An improved energy-efficient distributed storage system

Hongyan Li*

School of Information Management, Hubei University of Economics, Wuhan, China

Received 12 June 2014, www.tsi.lv

Abstract

Energy consumption has increasingly become a serious problem in contemporary data centres. The electricity bill contributes a significant fraction of the Total Cost of Ownership (TCO), and it is predicted to increase at an even faster pace in the following years as extremely large volume of data are being generated on a daily basis which would necessitate corresponding storage capacity to hold them. As a profitable work-around step toward the energy problem within the cloud infrastructure, in this paper, we propose REST, an energy-efficient cloud storage, which is built upon a cluster-based object store similar to GFS. It achieves high energy-efficiency by cleverly exploiting the redundancy already present in the system without compromising the inherently well-established schemes for consistency, fault-tolerance, reliability, availability, etc., while maintaining a reasonable performance level. By modifying slightly the data-layout policy, REST can safely keep a large amount of the storage nodes in standby mode or even powered off entirely most of the time. Deploying a sophisticated monitor, it also provides the flexibility to power up sleeping or powered down nodes when necessary to accommodate to the variations in workloads. Trade-offs between energy efficiency and performance can be conveniently made by simply adjusting a trade-off metric in REST. The FileBench and real world workload experimental results demonstrate that power savings can reach 29% and 33%, respectively, while still providing comparable or even surprisingly better performance.

Keywords: cloud storage, data centre, distributed storage, energy efficient, power saving

1 Introduction

With more and more internet services, outsourced storage services being concentrated in data centres and cloud computing infrastructures, added by a variety of data-intensive applications, like Google search engine, genetic projects, satellites images, data centres are increasingly getting filled with extremely large amount of data. Such huge storage requirements pose a lot of challenges to the IT management in terms of privacy, security, efficiency, energy consumption, etc. Even worse, the storage requirements have been reported to be rising by 60% annually. The phenomenal amounts of data in data centres not only call for tremendous investment on hardware, e.g. disks, to provide the corresponding storage capacity, but also need continuing power supply to feed the hardware. In large data centres, the energy cost consumed by the IT equipment over their lifetime is comparable to the hardware investment and occupies a significant portion of the TCO [1]. To make the situation more complicated, the power consumed never comes alone, but with many accompanying negative side effects, such as environmental impacts, noises, health disturbance. For example, according to the data from EPA, generating 1 kwh of electricity in the United States gives birth to an average of 1.55 pounds of carbon dioxide (CO2) emissions and consuming the same amount of electricity would further incur more emissions and heat which needs other additional electricity to keep data centres temperature from getting too high [2]. What's more, cutting the electricity bill is compelled by external factors and especially critical in certain situations. For example, in major cities the electricity prices are extraordinarily high and the requirements of increasing power supply may not be possible to be fulfilled at all [14].

Fortunately, many researchers from academic and industrial background have extensively investigated the power consumption problem and put forward a number of fruitful techniques to attack the problem over the past several years [3-8]. Generally speaking, those techniques can be classified into two broad subcategories: component-based solutions [6, 12] and system-level solutions [4, 5, 7, 8]. Since the data centres must be designed to account for the peak workloads, they are most of the time relatively over-provisioned due to the wide temporal variations exhibited by the workloads, e.g. diurnal peaks and troughs, which enable those techniques to be effective. However, while power-proportionality can be achieved relatively easier for some kinds of components, e.g. using dynamic voltage and frequency scaling (DVFS [13]) for CPUs, non-CPU components, especially like disks, are not power-proportional. Thus, to conserve energy consumption in storage subsystem, taking advantage of the observed idle periods between successive disk accesses is a common practice. For example, put some disks into standby mode under light or moderate workloads and try to manufacture and prolong

* ***Corresponding author*** e-mail: hongyanli78@aliyun.com

idle periods [4, 5, 7], both of which can be also perceived as forms of power-proportional.

Still, there remains a relatively less-explored spectrum of large-scale storage system power-saving space, which is from a high-level system design perspective. Distributed storage system systems such as GFS [9], HDFS [10], and KFS [11] are widely deployed as the backend storage infrastructure in large data centres and cloud computing infantries due to the aggregate high I/O performance and cost advantages over conventional SAN and NAS solutions. However, they were originally designed with little if not absolutely no power considerations. They were established on the assumption that instead of on enterprise-grade disks, they would be running on clusters consisting of hundreds of thousands of commodity servers, for which the unpredictable and sporadic fault or failure happenings should be considered as norm rather than exception [9]. Thus, they must be designed to be able to gracefully tolerate large numbers of component faults with little or no impact on service level performance and availability. Facing this hard situation, replication technique, a method widely thought to be able to provide high system reliability, better performance and high availability, had become a natural technique candidate to be deployed. As a result, each data block is replicated to a user defined level, typically three, replicas in those systems, resulting in large amount of redundancy. However, such redundancy at the same time introduces a lot of overheads to the system in aspects of storage capacity, replica consistency, networking bandwidth requirements, power consumption, etc. For the energy conservation consideration, an obvious and simple ideal occurs to us: is it possible to put some of the redundant nodes into power-saving mode or entirely power them down under light or moderate workloads while maintaining the existing sophisticated built-in mechanisms?

