Microsoft Research Ltd. Technical Report MSR-TR-2009-153, November 2009

Sierra: a power-proportional, distributed storage system

Eno Thereska, Austin Donnelly, Dushyanth Narayanan
Microsoft Research, Cambridge UK

Abstract

We present the design, implementation, and evaluation of Sierra: a power-proportional, distributed storage system. I/O workloads in data centers show significant diurnal variation, with peak and trough periods. Sierra powers down storage servers during the troughs. The challenge is to ensure that data is available for reads and writes at all times, including power-down periods. Consistency and fault-tolerance of the data, as well as good performance, must also be maintained. Sierra achieves all these through a set of techniques including power-aware layout, predictive gear scheduling, and a replicated short-term versioned store. Replaying live server traces from a large e-mail service (Hotmail) shows power savings of at least 23%, and analysis of load from a small enterprise shows that power savings of up to 60% are possible.

1 Introduction

Server power consumption is a major problem for both small and large data centers, since it contributes substantially to an organization's carbon footprint and power bills. Barroso and Hölzle have argued for *power proportionality* [3]: the power used should be proportional to the system load at any time, rather than to the peak load that the system is provisioned for. A power-proportional system could exploit temporal variations in load, such as the diurnal peaks and troughs (e.g., the "Pacific Ocean trough" [6]) seen in many user-facing services.

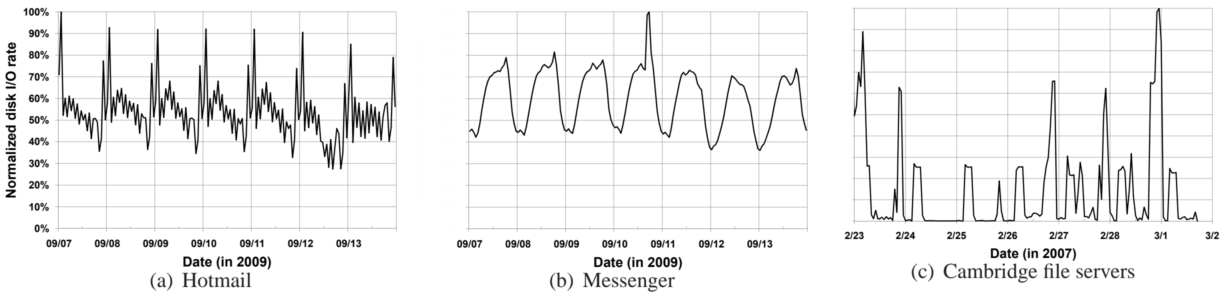
Storage is a key component for many large, scalable and highly available services such as web email (Hotmail or Google Mail), Amazon's EC2, and Windows Azure [1]. Because storage is not power proportional, it limits the power proportionality of the whole data center.

Some power-proportionality is achievable at the hardware level, e.g., using dynamic voltage scaling (DVS) for CPUs. However non-CPU components, especially disks, are not power-proportional.

We believe that at data center scale, the most promising approach is to power down entire servers rather save power in individual components. This is the position we take in this paper. It is challenging to power servers down and yet maintain high service availability. Although most services have clear troughs, the troughs do not go to zero. Furthermore, the system must still provide good availability and performance for users during troughs. To maintain service availability, all the load must first be migrated from the servers to be powered down, and consolidated on the active servers. Storage presents a challenge here again. While computational state can be migrated and consolidated, e.g., using virtualization techniques, it is not possible to migrate terabytes of on-disk state per server daily.

In this paper we present Sierra, a power-proportional distributed storage system. Sierra is a replicated object store that allows storage servers to be put into low power states (standby or powered down) when I/O load is low. It does this without making the storage unavailable or sacrificing consistency, performance, and fault tolerance. This paper addresses these challenges entirely from the point of view of distributed storage. Megascale data centers often co-locate computation and storage on the same servers; thus a complete solution would integrate Sierra with existing techniques for CPU consolidation.

Sierra exploits the redundancy that is already present in large-scale distributed storage, typically in the form of three-way replication across servers. The design of Sierra is based on the idea of running the system in a lower "gear" — a smaller number of active replicas per object — when load is low. This allows servers hosting inactive replicas to be powered down. In a three-way replicated system Sierra allows up to $\frac{2}{3}$ of the storage servers to be in standby. Sierra avoids sending client requests to the powered-down servers, and ensures that there are always active servers that can serve accesses to every object. This ensures that all objects are available even when a majority of the servers are powered down.



(a) Hotmail (b) Messenger (c) Cambridge file servers

Figure 1: One week of I/O load for two large and one small service

There are several challenges in ensuring that gearing down does not compromise the availability, fault-tolerance, or performance of the storage system. First, at least one replica of each object must be on an active server even in the lowest gear. Second, the system must allow updates when in low gear, and these updates must be kept persistent, consistent, and have the same replication for fault tolerance as data written in high gear. Third, there should be enough active servers to handle the load at any given time, and the load should be balanced across the active servers. Finally, gearing down should not impede recovery actions taken when a server fails either transiently or permanently.

Sierra achieves these goals through a combination of techniques. A power-aware layout scheme ensures that all objects are kept available even when a significant fraction of servers is powered down. A predictive gear scheduler exploits observed diurnal patterns to schedule servers for power-up and power-down. Read availability is provided through a pro-active primary migration protocol. Write availability is provided by using a short-term versioned store. These techniques also ensure that the system maintains read/write consistency, tolerates any two transient or permanent server failures, and can efficiently re-replicate the contents of a failed server.

This paper makes three contributions. First, using real load traces from both large and small services as evidence, we show that there are significant diurnal troughs in I/O load, which can be exploited for power savings. Second, we describe the design of a power-proportional distributed storage system that exploits these troughs without compromising consistency, availability, load balancing, or fault tolerance. Third, we present an evaluation of a prototype system running on a hardware testbed, using I/O traces from production servers of the Hotmail service, and achieving power savings of 23%.

2 Evidence and motivation

The design of Sierra is motivated by the observation that many workloads in both small and large data centers have periodic I/O load patterns with large peak-to-trough ratios. CPU load also shows these characteristics, and virtual machine migration and voltage scaling can be used to save CPU power during the troughs. There are no analogous solutions for disk, and hence we need a new approach to storage power proportionality.

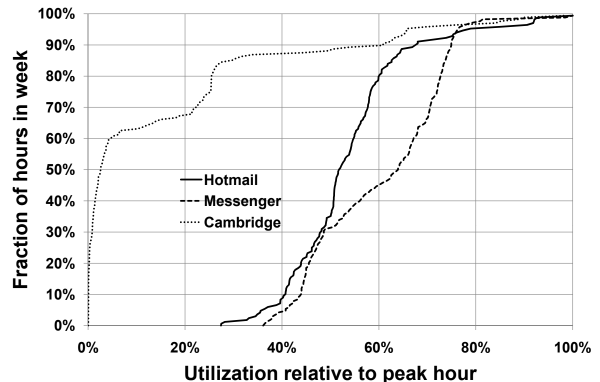


Figure 2: CDF of utilization for all three services

Figure 1(a) and 1(b) show the aggregated I/O load for two large online services, for 1 week. The load is aggregated over tens of thousands of servers for Hotmail (Windows Live Mail), and thousands of servers for Windows Live Messenger. The graphs show the total number of disk bytes transferred, aggregated across all the back-end storage in the service at one-hour intervals. The load is normalized to the peak value for each service. We observe clear periodic patterns with significant peak-to-trough ratios. In general, this load correlates with the expected diurnal variation for user-facing services. Thus, there seems to be substantial scope for operating with fewer resources during the troughs without significantly impacting request latency. We observed that an alternative, consolidation of resources across services, did not eliminate the troughs. Much of the load is correlated in time and furthermore, Hotmail needs an order of magnitude more servers than the other services.

