Sierra: Practical Power-proportionality for Data Center Storage

Eno Thereska
Microsoft Research
etheres@microsoft.com

Austin Donnelly
Microsoft Research
austind@microsoft.com

Dushyanth Narayanan
Microsoft Research
dnarayan@microsoft.com

Abstract

Online services hosted in data centers show significant diurnal variation in load levels. Thus, there is significant potential for saving power by powering down excess servers during the troughs. However, while techniques like VM migration can consolidate computational load, storage state has always been the elephant in the room preventing this powering down. Migrating storage is not a practical way to consolidate I/O load.

This paper presents Sierra, a power-proportional distributed storage subsystem for data centers. Sierra allows powering down of a large fraction of servers during troughs without migrating data and without imposing extra capacity requirements. It addresses the challenges of maintaining read and write availability, no performance degradation, consistency, and fault tolerance for general I/O workloads through a set of techniques including power-aware layout, a distributed virtual log, recovery and migration techniques, and predictive gear scheduling. Replaying live traces from a large, real service (Hotmail) on a cluster shows power savings of 23%. Savings of 40–50% are possible with more complex optimizations.

Categories and Subject Descriptors C.2.4 [*Computer-communication networks*]: Distributed Systems

General Terms Design, Performance, Reliability

Keywords Data center, energy, power-proportionality

1. Introduction

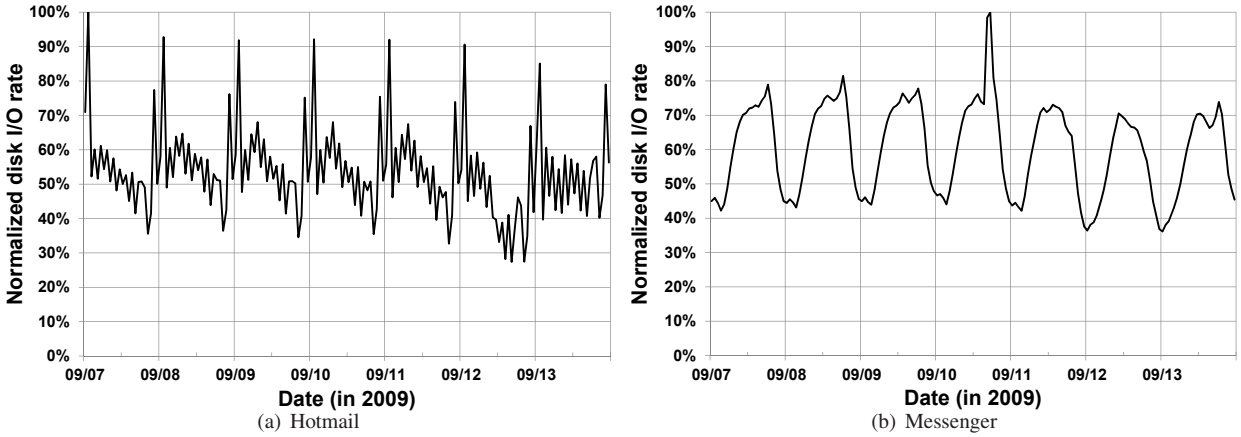
Server power consumption is a major problem for small and large data centers, since it adds substantially to an organization's power bills and carbon footprint. Barroso and Hölzle have argued for *power proportionality*: the power used should be proportional to the dynamic system load, even though the system is provisioned for the peak load [Barroso 2007]. Such a system could exploit diurnal variations in load, such as the "Pacific Ocean trough", seen in many user-facing services [Hamilton 2008].

Achieving power proportionality in hardware is hard, despite some support such as dynamic voltage scaling for CPUs. Servers continue to draw significant power even when idle, which can be up to 50% of the peak power consumption [Fan 2007]. A feasible alternative for large data centers is to turn off entire servers during troughs and consolidate load on the remaining servers. Virtualization techniques can transparently migrate computational state and network connections [Clark 2005]. However, storage presents a challenge. It is not feasible to migrate petabytes of storage several times a day for load consolidation (it takes too long).

This paper describes Sierra, a power-proportional distributed storage system. Sierra handles general workloads with read and writes (in contrast to recent work e.g., [Amur 2010] that only addresses read-only workloads) and Sierra supports in-place overwrites of data (this differs from append-only stores such as GFS [Ghemawat 2003]). Sierra exploits the redundancy (replication) in such systems to put the system in a lower "gear" during troughs by turning servers off. There are several challenges in doing this. To maintain read availability, at least one replica of each object must be on an active server always. This must be done without increasing the replication level and hence the system capacity requirements. For write availability, updates must succeed even when many servers are turned off. Durability, consistency, and fault-tolerance guarantees must be maintained on all data including the updates in low gear. Performance must not degrade: there should always be enough servers on to handle the load at any given time, and the load should be balanced across them. Powering servers down must not increase recovery time for transient or permanent server failures.

To our knowledge, Sierra is the first distributed storage system that meets all the above challenges and also provides power-proportionality. Our specific contributions are as follows. A new *power-aware layout* allows a significant fraction of the servers to be powered down without losing availability, load balancing, or fault tolerance. A novel use of a *distributed virtual log* allows updates to objects when some replicas are turned off. Such updates have the same

EuroSys'11, April 10–13, 2011, Salzburg, Austria.



(a) Hotmail (b) Messenger

Figure 1. One week of I/O load for two large online services

consistency, durability, and replication for fault tolerance as updates made when no servers are powered down. An *evaluation* of a full prototype running on a server cluster, using I/O traces from Hotmail, shows that significant power savings can be achieved with little performance impact.

2. Motivation, goals and non-goals

Sierra is based on the idea of *gear-shifting*: turning servers off and on to track diurnal variations in load. The savings achievable from this method depend on the average utilization of the system relative to the peak, and the predictability of load peaks and troughs. Figure 1(a) and 1(b) show the aggregated I/O load over time 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 disk bandwidth usage aggregated across all the back-end storage in the service at one-hour intervals and normalized to the peak value for each service (Section 4 will give more information about the services). We observe clear periodic patterns with significant peak-to-trough ratios.

Figure 2 shows the CDF of time spent at different utilization levels, again relative to the peak load. Substantial periods of time are spent with utilization much lower than the peak. The average utilization for Hotmail is 42% of its peak and that of Messenger is 60%. Thus, there is substantial scope for power savings through power proportionality. As an example, 50% savings in power for a 50,000 server data center could generate savings of $6 million/year assuming a power bill of $1 million/month [Hamilton 2009].

The primary scenario for Sierra, and the focus of this paper, is clusters of commodity servers where computation and storage are co-located on the same nodes. However, Sierra is also applicable to dedicated storage clusters. It is also relevant to SANs (storage area networks) but would require changing the internal behavior of the SAN. It is also

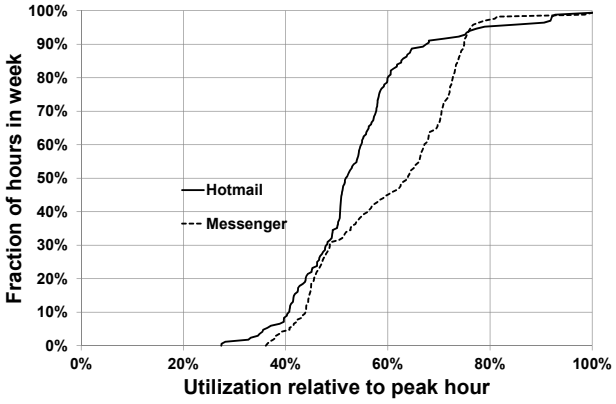


Figure 2. CDF of utilization for services

possible to use Sierra to spin down disks rather than turn off servers, e.g., if all server CPU resources are required.

This paper does not address migration and consolidation of computational and network load, since others have addressed them successfully (e.g., see [Chen 2008, Clark 2005]). In deployments where computation and storage are co-located, the selection of servers to turn off would need to be co-ordinated, for example between the VM migration layer and Sierra, to ensure that compute and I/O load are consolidated onto the same physical servers.

