SLIM: Network Decongestion for Storage Systems

Madalin Mihailescu, Gokul Soundararajan†, Cristiana Amza†
Department of Computer Science
Department of Electrical and Computer Engineering †
University of Toronto

ABSTRACT

We introduce SLIM, a hybrid storage system that uses disks within data center racks to implement persistent caches for storage area networks (SANs). SLIM leverages compression and block overwrites to reduce traffic on data center oversubscribed network links and to improve performance for low-cost SANs. We evaluate SLIM using various micro-benchmarks and industry standard benchmarks. Our results show that SLIM reduces network storage traffic by 40%-90% and significantly increases application performance in bandwidth-constrained environments.

1. INTRODUCTION

Modern data centers consist of multiple racks of physical servers, interconnected through a networking fabric built using commodity components. Each physical server hosts multiple virtual machines (VMs) using storage area networks (SANs) to reduce operational costs. SANs can be implemented by using either dedicated networking hardware such as Fibre Channel-based solutions or by sharing the existing server communication networking fabric with schemes based on iSCSI/Ethernet. Deployments using Fibre Channel offer high performance at a high cost. Fibre Channel solutions require a separate and dedicated infrastructure, adding significantly to the infrastructure cost. Therefore, SAN implementations that share the existing networking fabric are more prevalent in large data centers.

However, current data center networks are built hierarchically [3], with servers directly connected to a rack-level switch (RS). Rack-level switches are in turn connected to a layer of cluster-level switches (CS) through a number of network links. The rack-to-cluster links are *oversubscribed*, often by a factor of 10, thus creating a situation where communication external to a rack is more expensive than intra-rack communication. Hence, with higher degrees of server consolidation, the aggregate traffic from the servers/racks *overwhelms* the capacity of the network fabric, thereby limiting scalability. We thus focus on improving the performance of the more prevalent low-cost implementations, i.e., those built using commodity components such as iSCSI/Ethernet.

In this work, we propose SLIM, a *hybrid* storage solution that uses the disks located within each rack to offload remote network storage traffic. The *key* insight that SLIM leverages is to (i) use the disks on each server to build a reliable *local storage* within each rack, acting as a *persistent data cache*, and (ii) use the local storage to implement several optimizations, i.e., compression and batching, to reduce the SAN traffic on the performance critical rack-to-cluster network links.

Three factors influence the design of SLIM: (i) the abundance of CPU capacity, (ii) the availability of disks within the rack, and (iii) the capacity of intra-rack network bandwidth. First, we notice that, with the prevalence of multi-core architectures, there is an abundance of computation power in the physical servers. Second, we notice that disks on the physical servers are under-utilized, as the VMs access data hosted on the SAN. Third, in bandwidth constrained scenarios, there is substantially more bandwidth available within the rack, compared to the rack-to-cluster links.

With these observations, we build a distributed storage using the disks local to each rack to offload remote I/O writes – that is, we create a *persistent write-back cache* by pooling the under-utilized disks available within the rack. By caching writes locally, SLIM performs several I/O optimizations to achieve its main objective of decreasing the amount of bytes that exit a rack switch and reach the SAN. First, data is compressed prior to migration. While compression techniques are offered as options at various levels – application, file system, disk image [1, 2], they are performed on the critical path, rendering them expensive. By storing data temporarily in the rack cache, SLIM can compress data outside the critical path, without impacting performance. Data is stored compressed in the SAN and decompressed on the fly on I/O reads. Second, recent work [11] has shown that I/O overwrites, i.e., I/O writes to the same block, account for a significant fraction in many workloads. We use this observation in our system. SLIM controls the time data is stored in the rack cache, to take advantage of I/O overwrite patterns and further reduce the amount of bytes transferred over the network.

