GreenCHT: A Power-Proportional Replication Scheme for Consistent Hashing based Key Value Storage Systems

Nannan Zhao[12*], Jiguang Wan[12†], Jun Wang[3‡], and Changsheng Xie[12§]
[1]Wuhan National Laboratory for Optoelectronics, Wuhan, China
[2]Department of Computer science and technology, Huazhong University of Science and Technology, Wuhan, China
[3]Department of Electrical Engineering and Computer Science, University of Central Florida, USA
*nnzhaocs@hotmail.com, †jgwan@mail.hust.edu.cn, ‡jwang@mail.ucf.edu, §cs_xie@mail.hust.edu.cn
Corresponding author: Jiguang Wan

***Abstract*—Distributed key value storage systems are widely used by many popular networking corporations. Nevertheless, server power consumption has become a growing concern for key value storage system designers since the power consumption of servers contributes substantially to a data center's power bills. In this paper, we propose GreenCHT, a power-proportional replication scheme for consistent hashing based key value storage systems. GreenCHT consists of a power-aware replication strategy — multi-tier replication strategy and a centralized power control service — predictive power-mode scheduler. The multi-tier replication provides power-proportionality and ensures data availability, reliability, consistency, as well as fault-tolerance of the whole system. The predictive power-mode scheduler component predicts workloads and exploits load fluctuation to schedule nodes to be powered-up and powered-down. GreenCHT is implemented based on *Sheepdog*, a distributed key value system that uses consistent hashing as an underlying distributed hash table. By replicating twelve real workload traces collected from Microsoft, the evaluation results show that GreenCHT can provide significant power savings while maintaining an acceptable performance. We observed that GreenCHT can reduce power consumption by up to 35%-61%.**

Keywords*—CHT; Replication; Power Management; Key Value Storage System; Sheepdog

I. Introduction

Server power consumption is an important issue for distributed storage systems because it contributes substantially to a data center's power bills [9]. Furthermore, as the size of data increases, massive amounts of storage are becoming more necessary [8]. Sierra [9] and Rabbit [7] have been proposed for power proportionality: the energy consumed should be proportional to the system load. However, we find that a surplus of power is being consumed by many storage systems, especially during idle periods of system utilization [9].

Literature concerning data-center energy management shows significant power saving potentials in distributed file systems [9] [1] [15]. Sierra [9] [15] shows that I/O workloads in data centers exhibit significant dynamism. Therefore, servers in a distributed system (e.g., Google FS) can be powered down to save power during idle periods of utilization. Sierra exploits the redundancy of replicas in distributed storage systems. It maintains a primary replica available for access, and powers down the servers hosting inactive replicas to provide power proportionally. GreenHDFS [1] focuses on data differentiation. It separates a Hadoop cluster into Hot and Cold zones based on the observation that the data stored in the cluster shows significant variations in the access patterns. The cold zones with low utilization can be powered down for energy savings.

Our focus is on key value storage systems. Key value storage systems have been growing more attractive over the past few decades [5], as they are being employed by many modern popular networking corporations, such as Dynamo at Amazon [6], Cassandra at Facebook [2], and Voldemort at LinkedIn [3]. Key value storage systems use distributed hash tables (DHT) to distribute data to storage nodes. A consistent hash table (CHT) is often used by modern key value storage systems [6] [13] [2] [3] as the underlying distributed hash table because consistent hashing can adapt to a changing set of nodes without creating a large overhead of data migration [11] [12].

We take a practical approach to provide power-proportionality for consistent hashing based key value storage systems. Instead of randomly placing replicas of each object on a number of nodes on the consistent hash ring, we use a power-aware multi-tier placement of replicas that assigns the replicas of objects on non-overlapping tiers of nodes in the ring. It is known that replication is widely used by distributed storage systems to provide high availability and durability. Typically, in an R-way replicated storage system, we can maintain a set of $\frac{1}{R}$ nodes that consist of only a single replica for each object, and power down up to $\frac{R-1}{R}$ of the nodes for power savings during periods of light loads [9]. In our power-aware multi-tier replication strategy, we can maintain a number tiers of nodes that will be made available and power down the remaining tiers of nodes without disrupting data availability by exploring the load fluctuation.