This paper does not address "trough-filling": eliminating troughs by filling them with useful work. Trough-filling by service consolidation was not effective for the online services we examined for two reasons. First, even when all users within a service are consolidated, we still see troughs (the Hotmail load graph captures all Hotmail users globally). Geo-distribution of these services will only lead to larger peak-to-trough ratios. Second, when consolidating across services, some are much larger and dominate (e.g., Hotmail has an order of magnitude more servers than Messenger). Filling troughs by selling the idle resources is an alternative for large cloud providers. Users' tasks could be run during

troughs whenever the users' bid price (described by [Amazon 2010, Stokely 2009]) exceeds the cost of powering the servers. This approach is complementary to Sierra.

3. Design and implementation

Sierra adopts existing best practices to provide scalable replicated distributed storage. This "baseline" architecture for Sierra is described in Section 3.1. It provides scalability, load balancing, high availability, fault tolerance, and strong read/write consistency. The challenges lie in powering down significant numbers of servers at low load without sacrificing the above properties.

The first challenge is *layout*: the assignment of data replicas to servers. The commonly used "naïve random" approach is simple, and provides high availability, good load balancing, and parallelism when rebuilding data after a failure. However, it does not allow significant power savings. Perhaps surprisingly, it is possible to lay out data such that significant numbers of servers can be turned off, while still maintaining good availability, load balancing and rebuild parallelism. Section 3.2 describes such a layout, used by Sierra. The layout is also optimal in that it provides the maximum possible power savings for any desired level of availability. Further, this is achieved without increasing the replication level of objects or over-provisioning the system.

The second challenge is to maintain read and write availability at the original levels. This must be done in low gear (while a significant number of servers are off), during gear transitions (servers are being powered up or down), and on failures. Section 3.3 describes the distributed virtual log (DVL) used by Sierra to keep the system available for writes. The DVL builds on the concept of write off-loading for RAID-based storage [Narayanan 2008]. We have extended it to support a distributed storage system, which introduces new challenges of network efficiency, availability, consistency, and fault tolerance. By solving these challenges we were able to considerably simplify the rest of the system and yet provide write availability and consistency at all times.

The third challenge is to predict the number of servers required at any time to sustain the system load, so that performance is not degraded. This is done by the *gear scheduler* component of Sierra, described in Section 3.5.

3.1 Baseline architecture

The baseline design of Sierra (Figure 3) is not novel, and resembles existing scalable distributed storage systems (e.g., see [Azu 2009, Abd-El-Malek 2005, Ghemawat 2003, Saito 2004]); we describe it here for context.

Sierra provides read/write access to objects in units of *chunks*. The chunk size is a system parameter: a typical value is 64 MiB. Each chunk is replicated on multiple *chunk servers*; a typical replication factor for data is 3. We use primary-secondary replication; in general, at any time, a chunk server will be the primary for some chunks and a secondary for others. Clients can read or write arbitrary byte ranges within a chunk. All client read and write requests are sent to the primary, which determines request ordering and ensures read/write consistency. Write requests are sent by the primaries to all replicas in parallel and acknowledged to the client when all replicas are updated. Read requests are served by primaries directly from the local replica.

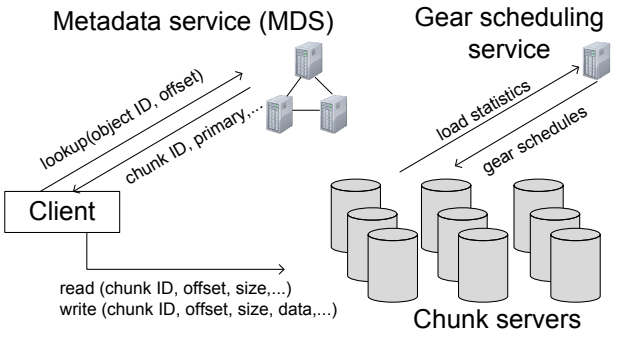
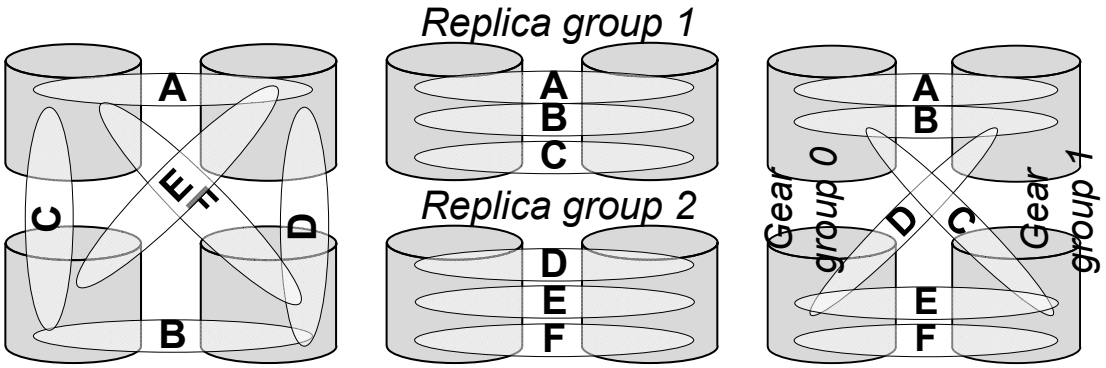


Figure 3. Baseline architecture for Sierra. We extend this architecture to support power-proportionality.

Sierra is intended to be a general-purpose storage system for use by a variety of applications. Hence, Sierra supports overwriting of existing chunk data; this differs from append-only stores such as GFS [Ghemawat 2003]. This allows Sierra to support a wider variety of workloads, but also introduces the challenge of supporting overwrites of chunks when one or more replicas are powered down.

A centralized metadata service (MDS) maps each object to its constituent chunks, and each chunk to its current primary. The MDS is an in-memory, deterministic state machine that could be replicated for high availability [Schneider 1990]. The MDS is not on the data path; clients cache metadata lookups from the MDS, and metadata is only updated on chunk creation or deletion and server failure, power-up, or power-down. The MDS grants leases to chunk primaries, which are periodically renewed through a heartbeat mechanism. If a lease expires it is reassigned by the MDS to a different replica of the chunk. Each such "view change" causes the MDS to increment a per-chunk *epoch*, which functions as a viewstamp [Oki 1988]. Requests from older epochs are rejected by chunk servers, ensuring that all replicas see a consistent ordering of client requests.

To reduce the metadata overheads, Sierra groups chunks into *chunk groups*. Leases and epochs are per chunk group rather than per chunk. All chunks in a chunk group are replicated on the same servers and have the same primary at any given time. 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. By default Sierra currently uses 64 chunk groups per chunk server. Load balancing in Sierra is done by uniform random assignment of chunks to chunk groups, chunk groups to servers, and primaries to replicas.



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

Figure 4. Different layouts for 6 chunks (A-F), 4 servers, and 2-way replication. The *naïve random* layout replicates each chunk on 2 servers chosen at random. This layout is excellent for load balancing and rebuild parallelism but leads to little or no power savings. The *naïve grouping* layout limits the location of chunks to replica groups (two in this case). A chunk must not be replicated across replica groups. This layout leads to great power savings but has poor rebuild parallelism. The *power-aware grouping* layout relaxes the constraint placed by *naïve grouping* and introduces a reasonable tradeoff between load balancing, rebuild parallelism, and power savings.

Individual chunks may experience different diurnal access patterns and loads, however chunk groups and chunk servers see the same aggregated load pattern.

Currently the system uses primary/secondary replication with 1-phase commit. This guarantees that all client requests are applied in the same order on all replicas. However a failure of a primary during a write request can result in some but not all replicas applying the write. On such a failure, the client library corrects the divergence by re-sending the write to the new primary, which uses the DVL to ensure that the write succeeds. Hence, unlike optimistic concurrency control systems (e.g., [DeCandia 2007]) there is no long period of divergence. Alternatively two-phase commit or chain replication could be used to prevent temporary replica divergence at the cost of higher update latencies [van Renesse 2004]. Any of these techniques are reasonable and would work well with the power-saving aspects of Sierra.