Figure 1(c) shows the variation in I/O traffic for a different environment: a small/medium enterprise with a

small number of on-site servers. Specifically we show the I/O aggregated across 6 RAID volumes on two file servers at MSR Cambridge; the graph is derived from the publicly available MSR Cambridge I/O traces [12, 7]. Here, the periodicity is less obvious; however there are clear peak and troughs, indicating a significant potential for power savings during the troughs.

Figure 2 shows the CDF of time spent at different utilization levels, where utilization is normalized to the peak seen in each trace. Substantial periods of time are spent with utilization much lower than the peak. Hence there is significant scope for exploiting these periods for power savings, even in a system that is optimally provisioned, i.e., with no power wasted at peak load.

3 Design and implementation

The base system model for Sierra is that of a replicated, cluster-based object store similar to GFS [5], the Windows Azure blob store [1], FAB [10], or Ursa Minor [2]. The base system is designed to provide scalability, load balancing, high availability, fault tolerance, and read/write consistency. We first briefly describe the basic architecture of Sierra, which is largely based on current best practices. We then outline the challenges in making such a system power-proportional without losing the other properties, and present in detail the techniques Sierra uses to address these challenges.

3.1 Basic architecture

Figure 3 shows the basic architecture of Sierra. Sierra provides read/write access to objects in units of *chunks*. The chunk size is a system parameter: a typical value is 64 MB. Each chunk is replicated on multiple *chunk servers*; the default replication factor is 3. For any given chunk, one of the replicas is designated as the primary at any given time, and the others are secondaries. At any time a chunk server will be the primary for some of the chunks stored on it and a secondary for the others. All client read and write requests are sent to the primary, which determines request ordering and ensures read/write consistency. Since we are designing a general-purpose storage system for use by a variety of applications, Sierra supports overwriting existing data in addition to appends of new data. Client reads and writes can be for arbitrary byte ranges within a chunk.

Read requests are sent by the primary to the local replica; write requests are sent to all replicas and acknowledged to the client when they all complete. Load balancing in Sierra is done by spreading a large number of chunks uniformly over a smaller number of chunk servers, and also by choosing primaries for each chunk randomly from the available replicas of that chunk.

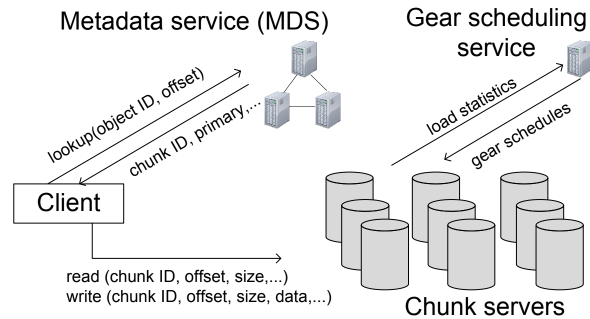


Figure 3: Sierra architecture

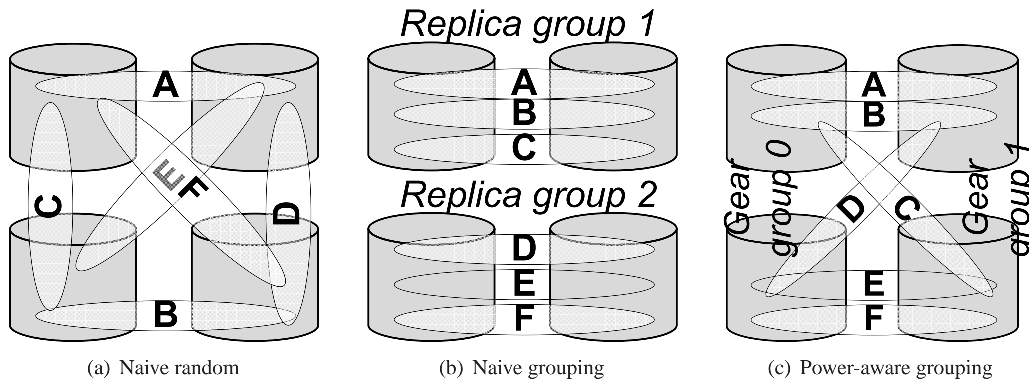
Sierra has a centralized metadata service (MDS), which functions as a naming service. It is implemented as an in-memory, deterministic state machine which can be replicated for high availability using state machine replication techniques [11]. It maps each object to its constituent chunks, and each chunk to its current primary. The MDS is not on the data path between clients and chunk servers; its state is updated when chunks are created or deleted but not when they are read or written.

The MDS also tracks chunk server availability, reassigns primaries as necessary, and initiates recovery actions when a server fails permanently. Availability is tracked through periodic heartbeats from chunk servers to the MDS. In response to each heartbeat, the MDS sends the chunk server a lease for the set of chunks that it is currently the primary for, and the locations of the secondaries for those chunks. Leases are set to expire before the MDS times out the heartbeat, and servers send fresh heartbeats before their leases expire. The MDS reassigns primaries on demand for chunks whose primaries have lost their lease. A chunk server with an expired lease will return an error to a client trying to access data on it; after a timeout period the client fetches and caches the new chunk metadata from the MDS.

3.2 Challenges

This baseline design provides good consistency, availability, load balancing and fault tolerance when all or most chunk servers are available. However, there are several challenges in saving power and maintaining these properties when significant numbers of servers are periodically powered down and up.

The first challenge in achieving power savings is *layout*: the assignment of chunks to chunk servers. Section 3.3 shows how a “naive random” approach, although simple and good for load balancing, does not give a good tradeoff between power savings and availability. This is surprising because this is the most commonly used approach today. To address the problem we devised a power-aware layout, described in Section 3.3. This al-



(a) Naive random (b) Naive grouping (c) Power-aware grouping

Figure 4: Different layouts for 6 chunks, 4 servers, and 2-way replication.

lows an r-way replicated system to be in any gear g such that only $\frac{g}{r}$ of the servers need to be active to keep g replicas of each object on an active server.

The second challenge is to correctly predict the number of servers required at any time to sustain the system load, so that excess servers can be turned off. This is done by the *gear scheduler* component of Sierra. This component tracks and predicts the load on chunk servers on a coarse time granularity (hours to days). Based on load predictions, it then sends "gear schedules" to the chunk servers specifying when they should shut down and start up. We have found that a simple predictor based on the hour of day works well on the workloads we have evaluated. Section 3.4 describes the load metrics and prediction algorithm used by the gear scheduler.

A third challenge is maintaining availability and load balancing during gear transitions, i.e., server power-ups and power-downs. While we use standard techniques to deal with server failures, we do not treat server power-ups and power-downs identically to failures. The reason is that, unlike failures, we expect a significant fraction of the servers to change their power state at least once over the period of a day. Section 3.5 describes how Sierra pro-actively migrates primaries away from a server that is about to be powered down. Hence, at every gear level, Sierra ensures that every chunk primary is on an active server, and that the primaries are uniformly distributed across the currently active servers.

While primary migration gives availability and load-balancing for reads, we also want to support writes to chunks during low-gear periods, when all replicas of the chunks are not available. Further, we want writes during low-gear periods to maintain the replication factor r; to maintain read/write consistency (i.e., every read sees the result of the last committed write); and to ensure that writes are eventually applied to all chunk replicas when they become available. Sierra achieves this using a *replicated short-term versioned store*, described in Section 3.6. This is a service that can use spare disk bandwidth and capacity on existing chunk servers, and/or a small number of dedicated servers. This mechanism also lets Sierra chunk servers support overwriting of chunk data, even when one or more replicas have failed. Hence chunk data is kept available for both reading and writing as long as at least one replica is on an active server.