We implemented a prototype of SLIM using the Network Block Device (NBD) protocol. In our preliminary evaluation, we used TPC-C, a write intensive OLTP benchmark, and TPC-H, a read intensive decision support benchmark. As TPC-C has a high I/O overwrite ratio, SLIM was able to reduce network storage traffic by 40%-90%, across various migration time interval values. For TPC-H, since SLIM stores data compressed in the network storage, the amount

This paper appeared at the Second Workshop on I/O Virtualization (WIOV '10), March 13, 2010, Pittsburgh, PA, USA.

of bytes transferred was reduced by 50% on average. As a secondary benefit, SLIM improved application performance for oversubscribed networks, and the improvements were higher as we increased the oversubscription factor. For TPC-C, SLIM increased application throughput by up to 250%, while for TPC-H, SLIM decreased latency by up to 42%.

The next section describes our problem setting in more detail. The design of SLIM is presented in Section 3 and the prototype implementation is discussed in Section 4. Section 5 evaluates SLIM using various benchmarks. Related work is presented in Section 6, while Section 7 concludes the paper.

2. PROBLEM SETTING

SLIM targets data center architectures with network-based storage, as shown in Figure 1. In this design, multiple racks of physical servers (hosts) are interconnected through a networking fabric built using commodity components. Each physical server is shared by multiple virtual machines (VMs) using storage area networks (SANs) to reduce operational costs. Block devices hosted in the storage arrays that compose the SANs are accessible from multiple hosts and provide persistent, reliable storage to virtual machines. A network-based storage design allows for storage optimizations such as *deduplication* to reduce the storage footprint and enables techniques such as *virtual machine migration* to be easily implemented, i.e., by avoiding the costs of transferring data between different hosts.

SANs can be implemented by using either dedicated networking hardware such as Fibre Channel (FC) based solutions or sharing the existing server communication networking fabric using iSCSI/Ethernet. While FC-based SANs offer high performance, these solutions require a separate and dedicated infrastructure, adding significantly to the infrastructure cost. Due to this reason, it is more prevalent to build low-cost SANs using commodity components such as iSCSI/Ethernet that share the existing server communication networking fabric.

Current data center networks are built hierarchically [3], with servers directly connected to a rack-level switch (RS). The rack-level switches are in turn connected to a layer of cluster-level switches (CS) through a number of network links. The number of rack-to-cluster links is usually a fraction of the total number of servers within a rack. As a result, communication links external to a rack are *oversubscribed* by a factor inversely proportional to the uplinks/servers fraction, with typical oversubscription factors of 10 or 20 – that is, the aggregate traffic generated within the rack is a factor of 10 more than the traffic that can be sent through the rack-to-cluster links. Storage arrays composing the SAN are directly connected to the cluster-level switches, as shown in Figure 1. As a consequence, with higher degrees of server consolidation, the aggregate traffic from the servers/racks *overwhelms* the capacity of the network fabric, thereby limiting scalability.

In this paper, we focus on improving the performance of the prevalent low-cost SAN implementations. Our main objective is to decrease the amount of bytes that exit a rack switch and reach the network storage system. We want to meet this goal while providing a similar level of data durability and reliability.

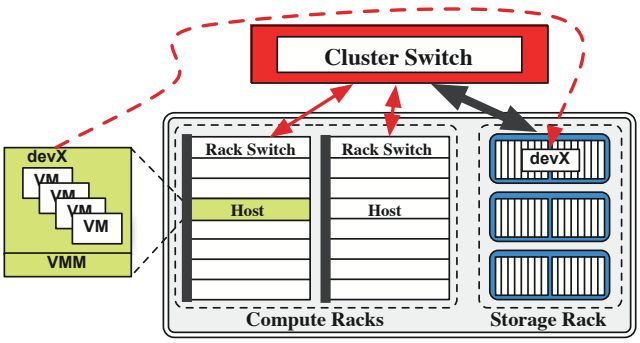

Figure 1: Data center architecture using network-based storage. We show the typical data center design using racks of compute servers and storage servers. The data from the compute servers pass through the cluster switch to reach the storage rack. These rack-to-cluster links are *oversubscribed* leading to a bottleneck with higher server consolidation.