3.2 Power-aware layout

We find that layout, the way in which chunks are assigned to chunk servers, makes a large difference in how power-proportional a system can be. The most commonly used layout today is a *naïve random* layout or a variant of it, where each chunk is replicated on r servers chosen at random. Figure 4(a) shows a simple example with 4 servers, 6 chunks, and 2-way replication.

Our goal is a layout that can maintain g available copies of each chunk (out of a total of r, the replication level) using only $\frac{g}{r}$ of the servers. If data is evenly distributed over servers, then this is the minimum number of servers required to host $\frac{g}{r}$ of the data. However, the layout must ensure that this fraction corresponds to exactly g replicas of each chunk. We define a layout as optimal for power proportionality if it achieves this property for every integer g, $1 \leq g \leq r$. The value of g is now the *gear level* of the system at any time.

	Active servers	**Rebuild**	**Load distr.**
Naïve random	$N-(r-g)$	N	rC / N
Naïve grouping	$N\frac{g}{r}$	1	C / $\frac{N}{r}$
Power-aware	$N\frac{g}{r}$	$\frac{N}{r}$	C / $\frac{N}{r}$

Table 1. Active servers, rebuild parallelism, and load distribution for different layouts with N servers, C chunks, and r-way replication. g refers to the available copies of each chunk, or gear level ($1 \leq g \leq r$). A load distribution of rC / N means rC chunk replicas are uniformly distributed over N servers.

Unfortunately, the naïve random layout is far from optimal. In Figure 4(a)'s example, to keep 1 replica of every chunk available, we need 3 servers out of 4, rather than 2. As the number of servers and chunks increases, the minimum number of active servers required approaches $N-(r-g)$ rather than $N\frac{g}{r}$ where N is the total number of servers.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. This layout is optimally power-proportional, since $r-g$ servers in each replica group can be turned off in gear g. However, naïve grouping of servers suffers from a lack of *rebuild parallelism* because of the constraint on where replicas can reside. When a server suffers a permanent failure, intuitively there are fewer servers to rebuild *from* (in the illustrated example just 1 — this could lead to a read bottleneck) and fewer servers that one can rebuild *on*, which could lead to a write bottleneck.

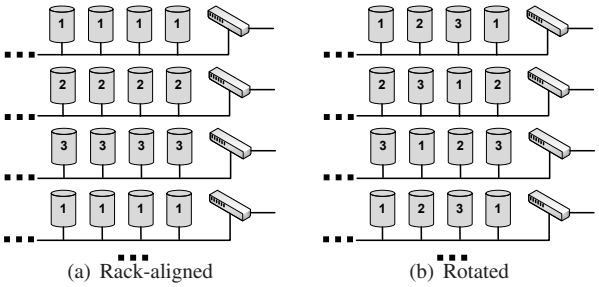


Figure 5. Two ways of assigning gear groups.

Sierra generalizes naïve grouping to achieve both power savings and high rebuild parallelism with *power-aware grouping* (Figure 4(c)). Each server is assigned to exactly one of r *gear groups*, each of size $\frac{N}{r}$. Each chunk is replicated once in each gear group. Now any g out of r gear groups can serve g replicas of each chunk. If a server fails, then its data can be rebuilt in parallel on all remaining servers in its gear group. Thus, the rebuild parallelism is $\frac{N}{r}$ where N is the total number of servers.

Table 1 summarizes the three approaches. The small examples in Figure 4 do not show this, but it is important to note that in realistic deployments, all three layouts will balance load well by distributing a large number of chunk replicas over a much smaller number of servers. Compared to the naïve random layout, the two power-proportional layouts spread $\frac{1}{r}$ as many replicas over $\frac{1}{r}$ as many resources. However, this is still a very large number: we are aiming at systems with millions of chunks spread across 1000s of servers. The same load balancing is maintained at lower gears.

For a large data center, servers might be further grouped into *clusters*. Different clusters could store different data, and be independently organized using power-aware grouping. Different clusters could also have different replication factors or coding schemes. E.g., small hot objects might be replicated with $r > 3$, and large cold objects might be erasure-coded. With an m-of-n erasure code, power-aware grouping allows $\frac{g}{n}$ of the servers to serve all data with a redundancy of $g - m$, $(m \leq g \leq n)$.

Smaller clusters reduce the dependencies between servers and hence the overhead of waking up sleeping replicas when an active replica fails. However, the cluster should be large enough so that rebuild speeds are limited by network bandwidth rather than server performance; with current hardware we expect each cluster to contain 100–1000 servers.

Sierra also places chunk replicas in different fault domains, i.e., racks. There are two options here for power-aware layout: rack-aligned and rotated (Figure 5). Both options give the same fault tolerance, power proportionality, and load balancing. They differ in the selection of servers to be turned off leading to different tradeoffs on power and thermal balance. In the rack-aligned case, all servers in a rack are in the same gear group and hence always in the same power

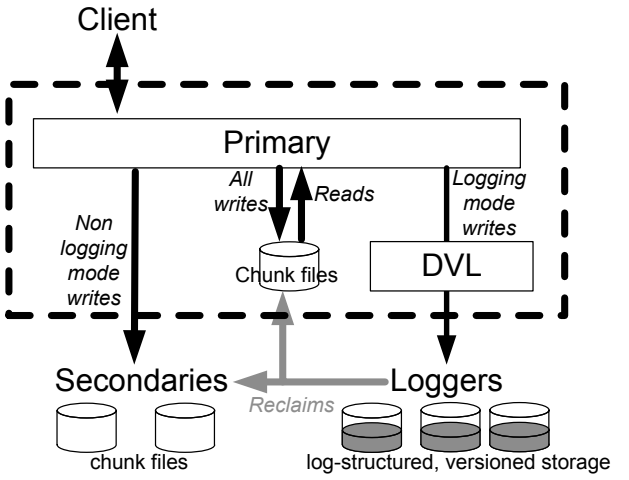


Figure 6. Data paths in logging and non-logging modes. The dotted box represents a single chunk server acting as a primary.

state. This permits entire racks to be turned off, allowing additional power savings by turning off rack-wide equipment such as switches. The rotated layout distributes the powered-up servers, and hence the thermal load, evenly across racks.

Our layout scheme does not add any additional complexity to data migration or adding and removing servers (or whole racks) to the system. As part of any data migration the constraint that the new location must be in the same gear group as the old location must be preserved.

3.3 Distributed virtual log

Each primary in Sierra is associated with one instance of a distributed virtual log or DVL (Figure 6). The DVL is used to reliably absorb updates to replicas that are on powered-down or failed servers. Hence, it is optimized for writes and for short-term storage. When one or more secondaries is unavailable (powered down or failed), a Sierra primary enters "logging mode". In this mode it sends writes to the primary replica and to the DVL but not to the secondaries. Data written to the DVL is replicated $r_L = r - 1$ times, guaranteeing a total of r replicas of all data. Along with the data, the DVL writes metadata including the chunk ID, the byte range written, a version number, and the primary's location. When all secondaries are available again, the primary enters "reclaim mode", where it scans the DVL and applies the deferred updates to the secondaries in the background. Once reclaimed, the data is deleted from the DVL.

No dedicated disk resources are assigned to a DVL; instead all DVLs share a common pool of *loggers*. Each logger uses a local on-disk log to store updates. Data from different DVLs is physically interleaved on the loggers but is kept logically separate. Write requests are load-balanced by the DVL across the set of available loggers. The DVL locally maintains a small in-memory map (recoverable from on-disk records at any time) that tracks the location of its data, and

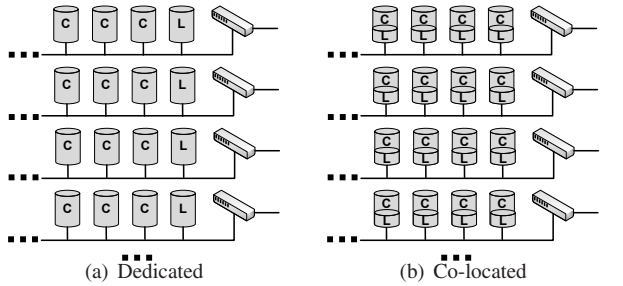


Figure 7. Two ways of configuring loggers (L) and chunk servers (C)

can also be queried by applications. Each DVL has a *logger view*, which is the set of all loggers available for use by it. This logger view is stored in the MDS, since it is small and infrequently updated.