As a final challenge, in addition to existing failure modes such as chunk server failures, Sierra must handle new failure modes, e.g., failure of an active replica when other replicas are powered down. Section 3.7 describes fault tolerance in Sierra, focusing on the novel failure modes and solutions in Sierra.

3.3 Power-aware layout

The layout defines the way in which chunks are assigned to chunk servers at chunk creation time. Our goal is a layout that allows $\frac{r-g}{r}$ of the servers to be powered down (where r is the replication factor and $0 \leq g \leq r$) while keeping g replicas of each chunk available. We refer to this as putting the system in the gth gear. Thus, a 3-way replicated system could be in gears 3, 2, 1, or 0. Gear 0, while possible in theory, is unlikely to be useful in practice, due to the high latency penalty of waiting for a server to start up to service a request.

One simple approach is the *naive random* layout: each new chunk is assigned replicas on three servers chosen at random. However, we discovered that this severely limits the scope for power savings. This layout makes it hard to power down more than $r - 1$ servers in the entire system. Figure 4(a) shows a simple example with 4 servers, 6 chunks, and 2-way replication. Since every chunk is 2-way replicated, we would like to be able to put 2 of 4 servers into standby at low load and yet keep one replica of each chunk available. With the layout shown, however, it is not possible to put more than one server into standby (there is at least one object for which both replicas would be unavailable if we did so). As the size of the system increases, the independent random layout

	Power-down	**Rebuild**
Naive random	$r - g$	N
Naive grouping	$N\frac{r-g}{r}$	1
Power-aware grouping	$N\frac{r-g}{r}$	$\frac{N}{r}$

Table 1: Number of servers that can be powered down in gear g, and the write parallelism for data rebuild. N is the total number of servers and r is the replication level.

makes it increasingly improbable that more than $r - g$ servers can be powered down while still keeping g active replicas of every chunk. Additionally, with this layout, finding a maximal set of servers that can be turned off without losing availability is likely to be computationally hard, since it is a special case of the NP-complete set covering problem.

An alternative approach is to put servers into *replica groups*, each of size r (Figure 4(b)). A chunk is then assigned to one replica group rather than to r independently chosen servers. Now we can switch off $r - g$ servers in each replica group and still have g replicas of each object available. However, naive grouping of servers reduces the *rebuild parallelism*. When a server suffers a permanent failure, with naive grouping its entire contents must be rebuilt (re-replicated) on a single new server, and this server becomes the bottleneck for the rebuild process. With a typical commodity disk with a write bandwidth of 80 MB/s and 1 TB of data, this would take 3.6 hours. With the naive random approach on the other hand, each chunk stored on the failed server can be independently rebuilt on any of the other servers in the system. This gives a high degree of rebuild parallelism, and hence a higher rebuild rate.

Sierra uses a generalized notion of grouping that achieves both power savings and high rebuild parallelism by using *power-aware grouping*. Each server is assigned to exactly one of r *gear groups*. Each new chunk is assigned exactly one replica from each gear group; selection of the server within each gear group is done uniformly at random. Now the system can be put into any gear g by turning off $r - g$ gear groups. If a server in some gear group G fails, then its data can be rebuilt in parallel on all remaining servers in G. Thus the rebuild parallelism is $\frac{N}{r}$ where N is the total number of servers.

Table 1 summarizes the three approaches. Note that all three layouts are equivalent with respect to load balancing of client requests, since all three allow chunk replicas and primaries to be spread uniformly over servers.

The above analysis also applies to cross-rack replication. For each of the three layouts, we can further constrain the replica choice such that different replicas are in different fault domains, typically different racks in the data center. However, the naive random policy will still only be able to turn off $r - g$ racks, rather than $\frac{r-g}{r}$ of all servers. Similarly the naive grouping policy can only rebuild over one rack rather than $\frac{1}{r}$ of the racks.

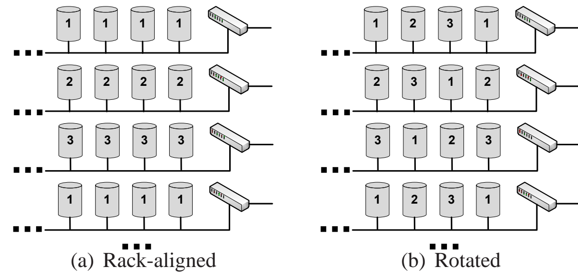


Figure 5: Two ways of configuring gear groups

Sierra uses power-aware grouping with cross-rack replication. It supports two variants: rack-aligned and rotated (Figure 5). While both allow the same number of servers to be turned off in a given gear, the location of the servers is different. In the rack-aligned case, all servers in a given rack are in the same power state; this could allow some additional power savings, for example, by turning off rack-wide equipment such as switches. However, the rotated layout might be useful if it is more important to distribute the powered-up servers (and hence the thermal load) evenly across racks. For concreteness, in the rest of the paper will assume a gear-grouped, cross-rack, and rack-aligned layout.

3.4 Gear scheduler

The main aim of gear shifting is to exploit the 24-hour cycle in load. It does not make sense to gear-shift at a time scale of minutes or seconds, since a server can take several minutes to start up and several seconds to come out of standby. Hence, in Sierra we gear-shift on a time scale of hours: the aim is to shift gears a few times a day to capture the broad diurnal patterns in load.

The load metric used by the gear scheduler uses the rate of reads and of writes, both aggregated over all the chunk servers; the write rate is weighted by a factor r, since each write is replicated r times. It also considers separately the random-access I/O rate measured in IOPS, and the streaming I/O rate measured in MB/s. Given the known performance per chunk server in terms of IOPS and MB/s, the load can then be computed in units of the number of chunk servers required to sustain the load.

$$L_{nonseq} = \frac{TotalIOPS_{read}}{ServerIOPS_{read}} + r \cdot \frac{TotalIOPS_{write}}{ServerIOPS_{write}}$$

$$L_{seq} = \frac{TotalMBPS_{read}}{ServerMBPS_{read}} + r \cdot \frac{TotalMBPS_{write}}{ServerMBPS_{write}}$$

$$L = \max(L_{nonseq}, L_{seq})$$

The load is measured at 1 sec intervals on each chunk server, and aggregated once an hour. By default we use the peak (i.e., maximum) load observed in the hour as the load for that hour: since I/O load is often bursty, using the mean value can significantly degrade performance during bursts. The gear scheduler then predicts the load for each hour of the following day, by averaging the past load samples for that hour in previous days.

To compute the gear g for a given hour, the gear scheduler measures whether the predicted load for that hour exceeds $\frac{1}{r}$, $\frac{2}{r}$, etc. of the total number of chunk servers N. It then chooses the lowest gear which leaves enough servers active to sustain the load:

$$g = \left\lceil \frac{L}{N} r \right\rceil$$

The Sierra gear scheduler is a centralized component that periodically aggregates load measurements from all the chunk servers, and computes the gear schedules for the following day. For any given gear g, the servers in the first $r-g$ gear groups are scheduled to be powered down. The gear schedules are then pushed to the chunk servers, and each server follows its own gear schedule. Over several days, the gear scheduler rotates the ordering of the gear groups, so that in the long term all servers spend an equal amount of time powered up. This allows all servers to do background maintenance tasks, e.g., scrubbing, during idle periods.

3.5 Primary migration protocol

Sierra balances load by spreading primaries uniformly across active servers, and hence across active gear groups. This means that when servers in some gear group G are powered down, any chunk primaries on those servers would become unavailable, and clients would not be able to access those chunks. Since Sierra handles server failures using a heartbeat mechanism (Section 3.7), we could simply treat the power-down events as failures; the MDS will reassign the primaries when it detects a failure. However, this will result in data being unavailable until the "failed" server's leases expire.