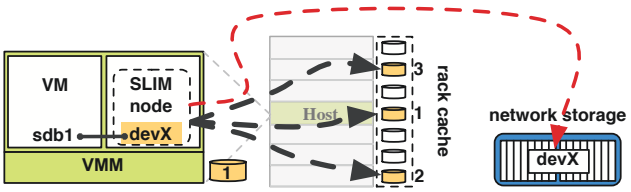

Figure 2: SLIM design. A client VM is given access to a network block device managed by SLIM. Data is replicated across 3 SLIM nodes in the rack. The local SLIM node labeled 1 is the device primary node. The SLIM nodes 2 and 3 are the device secondary nodes.

3. DESIGN

In this section, we present the design of SLIM. SLIM pools commodity drives located in rack servers to form a per-rack persistent cache for network storage block devices, as shown in Figure 2. All I/O operations to network block devices are intercepted by SLIM. I/O writes are logged to the rack cache and subsequently migrated to network storage. SLIM leverages the time window between storing data in the rack cache and migration to apply compression (outside the critical path), and to benefit from I/O overwrite patterns. On the server side, data is stored in its compressed form. I/O reads are served from both the network storage and rack cache, i.e., the I/O reads are directed to the most recent copy. The compressed data is de-compressed on the fly before returning the data back to the storage client. For higher reliability, SLIM uses N-way replication across nodes in the rack cache.

3.1 I/O Path

Handling I/O Writes: The I/Os issued by storage clients, e.g., block-level client driver in the virtual machine, are intercepted by SLIM and redirected to the rack cache. The clients are notified of an I/O completion as soon as data is stored in the rack cache. This allows the writes to take full

advantage of the intra-rack bandwidth. As the notification is sent after the data is written to the rack cache, the written data is also kept durable. Then, outside the critical path, SLIM applies a number of optimizations to the data in order to reduce the bytes sent to the network storage.

SLIM performs two optimizations. First, SLIM compresses the written data. Specifically, each write request is divided into fixed 4K blocks, compressed at a per-block granularity, and stored in its compressed form at the network storage server. Second, SLIM maintains the data in the rack for a time period to take advantage of overwrites. In more detail, SLIM controls the time data is stored in the rack cache before it is migrated back to the network storage server. By increasing the migration interval, SLIM is able to capture overwrites thus reducing the amount of data migrated back to the storage server. As an example, consider the following request stream: `write(0, 8192), write(0, 4096)` where the written data has a compressibility ratio of 0.5. Without SLIM, the number of bytes sent to the network storage is 12228. However, by redirecting requests to the rack cache, SLIM transfers only 4096 bytes, saving 4096 from overwrites and 4096 from compression.

Handling I/O Reads: SLIM satisfies a read request by fetching data from the network storage and, if necessary, combining it with data stored locally in the rack cache, depending on where the recent copy of data is stored. Specifically, each I/O read request is split into 4K blocks and each block is then filled with data from the network storage or rack cache. Blocks read from network storage are decompressed on the fly; decompression is usually much faster than compression therefore performing it on the critical path does not impact application performance significantly. Extending the previous example, assume a subsequent read request, `read(0, 16384)`. SLIM will read the first 8192 bytes from the rack cache, since the latest version of those 8192 bytes resides in the rack cache. SLIM will then fetch the last 8192 bytes from the network storage. Given a compressibility ratio of 0.5, 4096 bytes will be transferred from the network storage, saving 12228.

Data Migration: Data is migrated from the rack cache to network storage at various time intervals, depending on client I/O patterns. For instance, if a client application has a low percentage of I/O overwrites, then delaying the migration process may not provide savings. In this case, data is compressed and migrated immediately. On the other hand, for applications with high I/O overwrite ratios, a large migration time interval allows for significant savings thereby reducing the amount of data transferred back to the 'storage server.

3.2 Replication

SLIM uses N-way replication to deal with failures. There is a SLIM node running on each rack server managing the local drives as log-structured filesystems. I/O writes are logged to one of the local drives and replicated to N-1 other SLIM nodes within the rack. The SLIM node collocated on the same server with the client VM is the device primary node. The N-1 nodes part of the device replication group are the device secondary nodes. SLIM nodes in charge of a device form a device replication group. SLIM associates a device version number with every I/O write. Upon data migration, the device primary node updates a device checkpoint number as the highest device version number currently present in the network storage. All device secondary nodes are kept up to date with the latest device checkpoint number. SLIM uses the two numbers to recover from node failures, as detailed in the following section.