Although recent research and development has suggested that the most promising approach to saving power in enterprise data centers is to power down entire server rather than saving power in individual components [9], powering servers down in distributed key value storage systems is challenging since availability may be compromised. We make a tradeoff between power savings and availability. Our power-aware multi-tier replication ensures that at least one replica of each object is

978-1-4673-7619-8/15/$31.00

kept available even when a significant fraction of the servers are powered down.

Data consistency must be ensured, especially when a subset of the servers is powered down. All of the writes to the powered-down replicas are *offloaded* to the active servers to maintain replica consistency. We use a small storage space (called log-store) in each server to temporarily store offloaded data. When the replicas are powered up, offloaded writes are reclaimed.

This power-aware multi-tier replication strategy, coupled with log-store, not only ensures data availability and consistency but also maintains reliability as well as fault-tolerance of the whole system even when a subset of the servers is powered down. In the worst case, if a server fails while its peer servers that store its replicas are powered down, these objects are unavailable. A recovery process will be started to maintain data availability: The failed server's peer servers are powered up and the offloaded data is reclaimed.

Most modern distributed key value storage systems such as Dynamo [6], Cassandra [2], Voldemort [3], and Sheepdog [13] have a *decentralized structure*. In decentralized overlay networks, a group of nodes must cooperate with each other since there is no centralized coordinator. We implement a predictive power-mode scheduler to schedule nodes to be powered-down or powered-up. The predictive power mode scheduler functions as a "centralized power control service". It predicts I/O workloads and exploits load fluctuation to schedule nodes to be powered-up and powered-down accordingly. Moreover, it cooperates with the power-aware multi-tier replication strategy to provide power-proportionality for distributed key value storage systems. By powering down tiers of nodes, the system can switch to a wide range of power-modes to meet different performance, availability, reliability as well as power-saving requirements.

In this paper, we present the GreenCHT technique to provide power-proportionality for consistent hashing based key value storage systems. GreenCHT consists of a power-aware replication strategy — multi-tier replication strategy and a centralized power control service — predictive power mode scheduler. We have implemented GreenCHT on *Sheepdog* [13] as a power-proportional key value storage system. Specifically, we implemented our multi-tier replication scheme based on sheepdog, with modifications to the consistent hash table and replication strategy. The predictive power mode scheduler is implemented as an centralized power controller for a storage cluster.

The remainder of the paper is structured as follows. Section II presents an overview of GreenCHT. Section III presents the implementation of GreenCHT. Section IV presents experiment results. Section V discusses related work and Section VI concludes this paper.

II. GREENCHT OVERVIEW

A. Traditional Replication under Consistent Hashing

We first briefly describe the traditional replication under consistent hashing before we present our power-aware multi-tier replication. Modern key value storage systems such as Dynamo [6], Chord [11], Cassandra [2], Voldemort [3] and Sheepdog [13] 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 on 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 Figure 1(a), object x is mapped to node A because it is the first node encountered in a clockwise walk (see solid-arrow).

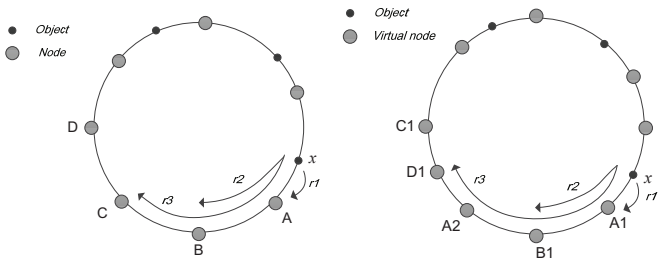
To achieve high availability and durability, modern key value storage systems replicate data on R distinct nodes, where R is a replica level parameter. As shown, object x is replicated on 3 clockwise successor nodes (A, B and C) in the ring. To achieve load balance, consistent hashing assigns multiple virtual nodes to each physical node. In this case, each node is responsible for multiple ranges distributed around the ring. For example, figure 1(b) 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 node 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. Thus, the traditional replication strategy prevents subsets of nodes from powering down without violating data availability [7].