To avoid this, Sierra chunk servers pro-actively migrate all chunk primaries onto other replicas before powering down, using the following migration protocol. If a server S wishes to power down, it executes the following protocol for each chunk C for which S is a primary:

1. S updates its in-memory state to mark itself as a secondary for chunk C. An error will be returned on future client requests. Client requests currently in flight will complete normally.
2. S signals the MDS with $released_primary(C)$.
3. The MDS randomly picks another replica S' of the chunk C as its primary and modifies its internal state to reflect this.
4. The MDS signals S' with $become_primary(C)$.
5. S' initializes any required in-memory state and starts servicing client requests as the primary for C.

The window of unavailability for chunk C is now one network round trip plus the time required to update MDS state and initialize the new primary on S'. If a client accesses C during this short window it will retry the operation, converting this temporary unavailability into a higher latency.

When the chunk server S has no more primaries or outstanding requests, it sends a final "standby" message to the MDS and goes into standby. The MDS then sends "gear shift" messages to all the peers of S (i.e., servers which share one or more replicated chunk with S) to inform them that S is no longer active. This is an optimization that avoids peers of S timing out on requests to S when accessing it as a secondary. When a chunk server S wakes up from standby it resumes sending heartbeats to the MDS. When the MDS receives a heartbeat from a server that was previously in standby, it rebalances the load by moving some primaries from other servers to S. This is done by sending S a list of chunk IDs to acquire primary ownership for, and the current primary for each. S then contacts each of the current primaries, which then initiate a primary migration protocol similar to the above.

In Sierra, chunks are collected into chunk groups to reduce the number of MDS operations involved in primary migration. All chunks in a chunk group are guaranteed to be replicated on the same servers, and have the same primary at any given time. Migration of primaries requires one MDS operation per chunk group rather than per chunk. Chunks are assigned randomly to chunk groups on creation. The number of chunk groups is a system parameter that trades off MDS load for fine-grained load balancing across servers. Sierra currently uses $64N$ chunk groups where N is the total number of chunk servers.

3.6 Replicated short-term versioned store

We want writes to be stored persistently and consistently even when chunk replicas are powered down or otherwise unavailable. The replicated short-term versioned store is used by Sierra primaries to ensure these properties. The basic design of this store is similar to that used in our previous work in the context of RAID arrays [7, 8], but has now evolved as a distributed system component. Here we give a high-level description of its basic properties and (network) optimizations in the distributed setting.

When one or more secondaries is unavailable (powered down for example), a Sierra primary enters "logging mode". In this mode it sends writes to the short-term ver-

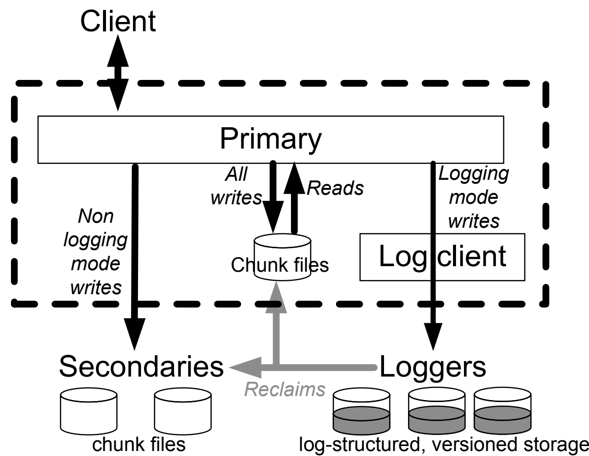


Figure 6: Data paths in logging and non-logging modes. The dotted box represents a single chunk server acting as a primary.

sioned store instead of the secondaries. When all secondaries are available again, the primary starts *reclaiming* the writes: data is read from the versioned store, written to all replicas, and then deleted from the versioned store.

Although the short-term store can share disk resources with the existing storage, i.e., the chunk servers, it is logically a separate service. Versions are essential in the short-term store to ensure that the state visible to clients is consistent and recoverable. The chunk servers on the other hand are designed to run on standard file systems such as NTFS or ext3, which do not support versioning. Using a separate implementation for the short-term store also allows us to optimize it for the write-dominated workload and the relative short lifetime of stored data items. Specifically, the short-term store uses a log-structured disk layout which is known to work well in this scenario.

The short-term store has two components: a *log client* that is associated with each primary, and *loggers* which send writes to a local on-disk log file. When in logging mode, the log client sends each write to r *loggers* in r different racks, maintaining the same fault-tolerance properties as the chunk data. Log clients track the location and version of logged data in memory; this state can be reconstructed from the loggers after a failure. Figure 6 shows the logging and reclaim data paths from the point of view of a single chunk server primary.

When sending writes to the versioned store, the primary also writes them to the local replica. This allows reads to be served from the local replica and reduces the load on the versioned store. It also avoids rewriting this cached data to the primary during reclaim. The metadata that tracks these locally cached writes is kept in memory, is small, but is not recoverable. Thus, this optimization only helps primaries that have not failed or migrated. In all cases correctness is maintained, with data being read from the loggers, if necessary, to service a client read.

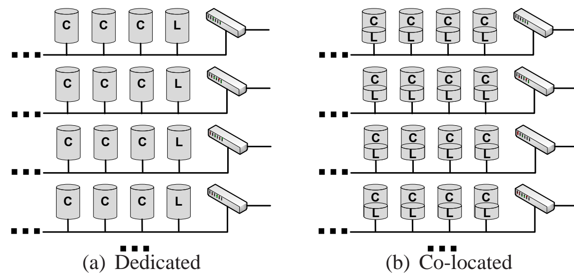


Figure 7: Two ways of configuring loggers (L) and chunk servers (C)

Loggers can be run on dedicated servers or co-located with chunk servers. Figure 7 shows examples of a dedicated and a co-located configuration. The dedicated configuration has the advantage that it minimizes contention between the chunk server workload and the logger workload, specifically allowing the loggers to service mostly writes, for which they are optimized. The dedicated configuration does require additional resources; however, we expect that these additional resources will be small, e.g., one dedicated logger per 20 chunk servers in a rack.

For each logging mode write, the log client can choose any r available loggers that are on different racks. In our previous work using the versioned store, these were chosen primarily by disk load. However, in a scalable distributed system such as Sierra, it is important also to minimize the network overheads of using the versioned store, and especially to minimize the use of scarce cross-rack network bandwidth. Hence for every log write, the log client for a chunk group G sorts the loggers in its logger view in the following order:

1. Loggers on the same server as a replica of G,
2. Loggers in the same rack as a replica of G,
3. Loggers in other racks.

Within each of these groups, loggers are sorted by disk load. For each log write the client greedily chooses the first r loggers that are in different racks. For log reads and reclaims only one logger is needed, the one closest to the primary (either co-located or on the same rack).

3.7 Fault tolerance and recovery

Sierra uses standard techniques (heartbeats and primary reassignment) to maintain read availability during transient errors; additionally it uses the short-term versioned store to maintain write availability. Here we describe how we handle new failure modes resulting from gear-shifting.

Failures when in low gear: Chunk server failures

might occur when in low gear, when some of the chunk servers are already in standby. When the MDS detects failure of a chunk server S, it wakes up all the servers that share any chunks with S. Since wakeup from standby typically takes a few seconds, and even powering up a machine can be done in minutes, this does not significantly increase the window of vulnerability for a second and third failure. However, when the system is already in the lowest gear (gear 1), failure of a server can cause the last active replica of a chunk to become unavailable while other replicas are being woken up. This will result in a large latency penalty for any client accessing the chunk during this window.