3.3 Failure Handling

By adding a level of indirection, e.g., a rack cache, SLIM introduces an additional point of failure. We classify failures as *server failures*, i.e., where an individual server crashes, and *rack failures*, i.e., where a rack switch or power failure can fail the entire rack. We describe how SLIM can handle each of these cases next.

Server failures: In case of a server failure, SLIM differentiates among network block devices that had the failed server as *primary* and devices for which the server was a *secondary*. For the former, SLIM migrates the changes from the latest device checkpoint up to the current device version to the network storage. For each device having the failed server as primary, one of the secondary nodes takes control of migrating the data remotely. For the latter, SLIM picks another node in the rack to replace the failed secondary. Data changes from the latest checkpoint are replicated to the new node by the primary. Note that, on a server failure, hosted client VMs are lost as well. If a user starts another VM to replace the crashed one, with the same network block devices, there could be a period of device unavailability, until the data migration process for the respective devices completes. This can be dealt with by placing the new VM onto the same rack as the crashed server, or by temporarily adding the previous device SLIM nodes as part of the device replication group.

Rack failures: As SLIM's goal is to reduce the traffic leaving the rack, the current SLIM design does not replicate data across multiple racks. This design decision makes SLIM susceptible to rack switch failures and rack power failures. Any failure of these components can lead to periods of unavailability of data that has not been migrated back to the SAN. We believe that the unavailability period is small and can be further minimized by adding some redundancy. Specifically, we believe that multiple rack switches and UPS units can be added at a minimal cost.

4. PROTOTYPE IMPLEMENTATION

We built a prototype of SLIM using the Network Block Device (NBD) protocol. NBD is a standard storage access protocol similar to iSCSI, supported by Linux. It provides a method to communicate with a storage server over the network. The client VM imports block devices from an NBD server. These block devices store application data. In the base case, the NBD server is the remote storage server. With SLIM, we implemented a proxy layer that sits between the client VM and the remote NBD server. The client now imports network block devices from SLIM, which in turn, becomes a client for the remote NBD server.

Our approach does not require any changes to the NBD client nor the remote NBD server. We store data in a log-structured file system both in the local and the remote network storage. On a write to a block, SLIM redirects the I/O

to the tail of the local storage log and records its new location in an internal in-memory map. The local storage runs a thread that migrates data from local to remote storage at a configurable interval and updates the remote log in-memory map. All data written in the previous interval is migrated periodically. The contents of the in-memory maps are periodically flushed to the local/remote storage for recovery purposes. The migration operation consists of reading the most recent version of the data from the local log, compressing (using `zlib`), and shipping the compressed data to the remote log.

5. PRELIMINARY EVALUATION

In this section, we evaluate SLIM against a base implementation that relies solely on network storage. We start by describing the methodology and benchmarks, then present our experimental results. In our experiments, we vary the data migration interval and the network oversubscription factor. We report savings in network traffic for write intensive and read intensive workloads, and the impact SLIM has on application performance.

5.1 Methodology

Our evaluation infrastructure consists of: (1) a remote storage server running NBD to provide block devices over the network, (2) a client running the benchmarks, and (3) two SLIM nodes, one co-located with the client VM, acting as a primary, and one located on a different server, acting as a secondary. We run these components in virtual machines on top of the Xen hypervisor on different physical servers. To evaluate SLIM with various oversubscription factors, we control the network bandwidth to the remote storage server using the Linux tool `tc`.