B. Multi-tier Replication under Consistent Hashing

1) Tiering and Power-Modes: To achieve power-proportionality under consistent hashing for key value storage systems, we first assign both servers and replicas to non-overlapping tiers. Each tier contains only one replica of each object. Thus, a multi-tier layout allows $\frac{(R-t)N}{R}$ of the 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. When the system switches to power mode t, the first $R-t$ tiers are powered down. We can power down $R-t$ tiers ($t < R$) while keeping t tiers available, which means that there are still t replicas of each object available.

Figure 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 figure 2, two tiers are powered down while there are 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 mode with three active tiers. By switching to different power-modes, the system can meet different performance, availability, reliability as well as power-saving requirements.

2) Multi-tier Consistent Hash Table and Multi-tier Replication: In multi-tier replication, the replicas of each object are assigned to the first successors in different tiers to ensure that



(a) Without virtual nodes (b) With virtual nodes

Fig. 1. Traditional replication under consistent hashing

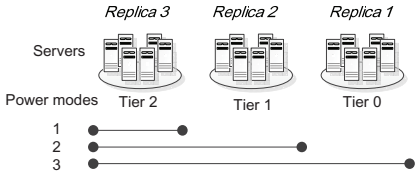


Fig. 2. Tiering and power modes with a replication factor of 3

each tier contains only one replica of each object. Figure 3 shows an example of multi-tier replication under consistent hashing. Instead of randomly placing replicas of each object on R distinct successors, GreenCHT assigns replicas of each object to its first successors in different tiers on the ring. As shown, the first replica of object x, $r1$, is associated with virtual node A1 which is its first successor in tier 0, and mapped to node A. The second replica $r2$ is associated with its first successor in tier 1: virtual node B1. In addition, the third replica is associated with its first successor in tier 2, virtual node C1, and mapped to node C. In this case, each tier contains only one replica of each object. This approach not only provides power-proportionality without disrupting data availability but also maintains load balance, fault tolerance, and scalability properties since multi-tier replication is based on consistent hashing.

Table I shows the allocation of replicas of object x to the three tiers. x has three replicas: $r1$, $r2$ and $r3$. We use the term "successor_i" to refer to the i^{th} **distinct** successor in a certain tier. $r1$ is assigned to x's first successor (i.e. successor_1) in tier 0. $r2$ and $r3$ are assigned to the first successors in tier 1 and tier 2 respectively.

C. Log-store and The Allocation of Log-replicas

Writes should be stored persistently and consistently, even when replicas are unavailable due to nodes being powered down. Therefore, we use a log-store to temporarily store *offloaded* data. All of the writes to standby (or inactive) replicas are *offloaded* to the log-store, which exists in active nodes to maintain replica consistency. We refer to these kind of writes as log writes. When the nodes containing replicas are powered up, log writes are reclaimed: logged data is read from log-store and written to the power-up nodes, and then deleted from the log-store.

We use a small space in each node to store log writes. The log-store of each node holds the log writes that are *offloaded* to this node. Each node has two regions: an object region and a log-store region. The object region stores the replicas that

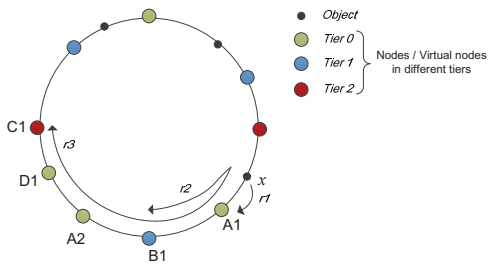


Fig. 3. Multi-tier consistent hash ring and Multi-tier replication

TABLE I. MULTI-TIER REPLICATION

Object x	**Successor_1**
Tier 0	$r1$
Tier 1	$r2$
Tier 2	$r3$

are assigned to this node while the log-store region holds the log-replicas created by log writes.