In the remaining of this paper, we present REST, a new cloud storage scheme based on a replicated, distributed file system KFS [11]. It aims to improve energy-efficiency without incurring severe performance degradation from the perspective of applications. The central point is that it turns down some redundant nodes under light or moderate workloads to conserve energy and also keeps the capability to power them up again in response to variations in workloads to prevent the performance from degrading too far. Our main contributions are: an energy efficient cloud storage scheme capable of reacting gracefully to variations in workloads and an architecture deploying new technology to manage cache and consistency under energy-saving mode which could be also potentially applied under disconnected conditions that happen frequently within large-scale systems. The design specifics are detailed in section 3. Experimental results have showed that REST has lived up to our expectations very well.

The remainder of the paper is structured as follows: Section 2 introduces the background and motivation. Section 3 details the design principles and implementation specifics. Section 4 presents our evaluation methodology and results. Section 5 makes a conclusive remark.

2 Background and motivation

Basically, many widely deployed distributed file systems share some common design and implementation strategies and tactics. They mainly consist of three parties assuming respective responsibilities: the client library, metadata server (MDS) and chunk-servers. The client provides API facilities to access the file system; MDS is the central component taking the responsibility of managing the whole file system name space; chunk servers are physical nodes where the data are actually stored. The objects stored in the systems are partitioned into chunks. Each chunk is replicated on multiple chunk servers to guard against disk or machine failures and to provide high performance and availability. The central MDS is implemented as an in-memory data structure so as to provide fast access speed, as it is visited much more frequently by normal operations and scanning checks [9]. It keeps all the metadata information. It is check pointed periodically to persistent storage to guarantee reliability and fast recovery in the event of MDS failure. Chunk servers communicate with MDS through frequent heartbeat messages reporting their storage status and receiving corresponding acknowledgement from MDS to maintain high availability. If the MDS notices some replicas are unavailable, it initiates re-replication process to prevent the data block getting under-replicated. From a high-level perspective, read and write requests are handled similarly. They both firstly go forward to the MDS to get the necessary information, like chunk servers' location and lease information, then directly contact the corresponding chunk servers to complete the data transfer, rendering the MDS out of the data transfer path. By doing this way, the involvement of MDS is minimized and the chances of its becoming potential bottleneck would thus be minimized.

However, there exists a significant difference between the phases of data transfer for read and write requests. For writes, all the nodes hosting the replicas should remain powered on for the purpose of strong consistency during the entire write process. For example, GFS [9] uses lease mechanism to define the write order of the replicas and applies pipelining technique to propagate the content. But for reads, after getting a list of chunk servers hosting the replica, the client would contact (may use an optimal algorithm, e.g. least distance first, to minimize the latency) one of them to fetch the replica, leaving the other ones unvisited if the firstly chosen chunk server succeeds in servicing the request. To get a holistic view of the read and write distribution, we modified the file access APIs of dbench4.0 [15] to those APIs exported by KFS, and then executed dbench4.0 on KFS. We configured KFS to consist of a MDS and 60 chunk servers. After the running

process finished, we analysed the statistics from MDS and all chunk servers. The overall number of reads is what MDS has recorded, while the total number of writes is the multiplicative result of the MDS recorded number and the replication factor. The read and write requests' cumulative distribution function (CDF) figure is shown in Figure 1.

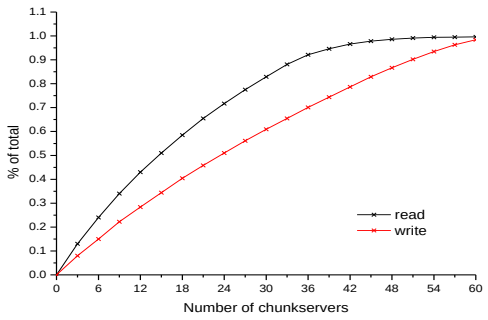

FIGURE 1 CDF of read/write among all the chunk servers

As shown in Figure 1, it shows a cumulative distribution function across the numbers of the chunk servers versus the percentage of read/write that the chunk servers experienced over the course of the running process. As pictured in the figure, read distribution embodies a wide discrepancy from write, i.e. read is more skewed than write, whereas write is approximately linear, which results from the inherent different handling processes. For example, nearly 83% of the reads were concentrated on 50% of the chunk servers, while 50% of the chunk servers had absorbed 61% of the total write operations. That's because, for read, every time the request for the same data block would with a strong likelihood be routed to the same chunk server hosting a copy (either primary or replicas) of the data block due to the same internal decision policy, e.g. judging by IP addresses, with the exception of the cases of chunk server having failed or the system topology having changed. But for write, because of the load-balance hinting data placing policy, especially for newly written data block, the write operations are more likely to be distributed uniformly among all the chunk servers. Such observed skewness in the read/write distribution motivates our work: is it possible to remove the write traffic from some chunk servers to make them presented with more read-dominated access patterns, which would lead to more skewness and exploit such skewness to conserve energy?

It is worth noting that due to hardware limitations, the experiment was not conducted on 60 physical chunk servers, but with each server hosting multiple chunk servers. Chunk servers are represented by different processes running on different ports with respective dedicated directories providing storage capacity. Such configurations have the following important implications: the chunk servers bear great homogeneity, have relatively flat network topology and are rendered to be equally subject to the network conditions. By contrast, in a genuine environment of large-scale storage system installation, like data centres, the situation is far away from here. Usually, it embraces a wide range of heterogeneity resulting from its hundreds of thousands of commodity components possibly differing widely in aspects of storage capacity, computing capability, network bandwidth, etc. Furthermore, it is typically constructed in a hierarchical form using different levels of switches to connect racks and chunk servers, which would make different constituents subject to various network conditions. However, under such a complicated situation, it is reasonable to assume the existence of similar or even better observations that we had experienced with our relatively simple and flat experimental setup. The assumption is based on the failure preference phenomenon observed in [16, 17], stating that failures tend to happen again to where they had happened before at a much higher probability. This phenomenon would potentially translate into more skewness in read/write distribution, which would present us a great opportunity to conserve energy as discussed in the following sections. Another thing that should be pointed out is that with no doubt we would rather like to admit dbench can never be representative of all read-world workloads, but we hope it would be reasonable to claim that though simple, it would shed some light on the problem we are discussing in this paper.