The Sierra gear scheduler takes the *minimum gear level* g_{min} as a policy input. The value of this parameter depends on the desired tradeoff between power savings and the risk of temporary unavailability on failure. We expect that for a 3-way replicated system, g_{min} will typically be 1 (for higher power savings) or 2 (for higher availability). Gear 0 is problematic because any access to a chunk will see a large latency penalty even in the absence of failures. $g_{min} = 3$ will not save any power.

Logger failures: Logger servers can also fail. When a server fails, this log data becomes unavailable; however, two other replicas of each log record are still available on other servers. Thus, logged data has the same level of fault tolerance as unlogged data. One option to maintain this fault tolerance is to re-replicate data within the logging service on failure. However, since the data will eventually be reclaimed back to the chunk replicas, this results in wasted work. Instead, Sierra primaries reclaim at high priority any at-risk data, i.e., logged data with fewer than three available replicas. At the end of the reclaim, the data will be no longer on the loggers but three-way replicated on chunk servers.

Permanent failures in low gear: On a permanent failure, the MDS initiates the rebuild of data stored on the failed server; this requires peers of the failed server to be powered up to participate in the rebuild. Sierra powers up peers of a server S whenever a transient failure is suspected on S. Hence the time to power up the peers is overlapped with the detection of permanent failure. In any case, the time to transfer the data to new replicas dominates the total recovery time. Hence, waking up machines in standby does not significantly increase the window of vulnerability to a second permanent failure.

Replica divergence: In a primary/backup replication system such as Sierra, it is possible for a server failure during a write request to result in replica divergence, with some replicas having applied the write and others not. Note that this problem is not specific to Sierra but to all systems which apply updates concurrently to replicas. In Sierra primaries react to update failures by re-sending the update to the versioned store. The versioned store avoids replica divergence by using explicit versions. If the primary fails, then the client will retry the request, and the new primary will send the update to the versioned store. However, if both the primary and the client fail while a write request is in flight, then replica divergence is possible. If this scenario is a concern, then *chain replication* [13] could be used, where updates are applied serially to the replicas rather than in parallel. Chain replication prevents replica divergence at the cost of higher update latencies. We have not currently implemented chain replication; however, adding it to the system only requires small changes and is orthogonal to the gearing and power-saving aspects of Sierra.

3.8 Implementation status

The evaluation in the following section is based on our Sierra prototype, which is implemented entirely at user level, with the MDS and each chunk server each running as a user-level process, and a client-side library that exports object *read()*, *write()*, *delete()* and *create()* calls. The core Sierra implementation is 10 KLOC of C code, with an additional 8 KLOC for the logger and log client implementations. Although the MDS is implemented as a deterministic state machine we have not currently implemented MDS replication; however, standard techniques exist for state machine replication and we are confident that the MDS could be replicated if required.

4 Evaluation

In Section 2 we saw that large data center services such as Hotmail as well as small data center services such as the Cambridge file servers, have substantial potential for power savings. Realizing this potential using Sierra requires sufficient, predictable troughs in the I/O load. Additionally, we would like the baseline system to have good, scalable performance, to maintain performance while in low gear and while transitioning between gears, and to have efficient rebuild of data on server failure. In this section we evaluate these different aspects of Sierra using real workloads as well as microbenchmarks.

First, Section 4.1 evaluates the accuracy of our load prediction algorithm as well as the expected power savings, from the three workloads described in Section 2: Hotmail, Messenger, and Cambridge. This analysis is based on coarse-grained measurements of load aggregated over the entire services for one week.

The rest of the section then evaluates the Sierra prototype running on a cluster testbed. We use I/O request traces from a small sample of Hotmail servers to measure the power savings and performance on real hardware. Using microbenchmarks, we then show the scal-

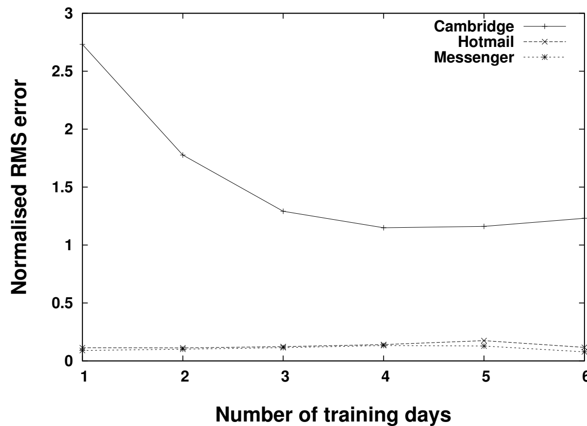


Figure 8: Prediction accuracy

ing of the base system's read/write performance and data rebuild rate as function of layout.

4.1 Load trace analysis

We applied the simple "hour of day" load prediction algorithm (see Section 3.4) to the aggregated load measurements described in Section 2; the load metric used was mean bytes transferred per hour, since that is all we have available from this data. For each of the three workloads, we have 7 days of data. We train the predictor on the first n days of the data and test it on the remaining $7 - n$ days. The error metric is the root-mean-square (RMS) error, normalized by the mean load during the test period to give a scale-free measure of error. Figure 8 shows how the error changes as n is increased.

We see that for Hotmail and Messenger, the error is low even after a single day of training and does not change significantly afterward. For Cambridge this error, while initially high, drops as more training data is used. These load errors translate into gear selection errors as follows (using 6 days for training and 1 for testing): for Hotmail all gears are correct; for Messenger, 90% of the time the gears were correct; for Cambridge, 75% of the time the gears were correct.

The left half of Figure 9 estimates the corresponding power consumed, defined as the average fraction of servers that cannot be switched off using Sierra. The numbers assume correct gear selection as defined by the actual load (i.e., an "oracle" predictor). The power savings look promising, but we cannot deduce how well workloads would perform in lower gears. In the next section, we evaluate power savings and performance for a live run of Hotmail I/O traces on real hardware.

4.2 Hotmail I/O traces

The next few sections show results obtained using I/O traces from 8 Hotmail back-end servers over a 48-hour period, starting at midnight (PDT) on Monday August 4 2008. Note that these I/O traces are from a different time period than the Hotmail load traces shown so far. The I/O traces are taken at the block device level, i.e., below the main memory buffer cache but above the storage hardware. During our experiments we disable caching, prefetching and write-backs, thus enabling accurate trace replay. During trace collection, each I/O to a block device results in a trace record containing the timestamp, the device number, the type of request (read or write), the logical block position accessed, and the number of blocks read or written.

The I/O traces include accesses both to data files (e-mail messages), which form the bulk of the storage capacity used, and metadata databases (user profiles, search indexes, etc.). Data files can be directly stored as objects in Sierra, whereas the metadata would be best stored using distributed tables. Since Sierra is a blob store and not a distributed table service, we ignore accesses to the metadata, which would have to be hosted elsewhere.

We map the traces to Sierra using *virtual disks*. Each block device in the trace maps to a virtual disk, which corresponds to a unique object ID in Sierra. The virtual disk object is then stored in Sierra as a set of chunks corresponding to logical extents within the disk. Thus, the trace replay mechanism converts an access of <*block device, logical block number, size in blocks*> to <*object ID, offset in bytes, size in bytes*>.

We measure the power savings and performance of Sierra based on these traces. As we only have 2 days of traces it is not meaningful to train a model for gear prediction. Hence, we use the "oracle" gear selection policy for these I/O traces.