The physical servers are Dell PowerEdge SC1450 and we deployed Xen 3.2.1 as virtualization technology. We use four workloads: two micro-benchmarks based on the Linux+XEN source code, and two industry-standard benchmarks, TPC-C and TPC-H. We divide the benchmarks into write intensive and read intensive. Both TPC-C and TPC-H are backed by a MySQL/InnoDB (version 5.0.24) database engine. The client virtual machine, running the benchmarks, is configured with 2 virtual CPUs and 1.5 GB of RAM. The virtual machines running the SLIM nodes are configured with 1 virtual CPU and 512 MB of RAM. The remote storage virtual machine has 1 virtual CPU and 1.5 GB of RAM. All CPUs run at 3 GHz. For MySQL, we use the Linux `O_DIRECT` mode to bypass OS-level buffer caching. The current SLIM prototype does not incorporate memory resources into the rack cache. This means that, upon migration, SLIM reads data directly from the local drive. Also, for read intensive workloads, SLIM does not cache data read from network storage. Thus, the reported results are worst case and we expect SLIM to perform significantly better if the local memory was used for optimization.

5.2 Benchmarks

LINUX+XEN: We create two micro-benchmarks based on the source code tree for Linux and Xen. First, as a write-intensive task, we use `rsync` to copy the entire source code to the remote storage. Second, we perform a recursive `grep` on the source code, i.e., a read-intensive operation.

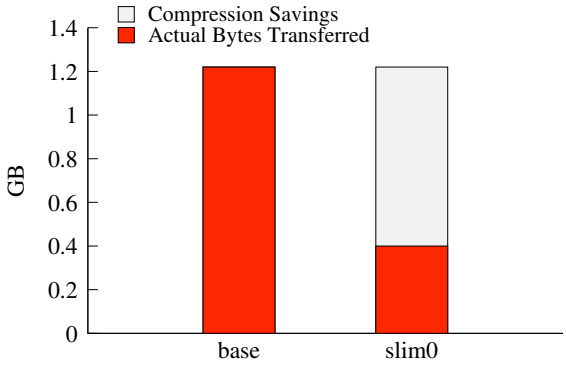

Figure 3: rsync XEN/LINUX micro-benchmark. Network storage traffic with `base` and `slim0` – SLIM achieves savings of over 50% due to compression.

TPC-C: The TPC-C benchmark simulates a wholesale parts supplier that operates using a number of warehouse and sales districts. The workload involves transactions from a number of terminal operators centered around an order entry environment. TPC-C is a write intensive benchmark with only 4% of the workload mix being read-only. We scale TPC-C by using 64 warehouses, which gives a database footprint of 6GB.

TPC-H: The TPC-H benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. TPC-H is a read intensive benchmark. It contains a set of 22 queries (Q1 to Q22) that are fired to the database engine. We scale TPC-H by a factor of 1, resulting in a database footprint of 4GB. For our preliminary evaluation we selected four I/O intensive queries: Q3, Q8, Q12, Q15.

5.3 Results

In our experiments, first, we measure the reduction in network traffic that SLIM achieves across different migration time intervals. Second, we evaluate the impact SLIM has on application performance with various network oversubscription factors.

We compare SLIM against a `base` system which communicates directly to the SAN. For SLIM, we vary the data migration time interval. We use `slimx` to term a SLIM solution with `x` seconds as migration interval. For instance, `slim240` represents SLIM with data migration performed every 240 seconds. We also evaluate against an ideal case, where data is migrated when load is low. We term this scheme `sliminf`.

To quantify network traffic savings, we measure the actual bytes transferred to network storage, the bytes saved due to compression and the bytes saved due to I/O overwrites.

5.3.1 *Write Intensive Applications*

rsync: First, we evaluate SLIM using a common write intensive task: `remote copy`. We use `rsync` to copy the Linux+XEN source code tree from the client VM root filesystem to the network storage. Figure 3 shows the network storage traffic breakdown over one run with the `base` and `slim0` schemes.