3 Design and implementation

As discussed in the preceding section, the skewness in I/O behaviour, especially in read distribution, may provide us with great potential to conserve energy. That is because skewness implies that some redundant chunk servers would remain idle even in the presence of I/O workloads. Thus, the node's storage subsystem or even the whole node if the node's only role is providing storage function can be transitioned into power-saving mode in a manner that is oblivious to the applications running on the system. This section is devoted to discuss how the energy-conservation potential can be realized. It deals with the details of the design and implementation of our prototype system REST. Specifically, it first outlines the design principles and goals that guide our design, and then presents the challenges and problems that should be resolved, followed by the overview architecture of REST. Finally, it discusses in depth the strategies and tactics we have deployed to achieve energy-efficiency purpose.

3.1 GOALS AND CHALLENGES

The ultimate goal of our work is to build an energy efficient storage system through exploiting the redundancy inherently existing in a replicated, cluster storage system. We choose the distributed file system KFS [11] as our basic architecture and develop our prototype on that. During the process of development, we strive to fulfil the following design disciplines and principles, which also act as our design guidelines:

- Changes made to the original system should be minimized.
- Energy efficiency should be obtained not at the expense of severe system performance degradation.
- Energy efficiency should be obtained without compromising reliability, availability, consistency and failure resilience.
- Trade-offs between energy efficiency and high performance should possibly be made by users and the system should be reasonably flexible to automatically respond to the workload variations.
- Failures of the components should be handled gracefully, ideally transparently, without adversely impacting the applications.

KFS was initially designed with almost no power-awareness considerations, but with its focus on building a high reliable, high available, high aggregate performance, scalable and fault-tolerant storage system on commodity-level components. Its salient feature is that reliability and availability can be well guaranteed even in the presence of failure occurrences to some components. The rationale behind our design is to conserve as much energy as possible while maintaining basic functions and desirable excellent features. To achieve this goal, the following challenges and problems should be well addressed:

- Data placement policy should be power-aware. The policy deployed by KFS places data blocks in a random way which would potentially cause all chunk servers to be highly correlated to each other. Using that policy, for example, at most N-1 chunk servers can be transitioned into power-saving mode to conserve energy if the system's replica factor is *N*.
- How many redundant chunk servers and when are they going to be put into power-saving mode while maintaining a reasonable performance level? And how to differentiate those deliberately powered-down servers from those that actually malfunction?
- Under what circumstances should the sleep servers be woken up? And in what manner are they woken up?
- What measurements should the system to take if failures occur?
- How to guarantee consistency among all replicas when some are temporarily not unreachable for power-saving purpose?

We present our approach to addressing those challenges and problems in the following subsections. At first, we give an overall description of the architecture of REST, and then we dive into the details of the design strategies and tactics integrated in REST.

3.2 SYSTEM OVERVIEW

As mentioned previously, our simple strategy is to exploit the redundancy present in the system and the skewness in IO distribution to realize energy-efficiency purpose. In order to do so, we have slightly changed the architecture of the original system and additionally integrated some functional modules into it. As shown in Figure 2, there are three main roles in REST: MDS as in the basic infrastructure with a newly added functional module instructor (not shown in the figure), loggers that are performed by dedicated high-performance servers and chunk servers. Note that the constant heartbeat messages among them are omitted for simplicity.

Compared with its basic architecture, REST differs itself in the following aspects: the back-end chunk servers are partitioned into several subsets, the new entering of loggers and an instructor that decides when and how many chunk severs are going to be powered down and up. Since the details are given in the subsequent subsections, we only give a brief description in the remain of this subsection. As shown in the figure, we strategically partition the entire space of chunk serves into several subsets. The most important one is named kernel subset coloured in red, and the others are named backup subsets coloured in green and light green, respectively. Ideally, the kernel is expected to remain powered on, while backups are to be kept powered down under light and moderate workloads. It is the responsibility of instructor that determines when to power backups down and when to up based on a number of factors. The up/down commands are piggybacked in the acknowledgements to periodic heartbeat messages from chunk servers. The loggers are designated the functions of providing temporary storage space to hold the data destined to the chunk servers that are powered down for that period of time, forwarding them to the corresponding chunk servers when they are powered on again and reclaiming the space. Read requests are sent to loggers as shown by read step: 1) (solid-line) if the requested data blocks exist there and the loggers are not too overloaded, otherwise they would be served by kernel or backup subsets as shown by read step; 2) (dash-line). Write requests are handled as usual in KFS if the kernel and backup subsets are all powered on, in other cases they are written to the powered-on chunk servers and loggers, and then immediately return to the clients to indicate write completion.

3.3 POWER-AWARE DATA LAYOUT

The inherent limitation that prevents the original system from powering down more than *N-1* chunk servers in an *N*-way replicated system lies in that the initial data block assignment policy logically imposes a strong tie between every pairwise, despite they are physically separate. For example, if *N* or more chunk servers were powered down, the data block of which all N replicas unluckily happen to reside on the *N* powered-down chunk servers or on a subset of the powered-downs would have been rendered unavailable.

