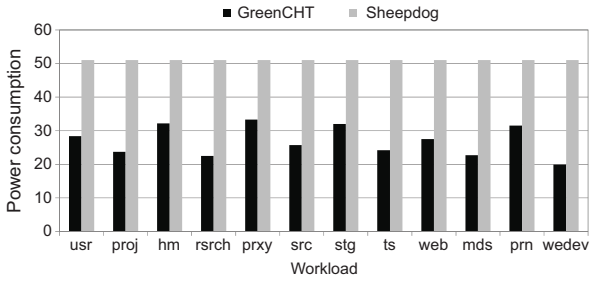

Fig. 11 Power consumptions for GreenCHT and Sheepdog (the y-axis parameter is the average number of active nodes that are switched on during the whole trace replay time)

Fig. 12 shows the power savings of GreenCHT under 12 traces. The power savings are computed over Sheepdog, which is the baseline system. As shown, GreenCHT provides significant power savings. For example, for prxy, GreenCHT can save 35% energy over the baseline. This is because GreenCHT uses multi-tier replication, which provides power proportionality, while the Sheepdog system does not use any power management strategy. Moreover, compared with Fig. 10, the more time GreenCHT spends on the lower power mode, the more power savings we would achieve. The maximum power savings was 61% (wedev) and the minimum was 35% (prxy).

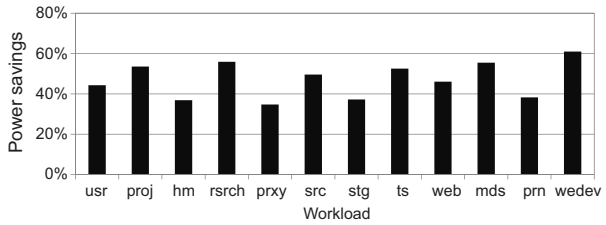

Fig. 12 Power savings for GreenCHT by using power mode scheduler's decisions

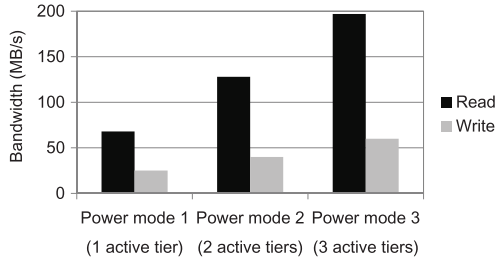

Fig. 13 Performance under different power modes (with different numbers of active tiers)

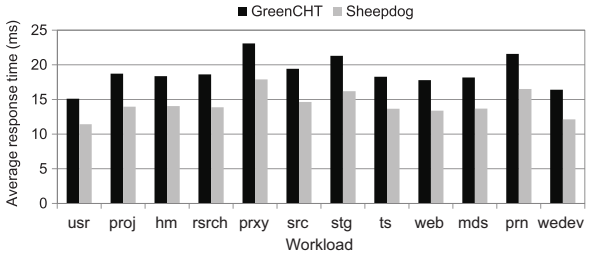

Fig. 14 Average response time for the two schemes under 12 traces

5.3 Performance

We first measured GreenCHT system's read/write performance by benchmarking the system under different power modes. Then, we evaluate the performance of the three schemes by using real workloads.

5.3.1 Performance under different power modes

We evaluated the performance of our system when clients were accessing objects under different power modes (with different numbers of active tiers) by using SysBench (MySQL, 2004). Seven clients made random-access read and write requests to the system. Fig. 13 shows the aggregated active server performance. Several observations can be made. First, in all cases, reads are faster than writes (by a factor of about 3) because the replica level is 3 and the bandwidth of writes is reduced to one-third. Second, ideally, the performance of n tiers is n times that of a single tier for reads, and one-third that for writes. The actual performance of n tiers in those cases is close to ideal for reads as shown in Fig. 13. Third, the highest client performance observed was about 35 MB/s in power mode 3 when the three tiers were all active for reads and about 10 MB/s for writes. We also measured single-client streaming random read and write bandwidth from a single server with three active tiers using SysBench. The read performance was 104 MB/s while the write performance was about 17 MB/s. We obtained only about a third of the expected bandwidth of single-client streaming for reads and half the bandwidth for writes because the latency of the disk seek time increases with concurrent random-access streams.

5.3.2 Latency

Fig. 14 shows the average response time of the GreenCHT system and Sheepdog. First, the performance penalty for GreenCHT was small. The average response time of GreenCHT increased only by 4–5 ms due to background data reclaim traffic. Second, compared with GreenCHT, Sheepdog, which has no power management policy, acted like a baseline system. It shows the lower bound for the average response time and upper bound for energy consumption.

5.4 Reliability

5.4.1 Different reliability requirements

In this section, we show the reliability of log store by evaluating the performance and power savings under different reliability requirements of GreenCHT. Fig. 15a shows the results for two policies: a 'high power savings' policy ($P_{\min} = 1$) where the system is required to keep at least one active replica for each object, and a 'high reliability' policy ($P_{\min} = 2$) where two replicas of each object are kept active to provide high reliability. We see that the power savings for the high power savings policy were significantly higher than those for the high reliability policy. However, the high reliability policy's power savings were still significant. Fig. 15b shows the performance under different policies. We see that the performance for the high reliability policy was slightly better than that for the high savings policy. This is mainly because the high reliability policy powers down fewer servers than the high power savings policy and has less background reclaim traffic.