The DVL provides the same semantics as a local on-disk log, even though successive writes and deletes can be sent to different loggers. These are:

- Write requests can be issued concurrently and are committed in FIFO order.
- Writes and deletions are durable when acknowledged.
- Reads always return the latest written data.
- Reads never return deleted data.

Write ordering is guaranteed by storing a monotonically increasing version (monotonic per-primary, not the whole cluster) with each write sent to the loggers. Correct deletion is guaranteed by writing a versioned, durable deletion marker. The deletion marker is deleted only when all older versions of the data have been deleted from all loggers.

Loggers can be run on dedicated servers or co-located with chunk servers as shown in Figure 7. The dedicated configuration minimizes contention between the chunk server workload and the logger workload, allowing the loggers to service mostly writes for which they are optimized. The dedicated configuration requires additional resources, but still provides significant total power savings. The provisioning method for both cases is described in Section 3.5.

3.3.1 Network-aware logging

Sierra minimizes the network overheads of logging and reclaim, especially the usage of scarce cross-rack (top-level switch) bandwidth. We use several optimizations to achieve this. First, the primary always serves reads from the local replica when the latest data is available there. This reduces disk and network load by avoiding demand reads on the DVLs, allowing the loggers to service mostly writes in logging mode and sequential scans in reclaim mode. A second network optimization is a direct data path from the loggers to the secondaries while reclaiming; for correctness, the control path always goes through the primary.

Third, the DVL uses a network-aware logger choice policy. In general, the DVL can send each update to any r_L loggers in its logger view. However, these loggers must be in different fault domains for fault tolerance. Also, if loggers are co-located with chunk servers, the r_L loggers must be in different gear groups. This ensures that turning off chunk servers while gearing down does not make logged data unavailable. Subject to these constraints, the logger choice policy minimizes both total network bandwidth and cross-rack bandwidth usage. For every write request, the DVL for a chunk group primary G considers its loggers in this order:

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

Within each of these groups, loggers are sorted by disk load. The DVL then greedily chooses the first r_L loggers in this ordering that satisfy the constraint of being in different fault domains from each other and from the primary. Typically this will result in one logger in the same rack as each secondary. Reclaiming is done by each replica transferring data from the logger closest to it. In rare cases the DVL gets demand read requests; these are sent to the logger(s) closest to the primary and holding the required data.

With these optimizations, logging of updates uses no more network bandwidth than normal replication without a DVL. Reclaiming uses a modest amount of additional in-rack bandwidth, but no cross-rack bandwidth is used except for control messages. Furthermore, reclaiming is a background process that is run at low priority.

3.3.2 DVL recovery and migration

The primary for a chunk group always holds a lease at the MDS for that chunk group. If the primary fails to renew the lease, the MDS initiates a view change with a new epoch number. It grants the lease to another replica of the chunk group, which becomes the new primary. Before accepting requests, the new primary must instantiate the correct local state of its DVL, i.e., the metadata which identifies the location and version of all logged data. This is done by requesting the metadata from all L loggers in the DVL's logger view, concurrently. Each logger first completes all outstanding requests for that DVL, and then atomically updates the epoch and returns the in-memory metadata for that DVL. The logger then does not accept any further requests from the old epoch. When $L - r_L + 1$ loggers have responded, the DVL's state is complete and the DVL can begin operation.

When a server is powering down rather than failed, we use an optimization that avoids running DVL recovery. Chunk group primaries on a server about to power down proactively migrate themselves. First, each primary completes any outstanding requests on the chunk group, while rejecting new ones (these will be retried by the client at the new primary). Then, it serializes its DVL state and sends it as part of a lease cancellation message to the MDS. The MDS forwards this state to the new primary as part of the lease grant message.

When all primaries have been migrated, a server S that is powering down sends a final "standby" message to the MDS and to all primaries for which it is a secondary. The MDS then marks S as "powered down". S's peers then use the DVL for updates rather than attempting to update S. When S wakes up from standby it resumes sending heartbeats to the MDS. The MDS then re-balances load by sending S a list of chunk groups to acquire leases on. S then contacts the current primary of each chunk group to initiate the migration.

3.4 Fault tolerance and availability

This section describes novel aspects of fault tolerance in Sierra. The lease reassignment protocol maintains read availability after a transient chunk server failure; this is standard and we do not discuss it further. Write availability on failure is maintained by using the DVL to absorb writes; the DVL was described in Section 3.3. After a permanent chunk server failure, its data is re-replicated in parallel on to other servers in the same gear group (Section 3.2). Here we consider other failure modes and their implications for Sierra.

Transient failures in low gear: When the MDS detects failure of a chunk server S, it wakes up all servers in the cluster holding other replicas of S's chunks. This takes a few seconds (if waking from standby) or minutes (if powering up) which does not significantly increase the window of vulnerability for two additional failures. However, when the system is already in the lowest gear, failure of a single server can cause the last active replica of a chunk to become temporarily unavailable during this wakeup time. This can be avoided by setting the *minimum gear level* g_{min} to 2. g_{min} is a policy parameter that trades power savings for a higher probability of unavailability on failure with typical values of 1 (for more power savings), and 2 (for higher availability).

Permanent failures in low gear: When a transient failure of a server S is detected, the MDS wakes up all sleeping replicas of S's chunks within a few minutes. Since detection of permanent failure usually takes longer than this, all replicas are available to begin rebuild as soon as permanent failure is detected. In any case, the rebuild time is much larger than the wakeup time and dominates the total recovery time. Hence, there is no significant increase in the window of vulnerability to additional failures.

Transient and permanent logger failures: When a server fails any logger on it also fails. However, replicas of each record on a failed logger are still available on $r_L - 1$ other servers. One option to maintain this fault tolerance is to re-replicate the at-risk data (log records with fewer than r_L available replicas) on other loggers. However, since the loggers are only intended for short-term storage, this results in wasted work. Instead, primaries first try to reclaim at-risk data at high priority, waking up secondaries as necessary. If a logger fails permanently it is deleted from the logger view of all DVLs. Before a new logger is added to a DVL's logger view, all data on it is deleted since it might be stale. This is done efficiently by writing a single "delete all" record to the log head.

Failures due to power cycling: We are not aware of any reliable data on the effect of power cycling on failures. However, Sierra minimizes the power cycling of servers in two ways. First, cycling is done at a coarse time granularity aimed at changing gears a few times per day. Second, the gear scheduler (Section 3.5) rotates the selection of gear groups for power-down over several days. This evens out the amount of power cycling per server (and also the amount of idle time for maintenance activities such as scrubbing.)

Garbage collection: When data is reclaimed from the DVL it is marked as deleted. The DVL does so through appending a versioned deletion marker to the loggers. The data and deletion markers must eventually be garbage collected from the DVL. When data in the DVL is overwritten, the older version becomes stale and becomes a candidate for garbage collection as well. Garbage collection of deletion markers must wait until all stale versions of the deleted data are removed. If the stale versions are on a failed logger, garbage collection of those deletion markers is blocked until the logger becomes unavailable or is deleted from the logger view. Deletion markers are small (i.e., they do not take up much space in the DVL) and this is not a problem in general. Similarly, during DVL recovery, garbage collection of all deletion markers is blocked until all loggers in the logger view have responded.

Garbage collection is a background maintenance process (it has background I/O priority). Write and delete requests are never blocked as long as r_L fault-uncorrelated loggers are available. The DVL is a best-effort service and is not meant to be used as a long-term file system. Hence, it is simpler than a general log-structured file system. In the worst-case that all loggers are unavailable (e.g., because a huge data burst has filled their capacity) and a write arrives that must be logged (e.g., because it is an overwrite of data already in the DVL), the write must block until the garbage collection frees up space in the DVL. In practice, past data should be used to provision the loggers with enough capacity to absorb data until it is reclaimed and such unavailability should be a very rare case. Section 11 provides some illustrative numbers on our logger capacity requirements.