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 II shows the allocation of the log-replicas of each object to different tiers. We define the logged data created by log writes as log-$replicas$. We first consider the allocation of log-replicas with $R = 3$. Each object x has 3 replicas, $r1$, $r2$ and $r3$, which are assigned to x's first successors in three different tiers. $r1$ has two log-replicas (log-$r1$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 $r1$ are forward to log-$r1$, which stored in x's second successor (successor_2) in tier 1. When both tier 0 and tier 1 are powered down, these writes are forward to log-$r1$, which stored in x's second successor located in tier 2. $r2$, has a single log-replica, log-$r2$, 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 $r2$ are forwarded to log-$r2$ and stored in x's third successor located in tier 2.

We then consider the allocation of log-$replicas$ with replica level R. Object x has R replicas, $r1$, $r2$, $r3$, ... , and rR. x's first successor in tier $R-1$ stores rR. x's remaining $R-1$ successors in tier $R-1$ store the log-$replicas$ of x as shown in table II. When the first $R-1$ 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 to a high power mode, a certain amount of log writes are reclaimed.

TABLE II. LOG-REPLICAS ALLOCATION

Object x	**Successor_1**	**Successor_2**	**Successor_3**	...	**Successor_R**
Tier 0	$r1$	—	—	—	—
Tier 1	$r2$	log-$r1$	—	—	—
Tier 2	$r3$	log-$r1$	log-$r2$	—	—
...	...	...	...	...	—
Tier R-1	rR	log-$r1$	log-$r2$	...	log-$r(R-1)$

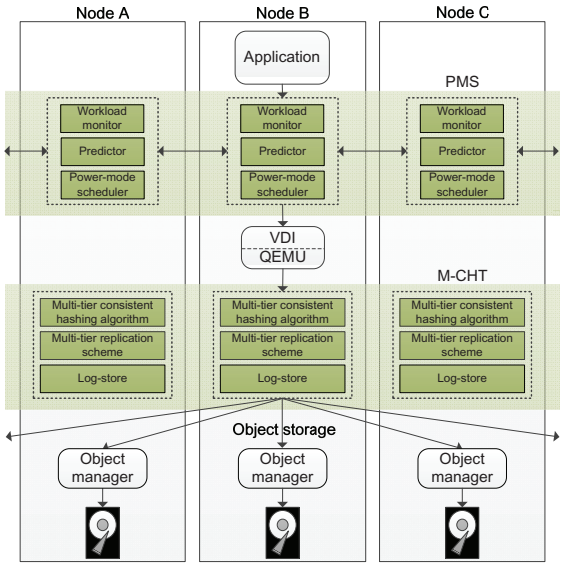


Fig. 4. GreenCHT system architecture

D. Predictive Power Mode Scheduler (PMS)

To choose the power mode with enough active servers to sustain a given load, we define three metrics, the 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 [9]. Thus, the load can be computed in units of servers. We choose to compute the load in units of tiers rather than servers since the power mode scheduler switches in units of tiers.

To track the load, the load is measured in 1 second 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 ARMAX model [17]. 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:

$$P = \lceil \frac{L_{predict}}{L_{tier}} \rceil \quad (1)$$

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

III. GREENCHT IMPLEMENTATION

A. GreenCHT Prototype

GreenCHT was prototyped on Sheepdog [13], which was chosen for its open source code and its consistent hashing based data distribution and replication mechanism. Sheepdog is a distributed object-based storage that provides block level storage volumes attached to QEMU/KVM virtual machines [10]. Sheepdog architecture is fully symmetric and provides an object storage. It assigned a 64-bit unique id to each object. In Sheepdog's object storage, the location of an object is calculated based on consistent hashing algorithm and each object is replicated to multiple nodes to avoid data loss as mentioned in Section II.A. Then, the object storage executes I/O request operations to the local disks.

We implemented our data distribution and replication module — M-CHT on Sheepdog by modifying its original data distribution and replication algorithm. In addition, the power mode scheduler (PMS) runs in the user space to schedule nodes to be powered-down and powered-up. As shown in figure 4, the power mode scheduler includes a workload monitor and a predictor. The workload monitor captures the local load information (such as the number of I/O requests or bandwidth) within each hour. The predictor predicts the workload for next hour as mentioned in Section II.D. As shown in figure 4, we use QEMU [13] to create virtual disk images (VDIs). Each I/O request accesses to the virtual disk image, which corresponds to an object in an OSD (Object-based Storage Device). 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.