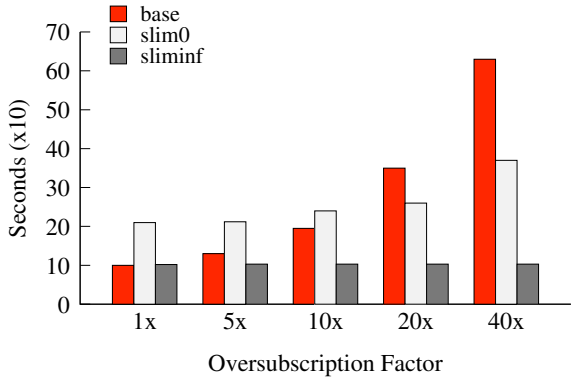

Figure 4: rsync XEN/LINUX micro-benchmark. Duration of `rsync` on source code tree – SLIM increases performance significantly with high oversubscription factors.

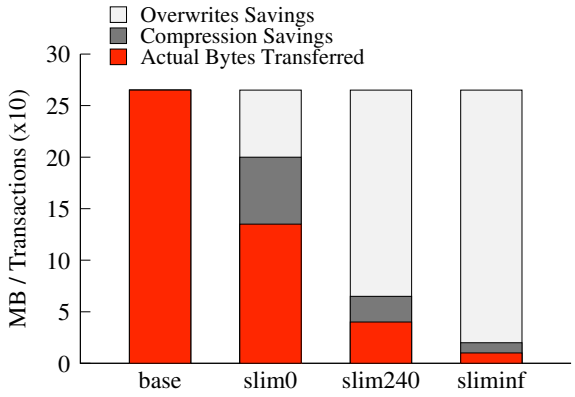

Figure 5: TPC-C benchmark. Network storage traffic with 20x oversubscription factor – SLIM reduces the amount of bytes transferred by leveraging compression and overwrites. Higher data migration intervals considerably reduce network storage traffic.

SLIM transfers less than half bytes to the remote storage, compared to the base solution. All savings come from compression, since the remote copy task has no I/O overwrites. In Figure 4 we plot the duration of `rsync` across various network bandwidth oversubscription factors with 3 schemes: `base, slim0, and sliminf`. With low oversubscription factors - 1x, 5x, `slim0` impacts performance. Although SLIM compresses data outside the critical path, it has to enforce the migration time interval. When the time interval expires, data needs to be migrated as soon as possible. SLIM is forced to slow down the application if migration is slower than the workload throughput. Note, however, that for our prototype, the SLIM nodes were configured with 1 CPU. By increasing the computing power in the rack cache, we should be able to compress faster and increase migration performance. With higher oversubscription factors - 10x, 20x, 40x, network becomes the main bottleneck. In these environments, SLIM increases the task performance significantly.

TPC-C: For the second set of experiments with write intensive apps we used TPC-C. We configured MySQL with 1 GB

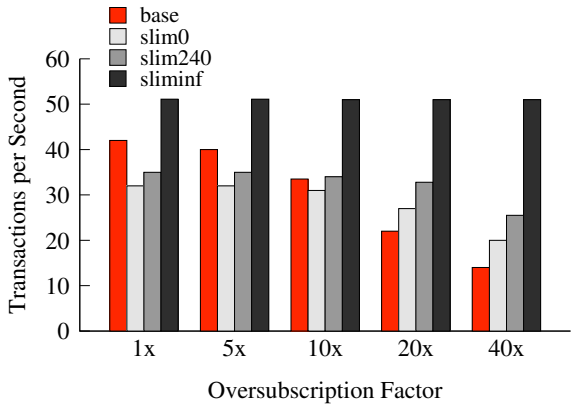

Figure 6: TPC-C benchmark. Throughput – SLIM has a minor impact on performance in well-provisioned networks, while doubling throughput when network is oversubscribed.