4.3 Testbed setup and provisioning

Our experimental testbed consists of 31 identical servers in 3 racks in one data center. Each rack has a Cisco Catalyst 3750E as a Top-of-Rack (ToR) switch providing 1 Gbps ports for the servers, and a 10 Gbps fiber uplink to a Cisco Nexus 5000. The testbed is assigned 10 servers in each rack, plus an extra server in one of the racks on which we run the MDS. Each server has two four-core 2.5 Ghz Intel Xeon processors, 16 GB of RAM, a 1 TB system disk and a 1 TB disk that holds the Sierra chunk files and log files. Although the machines have plentiful RAM, we do not use it for caching in our experiments, to match the traces available, which are taken below the main memory buffer cache. Each server runs Windows Server 2008 Enterprise Edition, SP1.

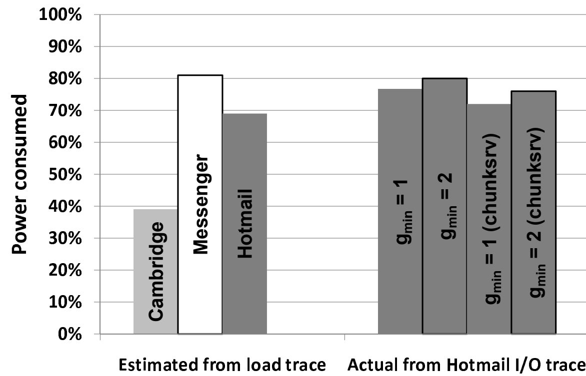


Figure 9: Estimated power consumed for three services and actual power consumed for the Hotmail I/O trace

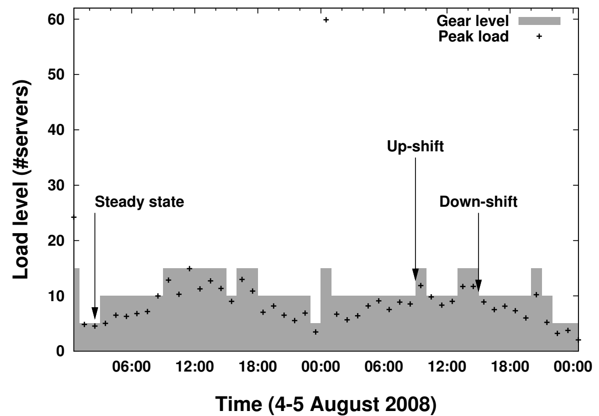


Figure 10: Gear schedule for Hotmail I/O trace

A note on terminology: throughout the paper we use bytes, not bits, e.g., MB, not Mb, and standard powers-of-two notation, e.g., 1 KB is 1024 bytes (not 1000).

For meaningful experimental results it is important to correctly provision the system for the workload, i.e., chose the correct number of chunk servers. Overprovisioning the system would increase the baseline system's power consumption unnecessarily, and thereby inflate the relative power savings of Sierra. Underprovisioning the system would also be meaningless because the system could not sustain the peak load even with all servers powered up.

In addition to performance, we must also match the availability and capacity requirements of the workload. For availability, we place each replica in a separate fault domain (in our case, in a separate rack). For capacity, we are limited by the total storage capacity of our servers, which is not sufficient to hold the entirety of the virtual disks in the trace. However, it is sufficient to store the specific chunks that are accessed during any of our experimental runs, if we use a chunk size of 1 MB. Hence, for each experiment, we pre-create exactly the chunks that are accessed during the experiment, using a chunk size of 1 MB. In practice a larger chunk size, e.g., 64 MB is more common [5].

To calculate the number of chunk servers needed to support a workload, we use the load metric L described in Section 3.4. It converts four workload metrics — streaming read and write bandwidth (in MB/s) and random-access read and write IOs per second (IOPS) — to a single metric in units of chunk servers. The server metrics (e.g., $ServerIOPS_{write}$), are obtained by benchmarking the servers in the system (Table 4 in Section 4.6 shows the results.) We then use the maximum value of L over the trace, rounded up to the nearest multiple of 3 for 3-way replication.

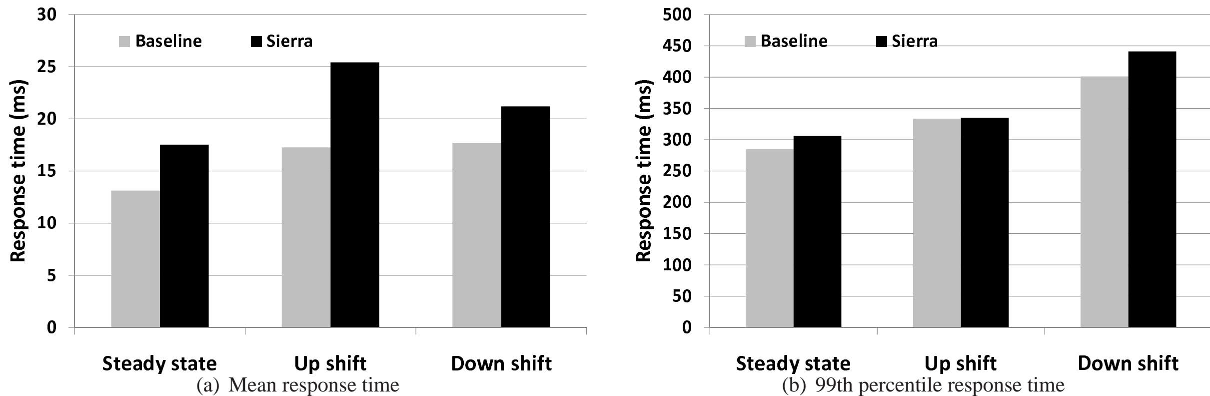
The peak load in our traces happens just after midnight during what we believe is a 2-hour period of maintenance background activity. We do not provision for this peak, but for the peak excluding this maintenance window. We do keep the maintenance period in the highest gear though. If we provisioned for the background peak, or ran the 2-hour period in gear 1 (intuitively the background jobs need to complete but are unlikely to have tight performance requirements) we would save more power.

For the Hotmail traces this methodology resulted in 15 chunk servers to meet the performance requirement (5 in each rack). The trace replay clients are load balanced on 9 of the remaining machines (3 in each rack). For all the performance experiments, we compared two configurations. The *Sierra* configuration had the above 5 chunk servers and 1 dedicated logger per rack. The *Baseline* configuration was provisioned with the same total resources, i.e., 6 chunk servers per rack and no loggers. In all cases we pre-create the necessary system state, including logger state, by first replaying the relevant preceding portions of the trace (up to 8 hours in one case).

4.4 Power savings

This section presents the results of replaying the Hotmail traces on the Sierra testbed. We first show the power savings achieved using the gear scheduler's decisions. We then present the performance results from live runs.

Figure 10 shows the load metric for each hour of the Hotmail I/O trace, and the gear chosen by the oracle predictor for each hour. The right half of Figure 9 shows shows the actual power consumed as a percentage of the baseline system, which has the same total number of servers (18). We show the results for two policies: the default policy ($g_{min} = 1$) where the system is required to keep a minimum of one active replica per object, and a "high availability" policy ($g_{min} = 2$) where two replicas are always kept active per object, to avoid any temporary unavailability if a server fails.



(a) Mean response time

(b) 99th percentile response time

Figure 11: Performance comparison for Hotmail I/O trace

We make two observations. First, the actual power savings from this real run match the analytical expectations for Hotmail. Second, even with a $g_{min} = 2$ policy the power savings are significant. These numbers include the power consumption of the loggers. Since our testbed is small, and we need one logger per rack at minimum, the loggers' contribution is actually larger than in a real system. The last two bars in Figure 9 show the power consumed by the 15 chunk servers alone, as a percentage of a baseline system with 15 chunk servers. We see that the power savings are very close to that predicted from the Hotmail load traces (for $g_{min} = 1$). In a larger system setup we expect a much lower ratio of loggers to chunk servers than the 1:5 seen here (this expectation is confirmed in the next section), hence the relative power cost of the loggers will decrease.