To eliminate this limitation, we divide the whole chunk servers into several independent subsets named

kernel and backups, and for each data block, it is guaranteed that there would be at least one replica in each of the subset. This is achieved by the new data placement policy. For the convenience of our discussion, we define the following symbols: N for the total number of chunk servers in the system; r for replication factor; k ($1 \le k \le \left\lfloor \frac{N}{r} \right\rfloor$) for the size of kernel. As a system parameter, k is critical to the system performance and energy efficiency, since it determines the kernel's performance and thus, how often backups are going to be powered down and up, as discussed in the next following subsection. Fortunately, its value determination can be hinted by the individual performance of the chunk servers and operator's well-understandings of the characteristics of the expected workloads or it can be gradually adapted to the most suitable value using test-and-tune method. Backup subsets are obtained by equally partitioning the rest of the chunk servers, making its size $\left\lfloor \frac{N-k}{r-1} \right\rfloor$. The definitions of kernel and backups can be specified in advance in a system configuration file, and are all maintained in MDS. The chunk servers that belong to kernel and backups are typically chosen to have the same fault-tolerance properties, like a rack, respectively, for they are highly logically related, i.e. the failing of any one of them would render the data blocks residing on it unavailable within the corresponding subset. Interestingly, such strong relationship among physically separate nodes is exactly the reason that motivates our new data placement policy. However, doing so has several advantages, for example, utilizing less precious networking bandwidth and creating opportunity for power savings for switches, which would otherwise remain powered on, but also bears potential disadvantages, like being easily bottlenecked and forming high-temperature spots in large data centres.

When allocating a data block, MDS firstly allocates one data block for it from kernel, and then allocates the remaining replicas from backups, one from each. Within each of the subset, we also balance the data blocks among the chunk servers. With this kind of data placement policy, the minimum availability would safely be guaranteed by kernel, and backups can be powered down freely and independently when necessary without compromising system availability. MDS dynamically tracks the status of kernel and backups and differentiates the failed chunk servers and those powered down chunk servers. The placement policy imposes several changes to the read/write processes as discussed previously. When the kernel fails down, one of or all of the backups can be powered up to take the role of the kernel and rebuild/recovery the kernel, based on the trade-offs between greater power savings and higher rebuild/recovery speed.

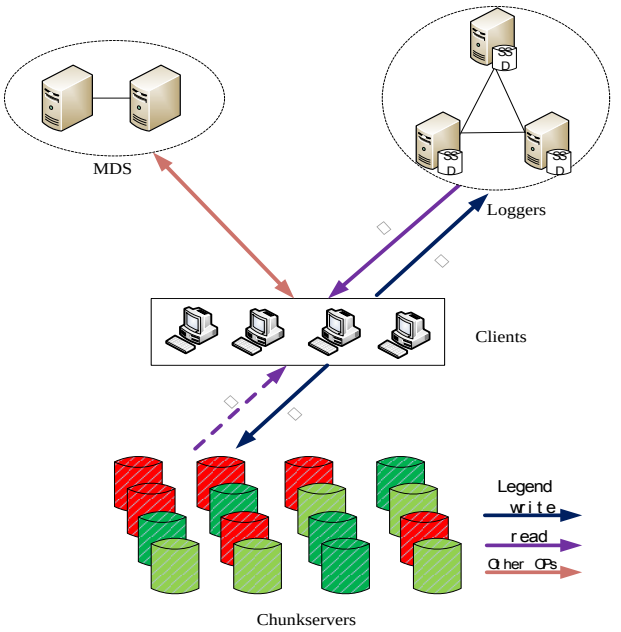

FIGURE 2 REST architecture

3.4 INSTRUCTOR

In reality, due to wide variations in workloads, it is improper to keep the number of powered down chunk servers constant. In REST, there is a central role named instructor that determines when and how many chunk servers are required to be powered down or up using WOL [18] technique. It periodically makes the decision based on a complex combination of multiple factors. Formally, the decision-making can be expressed as:

$$n = f(\lambda P, \beta R, \mu F, \eta C) \tag{1}$$

$$\lambda + \beta + \mu + \eta = 1 \tag{2}$$

In the above equations, n is the result of the decision making and indicates the number of the to-be-powered down or up chunk servers. P is defined as the ratio of performance to power, both of which are monitored by instructor, it reflects the system power efficiency; R is the required performance level, usually assigned as the minimum requirements that must be met; F represents the instructor's predication of the near-future workloads. It is obtained by analysing past workload characteristics and is used to instruct the decision. If it is predicated that the forthcoming write requests are very large or the writes would last for a long time, it will generate a larger value of F, indicating it is desirable to power more chunk serves up. The heuristic is that it tries to directly write large writes to all the chunk servers as possible as it can be, while redirect as many small writes as possible to the loggers, which is decided by two reasons: the loggers' capacity is limited and we want small write requests to be returned faster, since their latencies are more sensitive to the applications. If it is predicated that there are

enormous read-intensive access patterns in the near future, it would pro-actively prepare to power up more chunk servers to mask the relative long time, usually minutes, to power chunk servers up and to avoid degrading read performance too much. *C* is the health conditions of the kernel. Since the kernel is expected to be working all the time and unexpected failures in the kernel would be expensive due to the minimum availability being compromised, kernel health conditions [17] should be constantly monitored and reported to the instructor to take corresponding precautions, like replace new devices, if the health conditions are not so good. λ, β, μ and μ are their corresponding coefficients and can be assigned by the operator. They represent their relative importance in the process of decision making. For example, if the operator gives higher priority for performance than power-saving consideration, she/he can easily achieve that by assigning β bigger than λ.