buffer pool size and we started 8 client terminals. In Figure 5 we show the network storage traffic breakdown normalized over the application throughput. TPC-C has a high rate of I/O overwrites. We benefit from this pattern by increasing the migration time interval. The `slim0` scheme transfers 50% less bytes than `base`, with half of the savings due to compression and half to overwrites. As we increase the migration interval, SLIM transfers less data to the remote network storage, with more savings coming from I/O overwrites. For instance, `slim240` transfers 73% less bytes than the base scheme. Figure 6 plots TPC-C throughput over various network bandwidth oversubscription factors. There are a number of trends worth mentioning in this graph. Notice that as we increase the oversubscription factor, the throughput for all schemes degrades. However, while in well-provisioned networks, SLIM slightly impacts performance, in network constrained settings SLIM improves application performance significantly. With a 40x oversubscription factor, `slim0` doubles the `base` throughput, while `slim240` almost triples it. Also note that, as we increase the migration interval, SLIM increases the throughput as well. This is expected, since we transfer less data remotely. Lastly, `sliminf` is constant since there is no data migrated during the experiment, so `sliminf` does not depend on the rack-external network bandwidth. Also, `sliminf` gives better performance than `base` with 1x factor. This is because `sliminf` writes and reads data from two different drives.

5.3.2 *Read Intensive Applications*

grep: We evaluate SLIM against a common read intensive task: recursive `grep` on the Linux+XEN source code tree. The total size on disk without our scheme is 1.3GB. With SLIM we reduce it to approximatively 512MB. Let us look at how decompressing data on the critical path impacts performance. In Figure 7 we plot the duration of the grep task across various network bandwidth oversubscription factors. When bandwidth is not an issue - 1x, 5x factors, our approach decreases performance. This is because we decompress data on the critical path. Also, as previously mentioned, our local storage has a minimal configuration, running with a single CPU. This can be improved by putting more computing power into the local rack cache. Thus, we

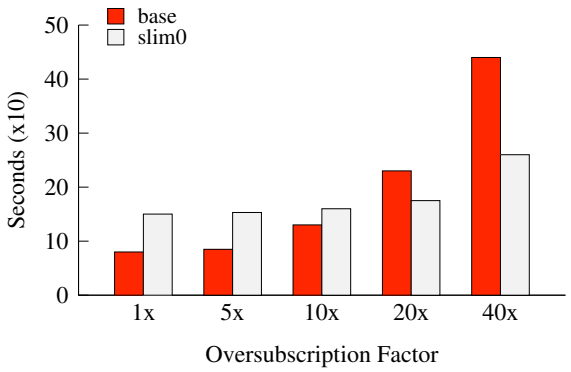

Figure 7: grep XEN/LINUX micro-benchmark. Duration of recursive grep on source code tree – SLIM impacts performance in unconstrained networks, but improves it in constrained settings.

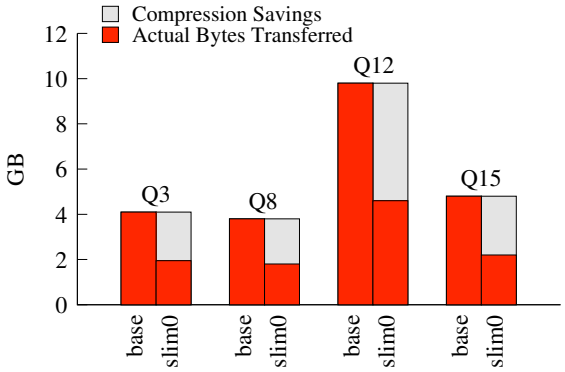

Figure 8: TPC-H benchmark. Network storage traffic per query – SLIM transfers 50% less bytes due to compression.

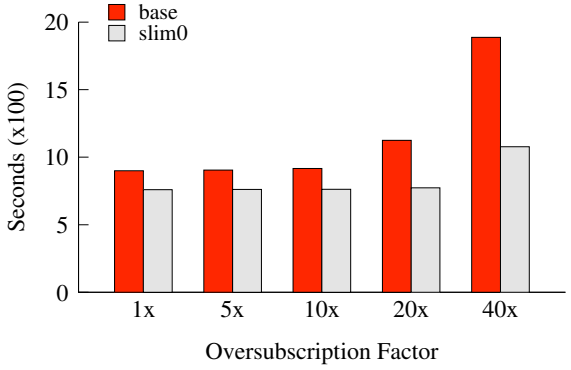

Figure 9: TPC-H benchmark. Duration of I/O intensive queries; average for the four queries used – SLIM performs consistently better.

can trade expensive I/O for cheap CPU. Note that, as bandwidth gets more oversubscribed - 10x, 20x, 40x, network becomes the bottleneck. Decompression on the critical path is no longer a problem.