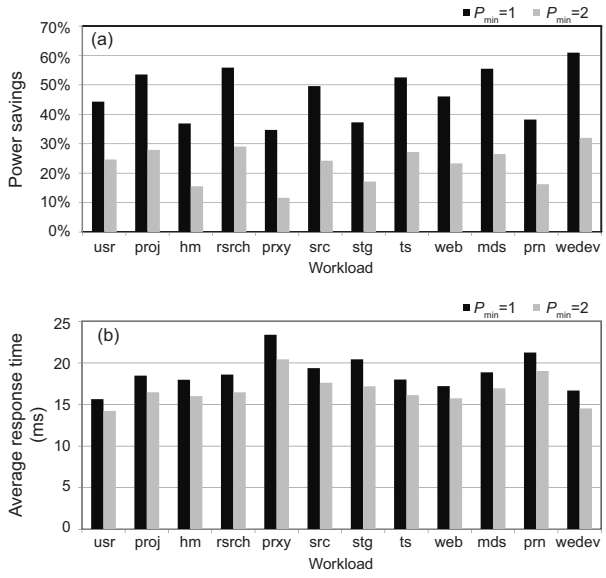

Fig. 15 Power savings (a) and average response time (b) for two different reliability requirements, i.e., high power savings policy ($P_{\min}=1$) and high availability policy ($P_{\min}=2$)

5.4.2 Log store performance

Table 4 shows the measurements from the log store. The average reclaim rate achieved was 13 MB/s for each server. Note that the reclaim process should not interfere with the foreground workload. In previous experiments, we observed that the interference of data reclaims led to slight performance degradation. The average reclaim time under the 12 traces was 20 s and the minimum time was only 8 s.

Table 4 Data reclaim statistics

Parameter	Value
Average reclaim rate for each reclaim (MB/s)	13
Minimum reclaim time (s)	8
Average reclaim time (s)	20
Maximum reclaim time (s)	28

5.4.3 Failure recovery

This experiment shows the recovery rate when we killed a server holding about 4 GB data under different numbers of active tiers. We varied the number of tiers on which the failed node can rebuild. Fig. 16 shows that the recovery rate increased linearly with the increase of the number of tiers. The recovery rate from the failed server was about 51 MB/s when there was only one tier of servers on which the failed server can rebuild. The recovery rate increased with the increase the number of tiers. When there were three tiers of servers on which the failed server can rebuild, the recovery rate achieved from the failed server was about 200 MB/s.

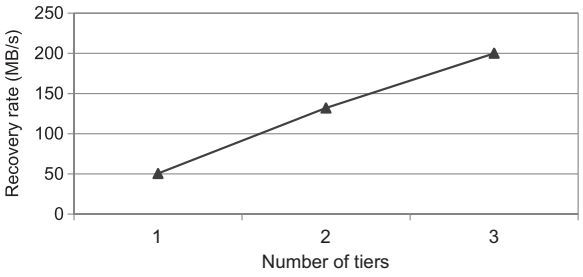

Fig. 16 Recovery rate for different numbers of active tiers

6 Conclusions

In summary, we proposed a reliable power management scheme for consistent hashing based distributed key value storage systems, called GreenCHT. Compared to the widely used traditional replication scheme, GreenCHT can provide significant power savings while maintaining the reliability of the whole system. GreenCHT uses multi-tier replication to achieve considerable power savings while ensuring data availability and a reliable distributed log store to ensure reliability as well as fault tolerance of the whole system. To provide power proportionality and ensure good performance, GreenCHT dynamically adapts to the workload variation under the required performance constraint by using a predictive power mode scheduler (PMS).

GreenCHT was implemented based on the Sheepdog storage cluster by modifying Sheepdog's original data distribution and replication algorithm. We have evaluated the performace of GreenCHT by comparing it with the unmodified Sheepdog system. By replaying 12 real-world traces, experimental results showed that GreenCHT could save significant energy while maintaining a desired performance.

References

Amur, H., Cipar, J., Gupta, V., *et al.*, 2010. Robust and flexible power-proportional storage. Proc. 1st ACM Symp. on Cloud Computing, p.217-228. http://dx.doi.org/10.1145/1807128.1807164

Bhagwan, R., Savage, S., Voelker, G.M., 2003. Replication strategies for highly available peer-to-peer storage. *In*: Future Directions in Distributed Computing. Springer-Verlag, p.153-158.

Box, G.E.P., Jenkins, G., 1990. Time series analysis forecast-

ing and control. *In*: Wiley Series in Probability and Statistics. Holden-Day, Inc.