3.5 Gear scheduler

The gear scheduler predicts system load and schedules servers to power down or up accordingly. It is a centralized component that periodically aggregates load measurements from chunk servers, and computes gear schedules for the future. Our gear scheduler is simple, because our workloads exhibit predictable patterns. It predicts load for each hour of the present day based on historical averages from that same hour in previous days. It is important to note that not all workloads will exhibit predictable patterns. For example, load will be higher than predicted for flash crowds and lower

than predicted for holidays. A hybrid scheduler based on historic averages and reacting to the current load is likely to be superior to ours for such cases. We opted to use a simple one because it worked well for us.

The load metric used by the gear scheduler is the overall I/O rate generated by all the clients of the storage system. Random-access I/O rate (measured in IOPS) and streaming I/O rate (measured in MiB/s) are considered separately. Reads and writes are also considered separately, and the write rate is multiplied by r to account for replication. The raw load metrics are measured every second at each primary, and periodically sent to the scheduler, which aggregates the load across the whole system. It then uses the peak values for each hour as the load metrics for that hour. We use the peak value since I/O load is often bursty and using the mean value can significantly degrade performance during bursts.

Given the measured performance per chunk server and the load, the scheduler computes the number of chunk servers required to sustain the system load in each hour:

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

$$N_{seq} = \frac{TotalMiB/s_{read}}{ServerMiB/s_{read}} + r \cdot \frac{TotalMiB/s_{write}}{ServerMiB/s_{write}}$$

$$N_{load} = \lceil \max(N_{nonseq}, N_{seq}) \rceil$$

This corresponds to a gear level $g = r\frac{N_{load}}{N}$; in general this will not be a whole number but have a fractional part. All servers in the lowest $\lfloor g \rfloor$ gear groups are left on, while all servers in the highest $r - \lceil g \rceil$ gear groups are turned off. This leaves up to 1 "fractional" gear group in which we turn off some servers (chosen at random) to achieve a total of N_{load} active servers. Alternatively the system can be configured for full rather than fractional gearing. In this mode, we power entire gear groups up and down, while always keeping at least N_{load} servers up. Hence, for full gearing the number of powered-up servers is $\lceil \frac{N\lceil g \rceil}{r} \rceil$ rather than $\lceil \frac{Ng}{r} \rceil$.

3.6 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 every chunk server running as a user-level process. A client-side library exports object *read()*, *write()*, *delete()* and *create()* calls. The core Sierra implementation (chunk servers, MDS, client library) is 11 kLOC. The DVL is 7.6 kLOC. There is an additional 17 kLOC of support code (RPC libraries, etc.) NTFS is used as the local file system on each chunk server and its lines of code are not included in the above measurements.

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.

3.7 Summary of tradeoffs and limitations

We summarize the tradeoffs described in this section and the limitations of our approach. First, we had to redesign the data layout to allow for the possibility of servers to be off. The main tradeoff involved power savings on one hand and rebuild speed and load distribution on the other hand. Because of this tradeoff our method works best for large clusters (hundreds of servers) and the effects of the tradeoff become more pronounced for small clusters.

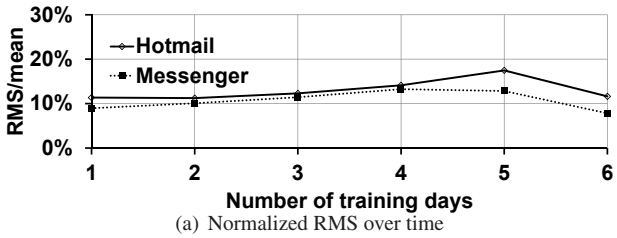
Second, we introduce a new service to the data center: a distributed virtual log (DVL). This service absorbs any writes that happen to a server that is powered down. Physically it is implemented as a short-term versioned store and it can reside on dedicated servers, or co-located with the existing chunk stores. The DVL would be simpler if the underlying chunk store file system were versioned, but ours (NTFS) is not. As such, the DVL adds some complexity in terms of lines of code and failure cases to the system. We hope it is still conceptually simple, since the notion of a "log" is well-understood and a distributed one is a rather natural extension for the data center environment.

Third, we described new failure scenarios that can be handled without introducing any new tradeoffs, and one failure scenario that introduces a tradeoff between power savings and availability. Specifically, when the system is already in the lowest gear, failure of a single server can cause the last active replica of a chunk to become temporarily unavailable during this wakeup time (a window of seconds to minutes). Depending on the workload characteristics and server failure profiles, some deployments might choose never to go to the lowest gear to avoid this tradeoff.

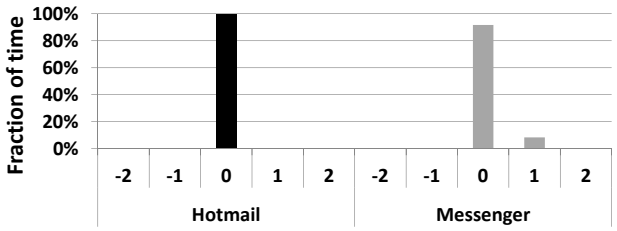
Fourth, the gear scheduler allows a spectrum of tradeoffs that can be explored (e.g., predictive vs. reactive gear selection policies, gearing timescales in the order of minutes and hours, and full vs. fractional gearing.) Several of these tradeoffs are shown in Figure 9(b). These are tradeoffs among power savings, complexity of workload characterization and ability of servers to rapidly switch on and off. We chose a simple default mode that works well for our workloads.

4. Evaluation

In this section we answer the following questions. First, does our *gear scheduler* work well? Second, what are the *power savings*? Third, what is the impact on *performance*? Finally, how does *rebuild* performance scale with the number of servers, for the Sierra layout. Our evaluation is driven by two sets of traces as well as microbenchmarks. We have obtained *load traces* from Hotmail (Windows Live Mail) and Windows Live Messenger. These are measurements of I/O load at a 1-hour granularity for 1 week, aggregated over thousands of servers as shown in Figure 1. The load traces are used to measure how well the gear scheduler works and expected power savings.



(a) Normalized RMS over time



(b) Distribution of gear selection error

Figure 8. Prediction accuracy

The load traces are not sufficiently fine-grained to measure performance on real hardware. For this, we have obtained a small set of *I/O traces* which contain I/O requests to e-mail messages stored on 8 Hotmail back-end servers over a 48-hour period, starting at midnight (PDT) on Monday, August 4, 2008. These I/O traces are from a different time period than the Hotmail load traces, but represent the same service. The I/O traces are taken at the block device level, i.e., below the main memory buffer cache but above the storage hardware (RAID). We map them onto our object-based file system as detailed in Section 4.3. The overall read/write ratio of these traces is 1.3 (i.e. slightly more reads). Approximately 276 GiB of data were written and 460 GiB of data were read from these servers during that time. The diurnal pattern for these traces is shown in Figure 10.

4.1 Gear scheduler

We applied the simple "hour of day" load prediction algorithm (see Section 3.5) to the Hotmail and Messenger load traces. The load metric used was mean bytes transferred per hour, since that is all we have available from this data. For each workload, we have 7 days of data. At the end of each day, we train the predictor using all the previous days' data, with equal weighting for all past days. We then test the predictor on the remaining days in the trace. The aim is to find how quickly, and to what value, the error converges.

We measure error taking the root-mean-square (RMS) of the difference between predicted and actual load. We normalize this by the mean load during the test period to give a scale-free measure of error. Figure 8(a) shows how the error changes over time. For both Hotmail and Messenger, the error is low (around 10% of the mean) even after a single day of training and does not change significantly afterward. Hence, even with this simple predictive method, the number of active servers when using fractional gearing will always be within 10% of the ideal.