B. Power Controlling in Key Value Storage Systems

GreenCHT is deployed in an Ethernet-based storage cluster. The GreenCHT system consists of multiple OSDs (Object-based Storage Devices). A manager is elected among the OSDs by using Zookeeper [4], which is responsible for powering down or up a subset of tiers of nodes. We also use Zookeeper as our cluster engine to manage cluster membership and failure detection. The metadata (M-CHT) is cached locally at each OSD. Currently, GreenCHT is implemented on an Ethernet-based storage cluster. Each OSD maintains the entire multi-tier consistent hash table (M-CHT) and cluster membership. Thus, the clients can connect to any OSD and their read/write requests can be routed to any OSD in the cluster with at most one-hop request routing. As shown in figure 5, all the OSDs (including manager) consists of two major modules: the M-CHT (multi-tier consistent hash table) and the PMS (power mode scheduler). As mentioned in Section II.B, the M-CHT module is used to distribute object replicas to OSDs while the PMS is responsible for deciding which OSD need to be powered down or powered up in the next epoch.

Figure 5 shows an example of power controlling process when a system switches from power mode 2 to power mode 3 in 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, which is responsible for powering down or up a subset of tiers of nodes based on the global knowledge of server load. Second, the PMS running on manager predicts the workload and computes the power mode for the next epoch: 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

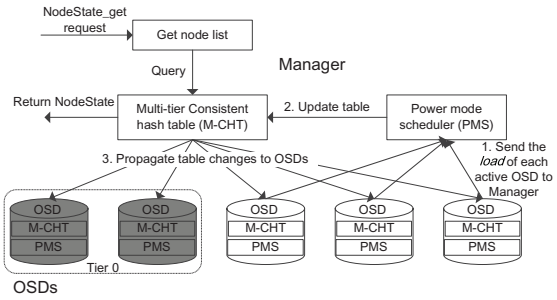


Fig. 5. Power controlling: The system switches from power mode 2 to power mode 3.

(or power mode schedules) to all active OSDs. Eventually, all active OSDs will update their metadata (M-CHT).

IV. EVALUATION

We evaluated GreenCHT with twelve real enterprise data center workloads by using trace-driven evaluation. The twelve traces we used in our evaluation were taken at block level and collected from Microsoft Cambridge servers [14].

In our experiments, we evaluate GreenCHT by comparing it with a *baseline* system based on sheepdog [13], which are designed and deployed without any power management policy. In the baseline system — sheepdog, a consistent hashing-based data distribution policy is used to distribute objects to OSDs, just as Section II.A describes. As mentioned in Section III, GreenCHT is implemented on 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 which presents the lower bound for average response time and upper bound for energy consumption. Additionally, the consistent hash function we used is FNV-1 hash function [13] which is used by both Voldemort [3] and Sheepdog [13]. Initially, Sheepdog storage cluster was formatted to use a default replication factor of 3.

A. Power savings

1) Power Mode Scheduler: Figure 6 shows the load metric for each hour and the number of active tiers (i.e. power mode) chosen by power mode scheduler for each hour of replaying *usr*. The load metric used was the number of servers needed to support a workload for each hour. The server metric is obtained by benchmarking the server in the system as mentioned in Section II.D. Power mode was defined as the number of tiers chosen to support the workload for each hour. Since we use the maximum load as the load for each hour, the load shown in figure 6 is referred as peak load. We make two observations. First, 90% of the time the power modes were correct. The number of active tiers chosen matches the workload for each hour. Second, the results show that 51 servers are sufficient to meet the performance requirement. Only a few hours of load are overhead. For example, the peak load during the period of 60^{th}-64^{th} hour cannot be sustained even with all tiers powered up.