Brockwell, P.J., Davis, R.A., 1991. Time series: theory and methods. *In*: Springer Series in Statistics. Springer-Verlag, New York, NY, USA.
http://dx.doi.org/10.1007/978-1-4419-0320-4

Cisco Systems, 2012. FNV-1. Available from http://www.isthe.com/chongo/tech/comp/fnv/index.html.

Colarelli, D., Grunwald, D., 2002. Massive arrays of idle disks for storage archives. Proc. ACM/IEEE Conf. on Supercomputing, p.1-11.

DeCandia, G., Hastorun, D., Jampani, M., *et al.*, 2007. Dynamo: Amazon's highly available key-value store. Proc. ACM SIGOPS Symp. on Operating Systems Principles, p.205-220.
http://dx.doi.org/10.1145/1294261.1294281

Goiri, I., Le, K., Haque, M.E., *et al.*, 2011. Greenslot: scheduling energy consumption in green datacenters. Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis, p.1-11.
http://dx.doi.org/10.1145/2063384.2063411

Goiri, I., Le, K., Nguyen, T.D., *et al.*, 2012. GreenHadoop: leveraging green energy in data-processing frameworks. Proc. 7th ACM European Conf. on Computer Systems, p.57-70.
http://dx.doi.org/10.1145/2168836.2168843

Gorini, S., Quirini, M., Menciassi, A., *et al.*, 2007. PARAID: a Gear-Shifting Power-Aware Raid.

Harnik, D., Naor, D., Segall, I., 2009. Low power mode in cloud storage systems. Proc. Int. Symp. on Parallel and Distributed Processing Systems, p.1-8.

Karger, D., Lehman, E., Leighton, T., *et al.*, 1997. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. Proc. 29th Annual ACM Symp. on Theory of Computing, p.654-663.
http://dx.doi.org/10.1145/258533.258660

Kaushik, R.T., Bhandarkar, M., 2010. GreenHDFS: towards an energy-conserving, storage-efficient, hybrid Hadoop compute cluster. Proc. Int. Conf. on Power Aware Computing and Systems, p.1-9.

Kaushik, R., Cherkasova, L., Campbell, R., *et al.*, 2010. Lightning: self-adaptive, energy-conserving, multi-zoned, commodity green cloud storage system. Proc. 19th ACM Int. Symp. on High Performance Distributed Computing, p.332-335.
http://dx.doi.org/10.1145/1851476.1851523

Lakshman, A., Malik, P., 2010. Cassandra—a decentralized structured storage system. *ACM SIGOPS Oper. Syst. Rev.*, **44**(2):35-40.
http://dx.doi.org/10.1145/1773912.1773922

Li, C., Qouneh, A., Li, T., 2012. iSwitch: coordinating and optimizing renewable energy powered node clusters. Proc. 39th Annual Int. Symp. on Computer Architecture, p.512-523.
http://dx.doi.org/10.1145/2366231.2337218

LinkedIn, 2009. Voldemort Project. Available from http://www.project-voldemort.com/voldemort/.

Microsoft Research Ltd., 2014. MRS Cambridge Traces. Available from http://project-voldemort.com/.

MySQL, 2004. SysBench. Available from http://sysbench.sourceforge.net/.

Narayanan, D., Donnelly, A., Rowstron, A., 2008. Write offloading: practical power management for enterprise storage. *ACM Trans. Stor.*, **4**(3):1-10.
http://dx.doi.org/10.1145/1416944.1416949

NTT Group, 2011. Sheepdog. Available from https://github.com/sheepdog/sheepdog/wiki.

Open Source and Linux Organization, 2007. Blktrace User Guide. Hewlett-Packard Company. Available from http://www.cse.unsw.edu.au/aaronc/iosched/doc/blktrace.html.

Park, H., Park, K., 2001. Parallel algorithms for red-black trees. *Theor. Comput. Sci.*, **262**(1-2):415-435.
http://dx.doi.org/10.1016/S0304-3975(00)00287-5

Pinheiro, E., Bianchini, R., 2004. Energy conservation techniques for disk array-based servers. Proc. 18th Annual Int. Conf. on Supercomputing, p.68-78.
http://dx.doi.org/10.1145/1006209.1006220

Pinheiro, E., Bianchini, R., Dubnicki, C., 2006. Exploiting redundancy to conserve energy in storage systems. Proc. Joint Int. Conf. on Measurement and Modeling of Computer Systems, p.15-26.
http://dx.doi.org/10.1145/1140277.1140281

Stoica, I., Morris, R., Karger, D., *et al.*, 2001. Chord: a scalable peer-to-peer lookup service for Internet applications. Proc. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications, p.149-160.
http://dx.doi.org/10.1145/383059.383071

Thereska, E., Donnelly, A., Narayanan, D., 2011. Sierra: practical power-proportionality for data center storage. Proc. 6th Conf. on Computer Systems, p.169-182.
http://dx.doi.org/10.1145/1966445.1966461

Zhu, Q., Chen, Z., Tan, L., *et al.*, 2005. Hibernator: helping disk arrays sleep through the winter. 20th ACM Symp. on Operating Systems Principles, p.177-190.
http://dx.doi.org/10.1145/1095810.1095828