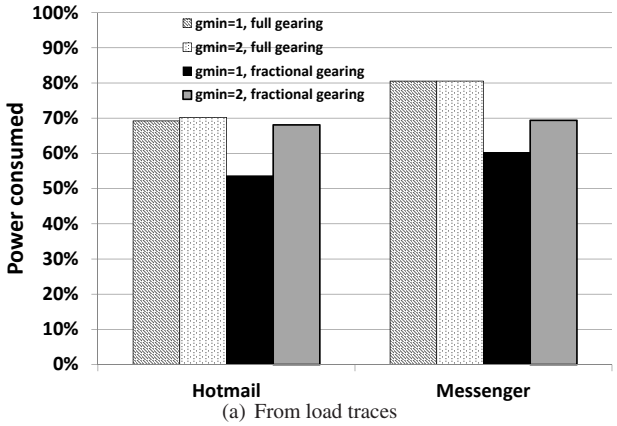
When using full gearing, we can also measure how often the predictor selects the wrong gear. For 3-way replication, the difference between the correct gear for the load, and the chosen gear based on predicted load, can range from -2 to +2, since gear values range from 1 to 3. Figure 8(b) shows how load prediction errors translate to gear selection errors in this case. The histograms show the frequency of occurrence of each value; perfect prediction would lead to a single bar centered on zero. With Hotmail we achieve this, i.e., the system always selects the correct gear, and for Messenger it selects the correct gear 90% of the time.

4.2 Power

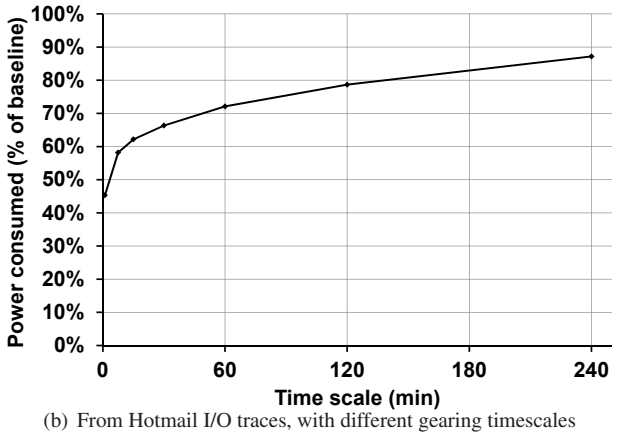
Figure 9(a) shows the analytical estimation of power consumption of Sierra for the Hotmail and Messenger load traces as a fraction of the baseline system. The estimation is based on the number of servers expected to be turned off over the week. To separate the effects of prediction and power savings, here we use an "oracle" predictor that always predicts the correct gear, i.e., we correct the 10% misprediction that occurs in Messenger (an incorrect gear selection could either save more power — and have bad performance — or could save less power than a correct one). The loggers are assumed to be co-located with the servers. We are not allowed to reveal the exact number of servers in the baseline system, but it is all the Hotmail and Messenger servers (several thousands) as first described in Section 2.

We show the power savings for both fractional and full gearing, and for two different settings of the minimum gear value: $g_{min} = 1$ and $g_{min} = 2$. Setting g_{min} to 2 increases power consumption very slightly for Hotmail, and more for Messenger. This is because, for Hotmail, the system stays above gear 2 most of the time for load reasons, even when $g_{min} = 1$. We also see that fractional gearing, by being finer-grained, achieves significant power savings compared to full gearing, as expected from the analysis in Section 3.5.

The results so far used gearing based on mean load per hour, and with a maximum gear-shift frequency of once per hour, since this is the finest granularity obtainable from the load traces. However, from the Hotmail I/O traces we are able to measure the effect of using the peak load metric, and also of shifting gears more frequently. Figure 9(b) shows the analytical estimate of power consumption for the 2-day Hotmail I/O traces, using the peak load metric with the oracle predictor, full gearing and $g_{min} = 1$, and varying the timescale of prediction. At the 1-hour timescale, the power consumption is 72% of the baseline system compared to 69% predicted from the load traces. Higher power savings seem possible by gear-shifting frequently, e.g., once per minute; however this requires frequent power state cycling and also accurate fine-grained load prediction. It is not clear whether such fine-grained prediction is possible: the fine-grained traces only cover 48 hours which is insufficient



(a) From load traces



(b) From Hotmail I/O traces, with different gearing timescales

Figure 9. Power consumption

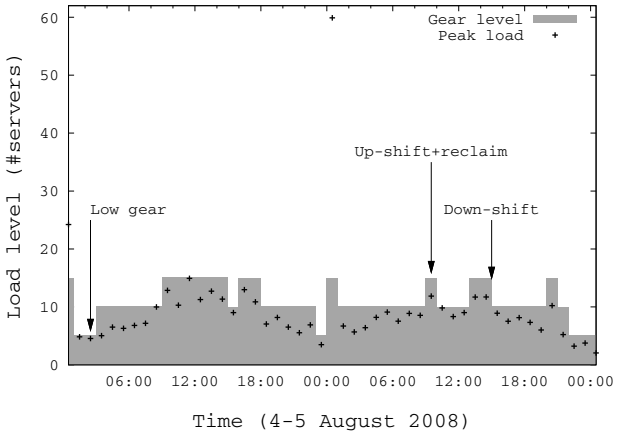


Figure 10. Gear schedule for Hotmail I/O trace

to evaluate this. All experiments in this paper use a 1-hour granularity as the 7-day load traces show good predictability at this granularity.

4.3 Performance on I/O traces

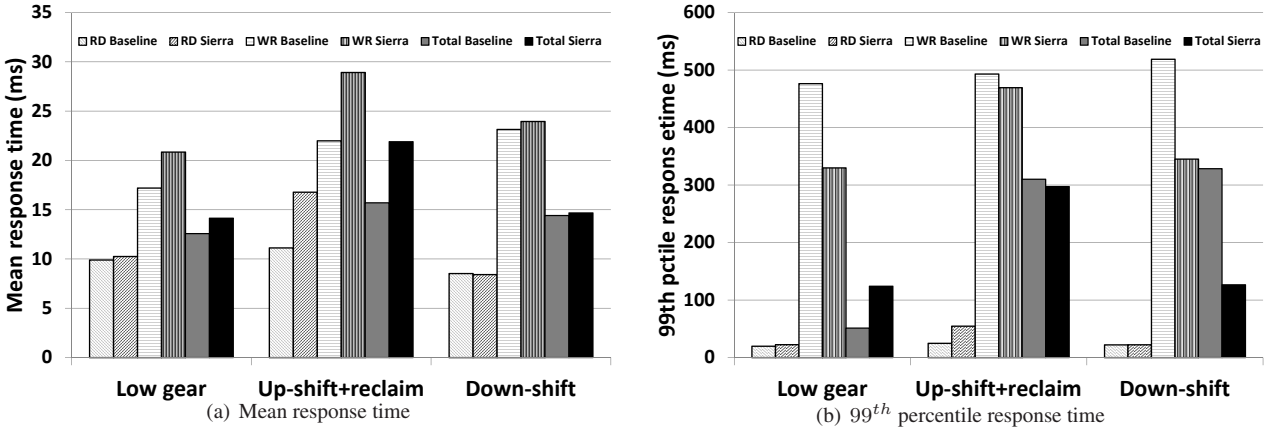
Our performance evaluation is based on replaying the Hotmail I/O traces on a Sierra system deployed on a small scale cluster. Our experimental testbed consists of 31 identical servers in 3 racks. Each rack has a Cisco Catalyst 3750E as a Top-of-Rack (ToR) switch providing 1 Gbps ports for the servers, 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 GiB of RAM, a 1 TB system disk and a 1 TB disk that holds the Sierra chunk files and log files. Each server runs Windows Server 2008 Enterprise Edition, SP1.

Trace replay: We map the Hotmail I/O traces to Sierra objects 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>*. The traces are then split by virtual disk and replayed from multiple client machines. Although both client and server 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. We also disable prefetching and write-back for the same reason.