In addition to the above power-down and up scheme, there are some other scenarios that the chunk servers should be powered up in REST. For example, in our current prototype implementation, the loggers and chunk servers' statuses are periodically reported to MDS through heartbeat messages. When MDS notices that if the overall space utilization of loggers has reached certain threshold, e.g. 80% or some members of kernel are unavailable, it would force all the chunk servers to be woken up immediately.

3.5 LOGGERS

In REST, unlike other techniques using dedicatedly reserved space on the existing devices, e.g. Eraid [19], DIV [7], write-offloading, we have used dedicated servers equipped with solid-state drives (SSD) to log the data destined to the temporarily powered down chunk servers. From the point of view of our considerations, there are four reasons for doing so. At first, we want the logged data to be persistently stored more reliably, and SSDs can fulfil our requirements, additionally, SSDs themselves consume much less power than conventional disks counterparts. Secondly, it is desirable to place logged data in different fault domains, typically different racks in data centres. Thus, even if the kernel subset fails down due to whole rack or switch failing down [26], it is still achievable to restore the data to the latest status with the logged data and the newly powered up servers. Thirdly, with the technology drastically advanced, the shortcomings formerly associated with SSD have already been well overcome [20] and their prices are not that high as before making them increasingly become acceptable and practical to be deployed in production systems. Lastly, SSD flash drives have blazingly fast read speed, including both of sequential and random patterns. We deploy this attractive characteristic to provide high performance for read requests by diverting read requests to loggers firstly if they are not overwhelmingly overloaded with traffics.

To take maximum advantage of SSDs and avoid their excruciating slow write shortcoming, we deploy a log-structured [21, 22] store engine to record the logged data. For each logged data block, there is corresponding information specifying whether it is logged and logged on which logger in its in-memory metadata entry in MDS namespace. In addition, each logger maintains a hash table in memory for itself. The table maps the unique 64-bit chunkID to the location where the on-disk entry resides within the store engine. The on-disk entry of logged data block is self-contained. It is composed of a logger header and a logger body. The logger header contains the following information: chunkID, version number, destinations, logger factor (meaning how many copies the chunk should be propagated to newly powered up chunk servers to in order to maintain the required replication factor) and chunk checksum. The logger body is the content of the data block.

For every data block write request, it would be firstly tried to be forwarded to the powered on chunk servers and if any of the destination chunk servers is powered down temporarily, it would be logged to one of the loggers before returning to the application. Specifically, MDS would choose a logger and send it a log request containing the necessary information and update its corresponding metadata entry. Receiving the log request, the logger appends an on-disk entry to the store engine, and then inserts a mapping entry into the local hash table, or replaces the mapping entry corresponding to the chunk if it has been inserted previously. That implies, for each logged chunk, there is only one mapping entry, which points to the most recent version of that chunk. For every read request, MDS would look the chunks in the namespace and preferably forward it to the loggers if it has been logged in any of the loggers, otherwise forward it to the powered on chunk servers. From the data paths of write and read requests, it would be expected that loggers would improve the system performance due to the log-structured design and SSDs' superior random read performance, which is actually proved by our experiments.

Considering the limited space of the loggers, we have designed each logger to initiate propagating and reclaiming processes at regular intervals, i.e. one minute. The propagating processes scans through the entire local hash table, and for each entry, it tries to contact those corresponding destination chunk servers to see whether they are reachable and if so it sends the logged data to them and update the entry information. The reclaiming process scans the on-disk entries from the beginning, and reclaims the space occupied by those entries whose logger factor is zero. It works in a similar way to the cleaner in [21]. Thanks to the two periodically-run processes, the loggers' utilization of space is reasonably prevented from going high quickly.

3.6 ENERGY MODELLING

To estimate the energy efficiency of REST, in this subsection, we analyse the energy models for both of the original system and REST. The following table summarizes the parameters in our analysis:

TABLE 1 Energy modelling parameters

Symbol	Description
N_c	The total number of chunk servers
N_l	Number of loggers
T_i^k	Length of the k[th] active interval of the i[th] chunk server
A_i	Number of active intervals of the i[th] chunk server
P_i^k	Power of the i[th] chunk server at time t within the k[th] interval $0 < t \le T_i^k$
E_i^u	Energy consumed by the i[th] chunk server to transition up
E_i^d	Energy consumed by the i[th] chunk server to transition down
C_i^u	Count of transition ups of the i[th] chunk server
C_i^d	Count of transition downs of the i[th] chunk server
P_j	Power of the j[th] chunk server when power up
T_j	Active time of the j[th] logger. Loggers have no up-and-downs

Two points should be noted about Table 1. One is that we have not outlined the energy consumed by MDS, since our main purpose is to compare the energy efficiency of the original system and REST. And we assume the amounts of MDS energy of them are approximately equal. We denote it as E_m in the following discussion. The other one is that we treat the instantaneous powers of loggers and chunk servers differently, i.e. we consider the power of loggers to be stable, while the power of chunk servers to be varying over time. Because we expect that individual chunk servers would experience bigger power gaps between peak power and the lowest power than loggers. Now we can calculate the total energy consumed by the original system using Equation (3).

$$E_b = E_m + \sum_{i=1}^{N_c} \sum_{k=1}^{A_i} \left(\int_0^{T_i^k} P_i^k \, dt \right). \tag{3}$$