4.5 Performance and efficiency

For the performance experiments, we selected three trace segments that represent the worst case for three aspects of Sierra. The *steady-state* experiment chooses, of all the 1-hour periods spent in the lowest gear, the one with the highest peak load. The aim is to show the performance impact of putting the system in a low gear. The *Up-shift* experiment chooses the transition into the highest gear having the largest amount of logged data. The aim is to show the performance impact of reclaiming logged data to the chunk servers. The run is from 10 minutes before the transition until 1 hour after the transition. The *Down-shift* experiment chooses the down-transition having the highest load in the minute immediately following the transition. The aim here is to show the effect on clients of the primary migration protocol and any resulting client retries. The run is from 10 minutes before the transition until 10 minutes after the transition. Figure 10 shows the times in the trace corresponding to the three experiments.

Total data logged in 48 hrs	166 GB
Time in top gear	14 hrs
Required reclaim rate	3.4 MB/s
Data reclaimed in up-shift	22 GB
Achieved reclaim rate	6.3 MB/s
Logger avg. queue size (steady state)	0.09
Logger avg. queue size (reclaim)	2.3

Table 2: Reclaim statistics

Request response times: Figure 11 shows the performance of the baseline and Sierra configurations during the three experiments; we show both the mean response time and the 99th percentile response time. We make several observations. First, given that these are the worst three scenarios, the performance penalty for Sierra is small. Second, the steady-state and down-shift experiment results show that our provisioning methodology is reasonable; the performance in the lower gear (Sierra) is comparable to the performance in gear 3 (Baseline). For both experiments the main reason performance slightly degrades is that our provisioning method only considers first-order performance metrics (IOPS and streaming bandwidth). In reality, workloads have second-order properties too (e.g., spatial and temporal locality) that our method does not capture. Third, the up-shift experiment sees the worst performance degradation of all three, since the foreground workload in the highest gear interferes at the disk with the reclaim process. We look next in depth at the reclaim rate and possibilities for improving the performance during up-shift even further.

Reclaim rate: A key requirement is that the reclaim rate be sufficient so that the amount of logged data does not increase over the course of a day, since the loggers are not provisioned or optimized for long-term storage. Hence, we also estimated the required reclaim rate (top half of Table 2), by measuring the number of unique bytes logged just before each up-shift in the 48-hour pe-

Number of migrations	102
Total migration time	28 ms
Number of retries	77

Table 3: Down-shift statistics

	Writes	Reads
Bandwidth (MB/s)	82/**82**/82	82/**82**/82
IOPS	144/**179**/224	129/**137**/147

Table 4: Single-server performance. The min/**avg**/max metric is shown for 5 runs.

	Writes	Reads
Bandwidth (MB/s)	96 *(246)*	348 *(738)*
IOPS	465 *(537)*	1152 *(1233)*

Table 5: Peak performance. In brackets is the *ideal* performance as nine times the performance of a single server for reads, and a third of that for writes.

riod, and summing these values. This gives an upper bound on the amount of data that needs to be reclaimed when in high gear. Another key requirement is that the reclaim process should not interfere with the foreground workload. We saw in the previous experiment that the interference can lead to some performance degradation.

The bottom part of Table 2 shows the measurements from the up-shift run. We easily meet the first requirement: the reclaim rate should be 3.4 MB/s, and we achieve 6.3 MB/s. Note that these reclaim rates indicate the network is unlikely to be a bottleneck. We can meet the second requirement and thus decrease performance degradation further if we throttled the reclaim process to 3.4 MB/s. We believe the right way to slow this process down is to have support for native background (low-priority) I/O in Sierra, where the reclaim I/Os would get background priority. We are in the process of implementing this support.

Since the chunk servers are currently the bottleneck for reclaim, we can increase the number of chunk servers per logger until the loggers or the network become a bottleneck. In our experiments we found that the loggers had an average disk queue size of only 2.3 while reclaiming, indicating a low level of load to achieve almost twice the target reclaim rate. The logger queue size in the steady-state experiment was only 0.09. The network usage is similarly low. Hence we believe that in a larger system we can support a substantially higher ratio of chunk servers to loggers than in our small testbed.

Primary migration: We measured the time it takes to migrate primaries when down-shifting. This process should be quick, so that few client requests have to retry. Measurements from the down-shift experiment indicate that it took a total of 28 milliseconds to migrate 102 chunk group primaries, as shown in Table 3. Within this period, each primary would have a smaller window of unavailability corresponding to its own migration. This resulted in 77 client requests retrying once (out of 254,536 total requests in the 20 minute interval).

Metadata state size: We measured the amount of metadata for the longest experiment, the up-shift one. The metadata service had about 100 MB of state in-memory. It contained information on 320 chunk groups and 4.6 million chunks. Hence, the MDS has on average 23 bytes of data per chunk or 320 KB of data per chunk group, i.e., a very small overhead.

4.6 Microbenchmarks

The goal of this section is to measure using microbenchmarks, the scalability of Sierra's read/write performance as well as the impact of layout on rebuild rates.

Single-server performance: This experiment establishes a baseline single-server performance. First, we measure single client streaming read and write bandwidth from a single server in MB/s using 64 KB reads and writes to a 2.4 GB file. Second, we measure random-access read and write performance in IOPS (I/Os per second) by sending 100,000 IOs to the server. The client keeps 64 requests outstanding at all times. Table 4 shows the results. Write performance is more variable than read performance due to inherent properties of NTFS.

Multi-server performance: This experiment shows the peak performance of our system when multiple servers and clients are accessing data. Rack 1 is used for chunk servers. Rack 2 is used for the clients. The metadata service is placed on a machine in rack 1. 9 clients (each identical in setup to the single one above) make read and write requests to 9 chunk servers. Table 5 shows the results in terms of aggregate server performance. Variance is measured across clients. For all write experiments it is negligible. For the streaming read experiment the minimum client performance was 37 MB/s and the highest was 41 MB/s. For the random-access read experiment the minimum client performance was 109 IOPS and the highest was 137 IOPS.

Several observations can be made. First, in all cases, reads are faster than writes (by about 3x) because 3-way replication reduces the writes' "goodput" to one-third. Second, the servers' disks become the bottleneck for the random-access workloads. All numbers in those cases are close to the ideal. Third, we only get around a third of the expected bandwidth with streaming reads and writes. This is because of a well-known problem: lack of performance isolation between concurrent streams. Streaming accesses from one client interleave with accesses from

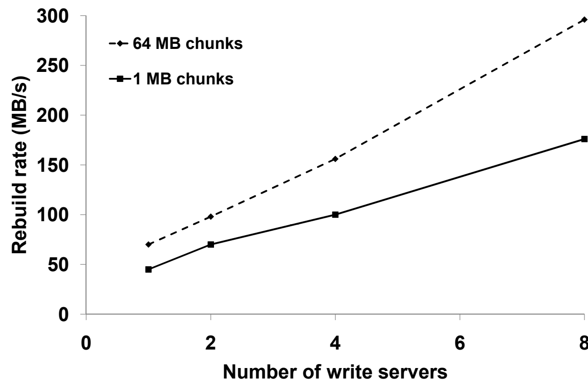


Figure 12: Rebuild rate

other clients and the combined accesses to the disk are not sequential anymore. A technique like Argon [14] would solve the problem, but we have not yet implemented it.