Provisioning: For meaningful experimental results it is important to correctly provision the system for the workload, i.e., chose the correct number of chunk servers. Over-provisioning the system would increase the baseline system's power consumption unnecessarily, and thereby inflate the relative power savings of Sierra. Under-provisioning the system would also be meaningless because the system could not sustain the peak load even with all servers powered up. To calculate the number of chunk servers needed to support a workload, we use the methodology based on the peak load metric described in Section 3.5. Provisioning was based on measured single-server performance (see Section 4.4 for a description of the microbenchmarks used).

In addition to performance, we must also match the availability and capacity requirements of the workload. For availability, we replicate chunks on servers in three different fault domains, i.e., racks. For capacity, we are limited by the total storage capacity of our servers, which is smaller than 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 MiB. Hence, for each experiment, we pre-create the chunks that are accessed during the experiment, using a chunk size of 1 MiB. In practice a larger chunk size, e.g., 64 MiB is more common [Ghemawat 2003].

For the Hotmail I/O 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



(a) Mean response time

(b) 99^{th} percentile response time

Figure 11. Performance comparison for Hotmail I/O trace. The "transition" experiment is for a downward gear shift.

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 the Sierra configuration, all 3 loggers are always left powered up to ensure that 3-way, cross-rack replication is always possible. Due to the small size of our testbed, the power cost of the dedicated loggers is not amortized well and their power overhead is 6% over that expected from Section 4.2 (which looked only at chunk server power savings).

We did not have access to a power measurement kit (i.e., the servers did not have a watt-meter attached) and furthermore the servers were in a remote datacenter and we did not have permission to turn them off. To emulate a server being off we instruct the RPC layer of that server to drop all received packets silently. This approach has the shortcoming that it does not capture the time it takes to power down and up a server, however we expect those times to be negligible compared to the time a server stays off (at least 1 hour).

Trace segment selection: For our performance experiments we selected three trace segments that represent the worst case for three different aspects of Sierra. First we chose the 1-hour period when the system was in the lowest gear, and saw the highest peak load of all such periods. The aim is to show the worst-case performance impact of putting the system in the lowest gear. Second, we chose the transition into highest gear when the amount of logged data in the DVL was the highest, and the 1-hour period following the transition. The aim here is to show the effect of an up shift (i.e., turning servers on and rebalancing load) as well as the performance impact of reclaiming logged data to the chunk servers. Finally, we chose the downward transition having the highest load in the minute immediately following the transition, and replayed the trace from the transition until the end of the minute. The aim here is to show the worst-case impact of turning servers off. Figure 10 shows the load metric for each hour of the Hotmail I/O trace, the gear chosen by the oracle predictor for each hour, and the periods corresponding to the three experiments[1]. All the experiments are based on full gearing, which gives the largest possible gear transitions; the lowest and highest gear levels are the same for both full and fractional gearing. In all experiments we pre-create the necessary system state, including logger state, by first replaying the relevant preceding portions of the trace.

Request response times: Figure 11 shows the mean and 99^{th} percentile (note the different scales on the two y-axes) of performance of the baseline and Sierra configurations during the three experiments. We show the performance of read and write requests separately as well as the total performance. We make several observations. First, given that these are the worst three scenarios, the performance penalty for Sierra is small. Second, the low gear experiment shows that our provisioning and gear selection methodology is reasonable; the performance of Sierra with 2/3 of the chunk servers turned off is comparable to that of the baseline, i.e., all servers turned on. Third, the performance penalty is significantly smaller for the 99^{th} percentile, showing that Sierra does not make the slow requests slower (Sierra writes are sometimes slightly faster than in the baseline system because they are sent to loggers which are optimized for writes.)

The performance penalty is slightly more pronounced for the up-shift/reclaim experiment. This is due to performance interference between the foreground workload and the reclaim process, which is faster than it needs to be. Table 2 shows this long-term reclaim rate required for this workload as well as the actual reclaim rate achieved during the up-shift/reclaim experiment. The long-term reclaim rate is obtained by measuring the number of unique bytes logged just before each up-shift in the 48-hour period, and summing

[1] The "peak" load in the Hotmail I/O traces happens just after midnight during a 2-hour period of background maintenance. Provisioning the system for the maintenance peak would only improve the I/O response time of background tasks. It would also artificially increase the baseline power consumption and hence improve Sierra's relative power savings. To avoid this effect, we exclude this maintenance period when provisioning the system, and keep the system in the highest gear during the window.

Total data logged in 48 hrs	166 GiB
Time in top gear	14 hrs
Required reclaim rate	3.37 MiB/s
Data reclaimed in 1 hour after up-shift	27 GiB
Reclaim rate achieved	7.7 MiB/s

Table 2. Reclaim statistics

# primaries migrated in down-shift	98
Total migration time	1.3 s
Total # of requests	11532
# of requests retrying	118
Average/worst retry penalty	9.8 / 74 ms
# primaries migrated in up-shift	116
Total migration time	2.4 s
Total # requests	18618
# number of requests retrying	152
Average/worst retry penalty	224 / 2168 ms

Table 3. Primary migration statistics

these values. This gives an upper bound on the amount of data that needs to be reclaimed for those 48 hours. Although all reclaim requests have background priority, we believe a better control mechanism [Douceur 1999] could reduce the rate to the long-term average. We note that, for this setup, provisioning the loggers with 30 GiB of capacity is sufficient to ensure all writes are absorbed without blocking.

Details on primary migration: Chunk group primaries are migrated during down-shifts to maintain availability and during up-shifts to rebalance load. After a primary migrates, the first request from each client for that chunk group will go to the old primary, fail, and be retried after re-fetching metadata from the MDS. Subsequently it is cached at the client and future requests do not pay the retry penalty.

Table 3 summarizes the migration statistics for the down-shift and the up-shift experiments. The total migration time is dominated by the time for the busiest primary to drain its queue (i.e., complete outstanding requests) before the new primary can begin accepting requests. This also determines the worst-case retry penalty. We could optimize this by waiting for primaries to have short queues before migrating them, rather than starting them at a fixed time dictated by the gear scheduler. However, given that so few requests are affected we have not prioritized this optimization.

Metadata state size: We measured the amount of metadata for the longest experiment, the up-shift one. The metadata service had about 100 MiB of in-memory state, corresponding to 4.6 million chunks in 320 chunk groups. This is a memory requirement of 23 bytes per chunk or 320 KiB per chunk group.

4.4 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 MiB/s using 64 KiB reads and writes to a 2.4 GiB object (system chunk size is the default of 1 MiB). Second, we measure random-access read and write performance in IOPS (I/Os per second) by sending 100,000 I/Os to the server. The client keeps 64 requests outstanding at all times. Table 4 shows the results. Write have more variable performance than reads due to inherent properties of NTFS.

	Writes	Reads
Bandwidth (MiB/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.

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, but r is 3 in this case) 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 client performance was 37–41 MiB/s. For the random-access read experiment the client performance was 109–137 IOPS.

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

Table 5. Peak system 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.

We observe that in all cases, write performance is about a third of read performance as expected because of 3-way replication. For the random-access workloads the servers' disks become the bottleneck. All numbers in those cases are close to the ideal. For streaming reads and writes, Sierra gets only around a third of the expected bandwidth. This is because of a well-known problem: lack of performance isolation between concurrent streams. A mechanism like Argon [Wachs 2007] would potentially be beneficial in our case but we have not explored it.

Rebuild speed: Sierra's power-aware layout provides not only power savings but also rebuild parallelism when a server fails permanently and its data must be rebuilt. To

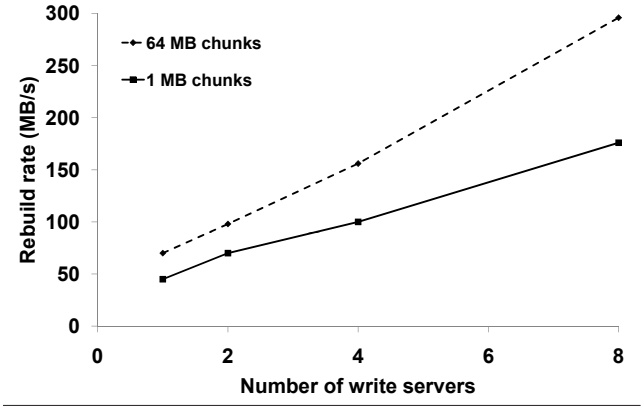


Figure 12. Rebuild rate as a function of cluster size

show the effect of this parallelism we ran a microbenchmark measuring rebuild times while varying the cluster size N, which determines the rebuild parallelism. In all cases we used 3-way replication, with data being read from $\frac{2N}{3}$ nodes and written in parallel to $\frac{N}{3}$ nodes. The write servers (being fewer) are the bottleneck rather than the read servers, and hence, Figure 12 shows the rebuild rate as a function of the number of write servers.