2) Power Savings: Figure 7 shows the power savings of GreenCHT under twelve traces. The power savings are computed over sheepdog — the baseline system. GreenCHT

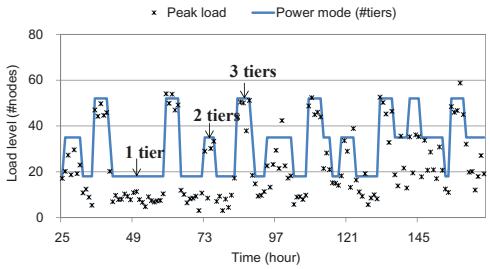


Fig. 6. Power mode schedule for GreenCHT system: High power mode with 3 active tiers, median power mode with 2 active tiers, and low power mode with 1 active tier.

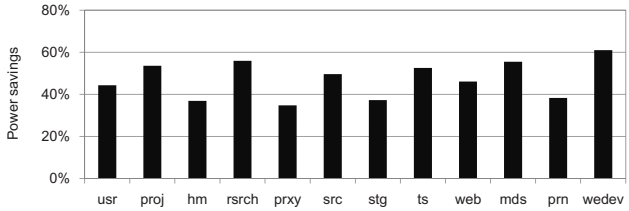


Fig. 7. Power savings for GreenCHT by using power mode scheduler's decisions.

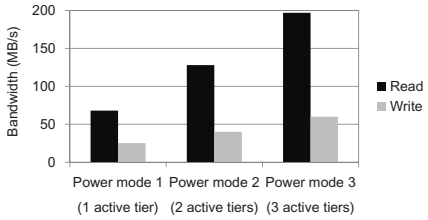


Fig. 8. Peak performance under different power modes (with different number of active tier)

provides significantly 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 sheepdog system do not use any power management strategy.

B. Performance

1) Peak Performance: We evaluate the peak performance of our system when clients are accessing objects under different power modes (with different number of active tiers) by using Sysbench [16]. 7 clients make random-access read and write requests to the system. Note that we use random read and write access performance as peak performance. Figure 8 shows the peak performance in terms of aggregated active server performance. Several observations can be made. First, in all cases, reads are faster than writes (by about 3×) because the replica level is 3 and the bandwidth of writes is reduced to $\frac{1}{3}$. Second, ideally, the performance of n tiers is as n times as the performance of a single tier for reads, and a third of that for writes. The actual performance of n tiers in those cases is close to the idea for reads as shown in figure 8. Third, the highest client performance we observed was about 35MB/s in power mode 3 when 3 tiers are all active for reads, and about 10 MB/s for writes. We also measure single-client streaming random read and write bandwidth from a single server with 3 active tiers by using Sysbench. The read performance was 104 MB/s

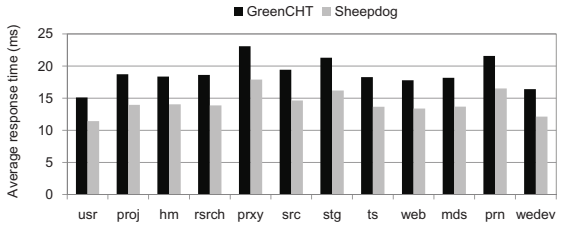


Fig. 9. Average response time for the two schemes under twelve traces.

while the write performance was about 17 MB/s. We only get about a third of the expected bandwidth of single-client streaming for reads and half bandwidth for writes because that the latency of disk seek time increases with concurrent random-access streams.

2) Latency: Figure 9 shows the average response time of GreenCHT system and sheepdog. Two observations can be made. First, the performance penalty for GreenCHT is small. The average response time of GreenCHT only increases by 4-5ms due to background data reclaim traffic. Second, compared with GreenCHT, sheepdog which has not power management policy acts like a baseline system. It shows the lower bound for average response time and upper bound for energy consumption.

V. RELATED WORK

A number of recent works have attempted to reduce power consumption in distributed systems. Sierra [9] exploits the redundancy in distributed systems, and powered down the servers hosting inactive replicas. Rabbit [7] uses equal-work data-layout, which stores replicas to nodes in a fixing order: Nodes are powered on by creating an expansion-chain, and the number of blocks stored on a node that is inversely related to where the node figures in the expansion-chain. Thus, the performance and the number of nodes that can be powered down would be scaled up at the fine granularity. Lightning and GreenHDFS [8] [1] focus on a data-classification, and separates cluster into hot and cold zones. Servers in cold zone can be powered down to save energy.