Since the original system has no power up-and-downs, the numbers of active intervals of all the chunk servers are the same and equal to 1. Thus, Equation (3) can be translated into Equation (4).

$$E_b^{'} = E_m + \sum_{i=1}^{N_c} \left(\int_0^{T_i^1} P_i^1 dt \right). \tag{4}$$

Involving with the chunk servers power up-and-downs, REST has a more complex energy consumption formula:

$$E_R = E_m + \sum_{j=1}^{N_l} (T_j \times P_j) + \sum_{i=1}^{N_c} (E_i^u \times C_i^u) + \sum_{i=1}^{N_c} (E_i^d \times C_i^d) + \sum_{i=1}^{N_c} \sum_{k=1}^{A_i} \left(\int_0^{T_i^k} P_i^k dt \right). \tag{5}$$

It says that the total energy is the sum of individual components' energy: MDS, loggers, chunk servers. The differences between the two energy models lie in that REST divides the energy of chunk servers into active status energy, transition energy and powered down energy, which is zero, and has additional energy consumed by loggers, while the original system keeps the chunk servers up all the time. The active energy is the sum of all the energy consumed by all the chunk servers over all their respective active intervals. The power up-and-down overheads are the sum of all the energy consumed by transitioning. Comparing with the original system, REST's potential gaining power savings stem directly from how many and how long chunk servers are powered down.

4 System evaluation

In this section, we evaluate REST from various aspects using benchmarking method and realistic workload. We also test the original system, which is referred to NO-REST later as baseline for comparison reason. Section 4.1 describes our test environment. Experimental results are discussed in the subsequent subsections.

4.1 EXPERIMENTAL SETUP

Our test bed consists of one MDS server and a number of chunk servers composed of 4 servers and 32 commodity PCs belonging to our lab's graduate students. The hardware configuration of the MDS server is characterized by a quad-core 2GHz CPU 16GB RAM and 16 1TB hard disks. One of the 4 servers is equipped with a quad-core 2GHz CPU, 4GB RAM and a Kingston 128 GB SSD disk drive and functions as a logger. The remaining three servers have the same configurations: a quad-core 2GHz CPU, 4GB RAM and 8 1TB hard disks structured as RAID5. The other 32 chunk servers bear a wide variety of configurations and performance, since they were purchased in the different years when their respective owners were enrolled in our lab.

We conducted our test using both benchmarking and realistic workload. The benchmark is FileBench [23], an application level workload generator that enables the users to emulate various workloads. Its Workload Model Language (WML) provides users with the capability to flexibly define the workload bearing different characteristics. A WML workload description is called a personality and it typically contains the following information about the workload: average file size, directory depth, the total number of files, and alpha parameters governing the file and directory sizes that are

based on a gamma random distribution [24]. FileBench can define the period of time for which the personalities are going to be run, and report the total number of performed operations at the end of each run. We selected four personalities included in FileBench to drive our testing. They are Web, File, Mail and Database servers, and their workload characteristics are specifically described in [24]. Each of them was run for a period of 1 hour. We deployed fuse support of REST and NO-REST to access the storage like conventional file systems. The realistic workload is a shared server workload in our lab. The server is shared by all lab staff doing their own jobs, e.g. upload/download files, doing backups, visiting CVS source code repository, etc., it is also the backup server of B-cloud [25] system developed at our lab which provides on-line backup services. We configured REST and NO-REST as the backend storage infrastructures of the server, respectively and monitored the server for 48 hours dating from 8:00 am GMT on Nov. 10 to 8:00 am GMT on Nov. 12 to observe the workload characteristics and how REST performs.

We emulated 76 chunk servers for both REST and NO-REST for fair comparison. Each of the 3 servers hosted four chunk servers, and each of those PCs hosted 2 chunk servers. The logger server emulated 2 loggers in REST. Both of REST and NO-REST were configured as three-way replicated systems. The kernel was set to include all those chunk servers hosted on the 3 high performance servers and its size was set to 24. For each run, we initially powered on all the chunk servers.

4.2 PERFORMANCE IMPACTS

We compare the performance of different workloads in terms of the performance metric reported by FileBench. FileBench reports file system performance under different workloads in units of operations per second (ops/sec). Due to the peculiar characteristics of the Web server workload, FileBench would report much higher ops/sec than the other workloads. In order to avoid the results figures being too skewed, we present Web result in units of 100 ops/sec and ops/sec for the others. Figure 3 shows the performance results.

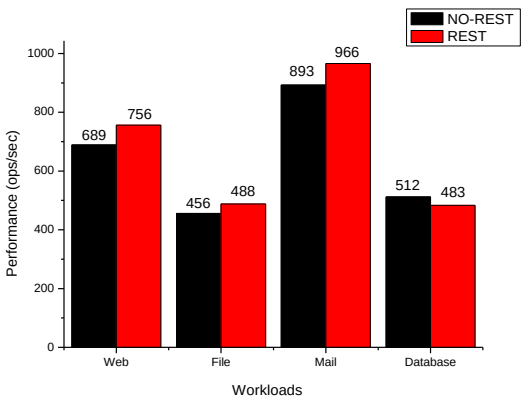

FIGURE 3 Performance of different workloads with REST and NO-REST

It is shown in the figure that REST well outperforms NO-REST for Web, Mail, File, achieving 9.7%, 8.17% and 7% performance gains respectively, while consuming less energy as discussed in the next subsection. And Database slightly lags behind NO-REST by a margin of 5.7%. Generally, the performance gains can be attributed to the newly added loggers in REST. Because reads are preferentially considered to be routed to loggers, which can provide excellent read performance, including both random and sequential reads and writes are redirected to loggers when the destination chunk servers are powered down temporarily. Due the log-structure, writes can be returned to applications much sooner, thus enhancing performance.