Rebuild scales with the number of servers, showing the importance of rebuild parallelism. Note that the single-server performance is lower than the raw disk bandwidth due to chunk creates (i.e., NTFS "create" system call) being the bottleneck. We show performance for a small 1 MiB chunk size (which was selected to accommodate the Hotmail I/O trace's capacity requirement on our testbed), and for a larger 64 MiB chunk size that amortizes better the overhead of the chunk creates. With 8 write servers and 64 MiB chunks, and no extra network optimizations, Sierra achieves a reasonable rebuild performance of 296 MiB/s (1 hour per TB rebuilt, or 2.3 Gbps of cross-rack network bandwidth usage).

5. Related Work

Section 3 described the (non power-proportional) storage systems that Sierra builds on [Azu 2009, Abd-El-Malek 2005, Ghemawat 2003, Saito 2004]. Here we contrast Sierra with related work on power savings.

The DVL component of Sierra builds on write off-loading [Narayanan 2008]. However, that work focused on traditional RAID-based storage on individual servers. Other previous work such as PARAID [Weddle 2007] (which first introduces the notion of gears in this context) and power-aware storage caching [Zhu 2005] are also based on enterprise storage. This previous work does not address the challenges of large data centers: providing power proportionality in while maintaining scalability, availability, consistency, and fault tolerance. Specifically, the Sierra DVL extends the previous work on write off-loading with network awareness, fast failure recovery, and asynchronous primary migration.

Work has been done on power-proportionality for read workloads [Amur 2010]. Sierra handles general workloads with read and writes. It is non-trivial to support writes while maintaining availability, fault tolerance and consistency. Achieving this through the design and implementation of the DVL is one of our key technical contributions. Some approaches rely on increasing the replication factor r, and hence the capacity requirements of the system, to achieve power proportionality [Amur 2010, Weddle 2007]. In addition to increasing capacity, these techniques come at the cost of write performance — more copies need to be updated on each write. This can result in more servers being provisioned for the baseline system, increasing its power consumption. Sierra achieves power-proportionality without increasing the replication factor of the original system.

Popular Data Concentration (PDC) exploits spatial locality by migrating hot data onto a small number of "hot" disks, and spinning down the cold disks [Pinheiro 2004]. However, when the cold data is accessed, there is a large latency in waiting for a disk to spin up. PDC also unbalances the load across disks, which means that the system must be over-provisioned for performance, which increases the baseline power consumption. Other work on analyzing effects of data layout targets availability [Yu 2007]. Sierra's layout builds on that body of work and shows surprising effects of layout on power-proportionality.

Current hardware-based approaches, such as CPU voltage scaling and multi-speed disks [Zhu 2005], do not offer a wide enough range of power consumption scaling. Sierra offers power proportionality through software, by turning off entire servers, without requiring specific power-proportionality support in hardware.

6. Conclusion and future work

Sierra is, to the best of our knowledge, the first power-proportional distributed storage system for general read and write workloads. It achieves power-proportionality in software while maintaining the consistency, fault-tolerance and availability of the baseline system. Analysis of load traces from two large online services show significant power savings from exploiting diurnal load patterns, and a performance evaluation on a small server cluster shows a modest overhead in I/O response time.

We have identified several directions for future work. First, work is needed to align methods for consolidating computational tasks (e.g., virtualization) with the I/O load consolidation that Sierra offers. Ideally the system should also preserve locality while shifting gears, i.e., the co-location of computation with the data it computes on. Second, more thinking is needed to achieve power-proportionality for optimistic concurrency control systems such as Amazon's Dynamo, which uses "sloppy quorums" [DeCandia 2007] and for Byzantine fault-tolerant systems.

7. Acknowledgments

We thank our shepherd Alistair Veitch, the anonymous reviewers, and Ant Rowstron, James Hamilton, Miguel Castro, Zhe Zhang and Paul Barham for their feedback. 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.

References

[Azu 2009] Windows Azure Platform, October 2009. URL `http://www.microsoft.com/azure`.

[Abd-El-Malek 2005] Michael Abd-El-Malek, William V. Courtright II, Chuck Cranor, Gregory R. Ganger, James Hendricks, Andrew J. Klosterman, Michael Mesnier, Manish Prasad, Brandon Salmon, Raja R. Sambasivan, Shafeeq Sinnamohideen, John D. Strunk, Eno Thereska, Matthew Wachs, and Jay J. Wylie. Ursa Minor: versatile cluster-based storage. In *Proc. USENIX Conference on File and Storage Technologies (FAST)*, December 2005.

[Amazon 2010] Amazon. Amazon EC2 spot instances, April 2010. URL `http://aws.amazon.com/ec2/spot-instances/`.

[Amur 2010] Hrishikesh Amur, James Cipar, Varun Gupta, Michael Kozuch, Gregory Ganger, and Karsten Schwan. Robust and flexible power-proportional storage. In *Proc. Symposium on Cloud Computing (SOCC)*, Indianapolis, IN, June 2010.

[Barroso 2007] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[Chen 2008] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, 2008.

[Clark 2005] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.

[DeCandia 2007] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, October 2007.

[Douceur 1999] John R. Douceur and William J. Bolosky. Progress-based regulation of low-importance processes. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, Kiawah Island, SC, December 1999.

[Fan 2007] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proc. International Symposium on Computer architecture (ISCA)*, San Diego, CA, 2007.

[Ghemawat 2003] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proc. ACM Symposium on Operating System Principles (SOSP)*, Lake George, NY, October 2003.

[Hamilton 2008] James Hamilton. Resource consumption shaping, 2008. URL `http://perspectives.mvdirona.com/`.

[Hamilton 2009] James Hamilton. Cost of power in large-scale data centers, 2009. URL `http://perspectives.mvdirona.com/`.

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

[Oki 1988] Brian M. Oki and Barbara Liskov. Viewstamped replication: A general primary copy method to support highly-available distributed systems. In *Proc. Symposium on Principles of Distributed Computing (PODC)*, Toronto, Canada, August 1988.

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

[Saito 2004] Yasushi Saito, Svend Frølund, Alistair Veitch, Arif Merchant, and Susan Spence. FAB: building distributed enterprise disk arrays from commodity components. In *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 2004.

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

[Stokely 2009] Murray Stokely, Jim Winget, Ed Keyes, Carrie Grimes, and Benjamin Yolken. Using a market economy to provision compute resources across planet-wide clusters. In *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–8, Rome, Italy, May 2009.

[van Renesse 2004] Robbert van Renesse and Fred B. Schneider. Chain replication for supporting high throughput and availability. In *Proc. Symposium on Operating Systems Design and Implementation (OSDI)*, pages 91–104, 2004.

[Wachs 2007] Matthew Wachs, Michael Abd-El-Malek, Eno Thereska, and Gregory R. Ganger. Argon: performance insulation for shared storage servers. In *Proc. USENIX Conference on File and Storage Technologies (FAST)*, 2007.

[Weddle 2007] Charles Weddle, Mathew Oldham, Jin Qian, An-I Andy Wang, Peter Reiher, and Geuff Kuenning. PARAID: The gear-shifting power-aware RAID. In *Proc. USENIX Conference on File and Storage Technologies (FAST'07)*, February 2007.

[Yu 2007] Haifeng Yu and Phillip B. Gibbons. Optimal inter-object correlation when replicating for availability. In *Proc. Symposium on Principles of Distributed Computing (PODC)*, Portland, OR, August 2007.

[Zhu 2005] 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, October 2005.