VI. CONCLUSIONS

In this paper, we propose an efficient power-proportional replication scheme for key value storage systems, called GreenCHT. GreenCHT uses a novel power-aware replication named multi-tier replication to provide power-proportionality. To ensure data consistency, GreenCHT uses log-store to temporarily store the writes to unavailable replicas. And GreenCHT uses a centralized power control service named predictive power mode scheduler to schedule nodes to be powered up and powered down without violating performance of the whole system. We implemented GreenCHT on Sheepdog storage cluster by modifying Sheepdog's original data distribution algorithm and replication strategy. And we evaluated GreenCHT by comparing it with the unmodified Sheepdog system. The experiment results show that GreenCHT saves significant energy while maintains an acceptable performance.

VII. ACKNOWLEDGMENT

We thankfully acknowledge the support of the National Natural Science Foundation of China under Grant No.61472152, the National Basic Research Program of China (973 Program) under Grant No.2011CB302303, the Fundamental Research Funds for the Central Universities, HUST:2015QN069, and the National Natural Science Foundation of China under Grant No.61432007 and No.61300047. We are thankful to Zhiyong Lu and Chenchang Tao for their helpful comments.

REFERENCES

[1] R. T. Kaushik and M. Bhandarkar, "GreenHDFS: towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster," In Proceedings of the 2010 international conference on Power aware computing and systems, HotPower10, pages 1-9, Berkeley, CA, USA, 2010. USENIX Association.

[2] A. Lakshman and P. Malik, "Cassandra - A Decentralized Structured Storage System," In Proceedings of the 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, 2009.

[3] Project Voldemort, A distributed Database. http://project-voldemort.com/. 2014.

[4] P. Hunt, M. K. Flavio, P. Junqueira and B. Reed, "ZooKeeper: Wait-Free Coordination for Internet-scale systems," In Proceedings of USENIX Annual Technical Conference, 2010.

[5] R. Bhagwan, D. Moore, S. Savage, and G. M. Voelker, "Replication strategies for highly available peer-to-peer storage," In Proceedings of FuDiCo: Future directions in Distributed Computing, June 2002.

[6] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazons highly available key-value store," In Proc. ACM Symposium on Operating Systems Principles (SOSP), Oct. 2007.

[7] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan, "Robust and flexible power-proportional storage," In SoCC, 2010.

[8] R. T. Kaushik, L. Cherkasova, R. Campbell, and K. Nahrstedt, "Lightning: Self-adaptive, energy-conserving, multi-zoned, commodity green cloud storage system," HPDC, 2010.

[9] E. Thereska, A. Donnelly, and D. Narayanan, "Sierra: A Power-Proportional, Distributed Storage System," Technical report, Microsoft Research, 2009.

[10] P. Maciel, R. Matos, G. Callou, B. Silva and S. Worth, "Performance Evaluation of Sheepdog Distributed Storage System," In Proceedings of the 2014 IEEE International Conference on Systems, Man, and Cybernetics October 5-8, 2014, San Diego, CA, USA.

[11] I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan, "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications," In Proceedings of ACM SIGCOMM (San Diego, California, Aug. 2001), pp. 149-160.

[12] D. R. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on theWorldWideWeb, ln Proc. 29th Annu. ACM Symp. Theory of Computing, El Paso, TX, May 1997, pp. 654-663.

[13] Sheepdog. https://github.com/sheepdog/sheepdog/wiki.

[14] MSR Cambridge Traces. http://iotta.snia.org/traces/388.

[15] D. Harnik, D. Naor, and I. Segal, "Low Power Mode in Cloud Storage Systems," In IEEE International Symposium on Parallel & Distributed Processing, 2009, pp.1-8.

[16] sysbench. http://sysbench.sourceforge.net/.

[17] J. Wan, X. Qu, N. Zhao, J. Wang, and C. Xie, "ThinRAID: Thinning down RAID array for energy conservation," Parallel and Distributed Systems, IEEE Transactions on. Sept. 2014. DOI: 10.1109/TPDS.2014.2360696.