It is interesting to note that the amounts of percentage performance gains are closely related to the read/write ratio. And it's surprising to know that the performance gains are proportional to the energy savings. For example, Web conserves the most energy but with the most performance improvement, which are 17% and 9.7% of the NO-REST counterpart respectively, due to its highest read/write ratio and its sequential reading entire files patterns. Analysing the workloads, we know Web, File, Mail and Database's R/W ratios are 10:1, 1:2 and 20:1 respectively. It reveals that workloads with higher percentage of reads can get more energy savings and the chosen kernel can reasonably satisfy the read requests in most scenarios. However, there is an exception to that: Database has the biggest R/W ratio, but exhibits degrading performance. It is partly because, besides 200 readers, it launches 10 asynchronous writers and a log writers. In addition, they perform extensive concurrency and random read/writes, which would adversely prevent chunk servers from powering down. Another reason is that the writers' extensive writing operations would quickly cause the loggers' space utilization threshold to be reached and chunk servers to be waken up more frequently.

4.3 TRANSITIONS AND POWER SAVINGS

In this section, we discuss the chunk servers transitions and power savings of REST. Power savings were calculated by substituting the parameters in our energy models with corresponding dynamically monitored numbers and representative real world values. As pointed out previously, the chunk servers' transitions are determined by the instructor. We define a trade-off metric T as λ/β, i.e. T = λ/β. It reflects the preference degree for power efficiency, meaning the bigger T is, the more power savings are desired. To see how instructor affects the power up-and-downs, we collected all chunk servers up-and-downs for the four workloads with varying T (as a side note, the preceding section's results are from experiments conducted with T = 1 for REST) value while keeping η and μ invariable and let their sum equal to 0.3. The statistics are summarized in Table 2.

TABLE 2 Transition counts summary

Workload	T=0.5	T=1	T=1.5	T=2
Web	126	113	110	95
File	243	220	201	198
Mail	189	165	146	134
Database	74	87	102	121

The table demonstrates that for Web, File and Mail, the number of transitions decrease with *T* increasing. Bigger value of T implies trying to achieve better energy efficiency, which means once chunk servers are powered down, they would be powered up under more serious conditions. Again, Database is the exception with transitions increasing with *T* increasing. As explained before, the nature of Database prevents chunk servers transitioning, but bigger value of *T* tries to force transitioning, causing REST struggling powering down and up.

Since the energy savings and performance of all the four workloads share similar characteristics, we take Mail as our example to discuss power savings and performance with varying value of *T*. Figure 4 shows the relationship between them. The energy savings and performance are percentages relative to its NO-REST counterpart. It is apparent that power saving and performance can be traded off with different *T* values. For example, we can get 29% power savings at 83% performance level, or alternatively, we can enjoy 112% performance level at the cost of less potential power savings, which is 10%. This has important practical implications for applications, i.e. performance and power efficiency can be flexibly traded off in REST by simply changing *T*.

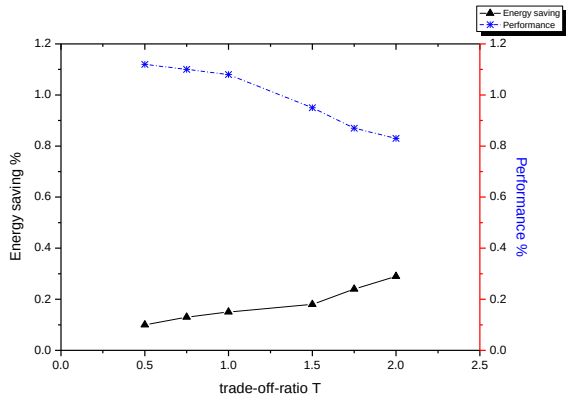

FIGURE 4 Energy saving of REST relative to NO-REST for Mail at different *T* value

4.4 REALISTIC WORKLOAD EXPERIMENT

For our realistic lab workload, we obtained similar energy efficiency and performance results. The power saving surprisingly reaches as high as 33% due to the wide variations in the workload, while maintains comparable performance level. In addition to that, we sampled the REST dynamical number of powered on chunk servers every 2 hours over the 48 hours experiment period. The result is portrayed in Figure 5.

It shows that the workload exhibits periodicity, and REST responded to that in a power-proportional way. For each of the two days, we observed that at 12:00, 18:00, the up chunk servers are more than other times. We find that is because our lab members often saved their work on to the server before leaving the lab, resulting in peak time in workload. And for the spike points at 24:00 each day, we assume that is caused by our B-cloud server services. Several small and medium size companies are using B-cloud for their daily backup tasks, and they usually do backups at that point of time. We also notice there is a spike point at 04:00 on Nov 12 due to a kernel network partition occurrence. When failures occur to the kernel, REST would wake up chunk servers swiftly to rebuild/recovery the kernel.

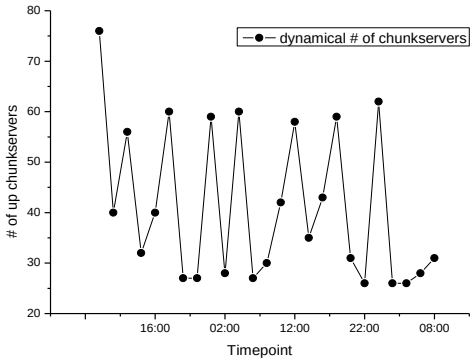

FIGURE 5 Dynamical number of powered on chunk servers over the experiment period References

Our real world lab workload experiment has revealed that REST has practical applicability. It can save energy by exploiting both the redundancy in the storage system and real workload characteristics, like periodicity and burstness in the workload.