Recovery and layout: Choosing the correct layout is important for power savings, but also for recovery during permanent failures. Our scheme accommodates both needs. This next experiment shows recovery rates when we kill one server holding approximately 200 GB of data. We vary the number of write servers one can rebuild on. This serves to show the difference between naive grouping and power-aware grouping, first discussed in Section 3.3. With naive grouping, there would only be one write server, and the rebuild rate for the failed server's data correspondingly low. Figure 12 shows the results. As expected, the rebuild rate increases linearly with the number of servers.

We used a chunk size of 1 MB throughout the paper due to the need to accommodate the Hotmail I/O trace's capacity requirement on our testbed. Chunk sizes could be larger in practice, and we re-evaluated rebuild rates with a chunk size of 64 MB. As seen from Figure 12, the large chunk size improves the rebuild rate substantially. Although we are rebuilding 200 GB in both cases, larger chunks mean fewer per-chunk fixed costs associated with copying a chunk (e.g., fewer file creates, which are expensive). Larger chunks also reduce the performance interference mentioned earlier, between multiple chunks being rebuilt in parallel on the same server. In our experiments, we are able to increase rebuild rate linearly with the number of write nodes, at both 1 MB and 64 MB chunk sizes. With a sufficient number of write nodes, the network would become the bottleneck; due to the small scale of our testbed, the chunk servers remain the bottleneck.

5 Related Work

Section 3 described the non-power related previous work on which Sierra builds on: scalable distributed storage systems [1, 2, 5, 10] and our own previous work on short-term versioned stores [7, 8]. We also note that a large amount of work exists in saving power by turning off servers that are stateless or that have in-memory state that can be migrated. This allows consolidation of the CPU workload during troughs and is complementary to Sierra.

Here we contrast Sierra with previous work on saving energy in server storage. Sierra is designed for clusters of commodity servers, where "failure is a common case" and the system must keep data available despite server failures. Previous work is mostly aimed at saving power within RAID arrays attached to individual servers, and does not address the challenges of maintaining availability in a distributed system. A second key difference is that in Sierra I/O requests do not block waiting for a component to wake up from a low power state. This "spin-up wait" is a problem for many schemes based on entering low power states when idle. Powering entire servers down would extend this wait from seconds to minutes.

Write off-loading [7] exploits long periods of low, write-only load on RAID arrays on enterprise servers. During these periods, all the disks in the array are spun down, and the write load is consolidated onto versioned logs on a small number of active volumes, and reclaimed in the background when the original volume is spun up. However reads that go to a spun-down volume cannot be serviced until all the disks spin up again.

Popular Data Concentration (PDC) [9] exploits spatial rather than temporal workload properties, by periodically migrating hot data onto a small number of disks, and spinning down the cold disks. However, when the "cold" data is eventually accessed, the relevant disk must be spun up. PDC intentionally unbalances the load across disks, and a subset of the disks must now deliver the performance previously delivered by all of them. Hence, this is only applicable for disk arrays that are overprovisioned for performance, i.e., capacity-constrained ones.

Hibernator [16] adapts to varying loads by changing the rotational speed of disks in a RAID array. Data is kept available except while changing speeds. However this relies on multi-speed disks, which are not widely available today and they seem unlikely to enter widespread use.

Power-aware RAID (PARAID) [15] applies the notion of "gears" to RAID arrays. Data is simultaneously stored on multiple RAID arrays of different sizes, overlaid on the same set of disks. A RAID array with fewer disks is used when load is low, with the remaining disks being spun down. Data is kept available at the price of keeping one copy of it in each overlaid RAID array. This wastes both capacity and write bandwidth.

6 Future work and conclusion

In the short term, there is room for fine tuning the implementation of our protocols and improving performance (e.g., especially for handling performance insulation among concurrent streams). In the medium-term we plan to investigate the potential for more fine-grained gearing, i.e., turn individual servers on and off in response to load rather than entire gear groups. We also want to investigate the tradeoffs involved when erasure codes are used for redundancy instead of replication.

In addition, we believe our work on Sierra opens up three broad future research directions. First, it is worth examining ways of "filling the troughs" with more work, rather than powering servers down. For very large data centers the server cost is higher than the power cost [6] and improving server utilization could be more cost efficient than saving power. Troughs could be filled by running additional background tasks unrelated to the current services, e.g., by offering a generic utility computing service. Additionally, existing services could be modified so that their background maintenance tasks are more carefully scheduled.

Second, work is needed to align methods methods for consolidating computational tasks (e.g., virtualization) with the I/O load consolidation that Sierra offers. Removing both CPU and I/O load from a server allows the entire server to be powered down, rather than individual components. Ideally the system should also preserve locality while shifting gears, i.e., the co-location of computation with the data it computes on.

Third, more work is needed to achieve power-proportionality for optimistic concurrency control systems such as Amazon's Dynamo [4], which uses "sloppy quorums" to achieve a highly available, "eventually consistent" system. Similarly it would be interesting to extend the ideas from Sierra (which has a fail-stop failure model) to Byzantine fault-tolerant systems.

In conclusion, Sierra is, to the best of our knowledge, the first power-proportional distributed storage system. Achieving power-proportionality required maintaining similar consistency, fault-tolerance and availability as a default storage system, all while ensuring good performance. Live runs of a subset of a large-scale service, Hotmail, confirmed that achievable power savings match expected ones.

7 Acknowledgements

We thank Bruce Worthington and Swaroop Kavalanekar for the Hotmail I/O traces, Tom Harpel and Steve Lee for the Hotmail and Messenger load traces, Dennis Crain, Mac Manson and the MSR cluster folks for the hardware testbed and the IT facilities in Microsoft Research Cambridge for the Cambridge traces.

References

[1] Windows Azure Platform. `http://www.microsoft.com/azure`, Oct. 2009.

[2] M. Abd-El-Malek, W. V. Courtright II, C. Cranor, G. R. Ganger, J. Hendricks, A. J. Klosterman, M. Mesnier, M. Prasad, B. Salmon, R. R. Sambasivan, S. Sinnamohideen, J. D. Strunk, E. Thereska, M. Wachs, and J. J. Wylie. Ursa Minor: versatile cluster-based storage. In *Proc. USENIX Conference on File and Storage Technologies (FAST)*, Dec. 2005.

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

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

[5] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *ACM Symposium on Operating System Principles*, Lake George, NY, Oct. 2003.

[6] J. Hamilton. Resource consumption shaping. `http://perspectives.mvdirona.com/2008/12/17/ResourceConsumptionShaping.aspx`, Jan. 2008.

[7] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. In *Proc. USENIX Conference on File and Storage Technologies (FAST)*, San Jose, CA, Feb. 2008.

[8] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. Rowstron. Everest: Scaling down peak loads through I/O off-loading. Proc. Symposium on Operating Systems Design and Implementation (OSDI), 2008.

[9] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *Proc. Annual International Conference on Supercomputing (ICS'04)*, June 2004.

[10] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. FAB: building distributed enterprise disk arrays from commodity components. In *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2004.

[11] Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *CSURV: Computing Surveys*, 22, 1990.

[12] SNIA. IOTTA repository. `http://iotta.snia.org/`, Jan. 2009.

[13] R. van Renesse and F. B. Schneider. Chain replication for supporting high throughput and availability. In *OSDI*, pages 91–104, 2004.

[14] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger. Argon: performance insulation for shared storage servers. In *FAST*, 2007.

[15] C. Weddle, M. Oldham, J. Qian, A.-I. A. Wang, P. Reiher, and G. Kuenning. PARAID: The gear-shifting power-aware RAID. In *Proc. USENIX Conference on File and Storage Technologies (FAST'07)*, Feb. 2007.

[16] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: Helping disk arrays sleep through the winter. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, Brighton, United Kingdom, Oct. 2005.