5 Conclusions

In this paper, we present REST, a redundancy-based energy efficient cloud storage system. Motivated by the observations of workloads' periodicity and asymmetric phenomenon in read and write requests, we suggest in REST powering down the whole redundant chunk servers to achieve energy efficiency. We explicitly explain the techniques that we deployed in REST, including power-aware data layout, instructor, loggers and the energy models. Our experimental results show that a reasonable amount of energy savings can be achieved at comparable or unexpected better performance level, especially for realistic workload.

References

[1] Belady C 2010 In the data centre, power and cooling costs more than the IT equipment it supports *Electronics Cooling*

[2] Zong Z, Briggs M, O'Conner N, Qin X 2007 An energy-Efficient Framework for Large-Scale Parallel Storage Systems *Proc. Int'l Parallel and Distributed Processing Symp., March 2007*

[3] Benini L, Bogliolo A, de Micheli 1999 A survey of design techniques for system-level dynamic power management *IEEE Transactions on VLSI Systems* **8**(3) 813-33

[4] Papathanasiou A E, Scott M L 2004 Energy efficient prefetching and caching *In Proc. of the USENIX Annual Technical Conference, June 2004*

[5] Zhu Q, David F M, Devaaraj C F, Li Z, Zhou Y, Cao P 2004 Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management *Proc. High-Performance Computer Architecture, 2004*

[6] Zhu Q, Chen Z, Tan L, Zhor Y, Keeton K, Wikes J 2005 Hibernator Helping Disk Arrays Sleep Through The Winter *Proc. ACM Symp. Operating Sys. Principles, October. 2005*

[7] Pinheiro E, Bianchini R, Dubnicki C 2006 Exploiting redundancy to conserve energy in storage systems *In Proceedings of the 2006 SIGMETRICS Conference on Measurement and Modeling of Computer Systems. Saint Malo, France, June 2006*

[8] Kaushik R T, Bhandarkar M 2010 GreenHDFS: Towards an energy-Conserving, Storage-Efficient, Hybrid Hadoop Compute Cluste *In HotPower*

[9] Ghemawat S, Gobioff H, Leung S-T 2003 The Google File System *In the Proceedings of the 9th Symp. on Operating Systems Principles, Oct. 2003*

[10] Konstantin S, Kuang H, Radia S, Chansler R 2010 The hadoop distributed file system *MSST*

[11] http://kosmosfs.sourceforge.net/

[12] Barroso L A, HÖlzle U 2007 The case for energy-proportional computing *IEEE Computer* **40**(12) 33–37

[13] Chen Y, Das A, Qin W, Sivasubramaniam A, Wang Q, Gautam N 2005 Managing server energy and operational costs in hosting centres *In SIGMETRICS '05: Proceedings of ACM SIGMETRICS International Conference on Measurement and Modelling of Computer Systems* **33** 303–14

[14] Rakesh Kumar 2006 Gartner: A message from data centre managers to CIOs: Floor space, power and cooling will limit our growth, August 2006

[15] http://dbench.samba.org/

[16] Clement A, Kapritsos M, Lee S, Wang Y, Alvisi L, Dahlin M, T Riche M 2009 UpRight cluster services *In SOSP*

[17] Pinheiro E, Weber W-D, Barroso L A 2007 Failure trends in a large disk drive population *In Proc. USENIX Conference on File and Storage Technologies (FAST'07), San Jose, CA, Feb. 2007*

[18] http://en.wikipedia.org/wiki/Wake-on-LAN

[19] Li D, Wang J 2005 Conserving Energy in RAID Systems with Conventional Disks *In Proceedings of the International Workshop on Storage Network Architecture and Parallel I/Os, Sept 2005*

[20] http://www.samsungssd.com/

[21] Rosenblum M, Ousterhout J 1991 The design and implementation of a log-structured file system *In Proc. ACM Symposium on Operating Systems Principles (SOSP'91), Pacific Grove, CA, Oct. 1991*

[22] Ganesh L, Weatherspoon H, Balakrishnan M, Birman K 2007 Optimizing power consumption in large scale storage systems *In Proc. Workshop on Hot Topics in Operating Systems (HotOS'07), San Diego, CA, May 2007*

[23] FileBench, July 2008 www.solarisinternals.com/wiki/index.php/FileBench.

[24] Sehgal P, Tarasov V, Zadok E 2010 Evaluating performance and energy in file system server workloads *In Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST '10), February 2010*

[25] Wei J, Jiang H, Zhou K, Feng D 2010 MAD2: a scalable high-throughput exact deduplication approach for network backup services *MSST*

[26] Ford D, Labelle F, Popovici F, Stokely M 2010 Availability in Globally Distributed Storage Systems *In proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, 2010*

Authors

Hongyan Li, born on October 12, 1978, Wuhan, China

Current positions, grades: Now she is a PhD candidate in school of Computer Science & Technology of Huazhong University of Science & Technology. She is an associate professor at School of Information management, Hubei University of Economics. Since 2008 she is Member of IEEE.
University studies: received her M.Sc. in Computer Science (2005) from Central China Normal University.
Scientific interests: different aspects of storage Systems.
Publications: (co-)authored more than 20 papers.