TPC-H: For our last set of experiments we used TPC-H, a read-intensive workload. We selected four I/O intensive queries. We configured MySQL with 256 MB buffer pool size. TPC-H has a compressibility ratio of around 0.5. In Figure 8, we show the network storage bytes transferred with `base` and `slim0`. SLIM generates 50% less traffic for each query due to compression savings. Figure 9 plots the average latency for all queries over different network constraints. With high oversubscription factors - 20x, 40x, SLIM decreases query latency by up to 40%. In unconstrained networks, SLIM also improves performance, although by a lower percentage. Most of the queries generate a lot of traffic, but they are disk bound rather than network bound. Thus, even with one CPU for decompression, SLIM is able to sustain the workload.

6. RELATED WORK

Our scheme is similar to work in the area of optimizing data transfers for network file systems over wide area networks [7, 9]. These systems use techniques such as content hashing, compression or deferred writes to save bandwidth on WAN links. With SLIM, we apply similar concepts to the problem of reducing costs for SANs. However, unlike WAN-oriented approaches, which use the client disk drive as a cache, SLIM incorporates a reliable persistent cache by pooling disks within data center racks.

Compression techniques are implemented at various levels – that is, at the application, file system, or disk image level [1, 2]. In these approaches, the data is usually compressed on the critical path, thereby impacting application performance. Thus, clients may be reluctant to enable such optimization features. On the other hand, by storing the data temporarily in the persistent rack cache, SLIM can compress data transparently, outside the critical path, without a significant performance impact.

I/O offloading has been proposed at different levels in the storage hierarchy. Meyer et al. [8] developed Parallax, a storage system that leverages local server resources to implement fast snapshotting capabilities for virtual machine disk images on the client side. SLIM uses local drives to implement rack-level persistent write back caches for SANs. Narayanan et al. [10] used I/O offloading to handle transient spikes in workloads for network-based storage. Unlike our system, this work is targeting environments where network bandwidth is plentiful among all components (servers/storage). Soundararajan et al. [11] implemented a system that offloads I/O writes to HDDs to increase the lifetime of SSDs. Similarly, Hu et al. [6] proposed an approach called Disk Caching Disk (DCD), where a HDD is used as a log to convert small random writes into large log appends. During idle times, the cached data is de-staged from the log to the underlying primary disk. We leverage offloading to decongest rack-external network links to remote storage servers. Our goal is to minimize the traffic that hits a network storage. SLIM is applicable in current hierarchical data center networks, where high-level links are often oversubscribed.

Our work is also related to recent work studying data center networking architectures. Unlike recent approaches for scaling data center networks, which require significant architectural changes [4, 5], we propose a practical, low-cost solution for reducing network storage traffic on oversubscribed links.

7. CONCLUSIONS & FUTURE WORK

In this paper, we propose SLIM, a hybrid storage implementation, which uses disks located within each data center rack to build persistent write-back caches for network storage traffic. SLIM decreases traffic on the performance critical rack-to-cluster network links through the use of compression and batching. Our evaluation shows that SLIM is able to reduce network storage traffic by 40%-90%. Moreover, in bandwidth-constrained settings, SLIM increases application performance by up to 250% for TPC-C, a write intensive OLTP benchmark, and decreases latency by up to 42% for TPC-H, a read intensive decision support benchmark.

As future work, we are looking into improving the current prototype according to the design and evaluate its scalability with a high number of clients in large data centers. We are interested in making SLIM adaptive to application I/O patterns and resource utilization levels. The current SLIM design does not incorporate memory resources. In the future, we are considering rack level caches for both I/O reads and writes, by pooling available memory as well.

8. REFERENCES

[1] `http://opensolaris.org/os/community/zfs/`.

[2] `http://www.gnome.org/~markmc/qcow-image-format.html`.

[3] Barroso, L. A., and Holzle, U. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. In *Synthesis Lectures on Computer Architecture no. 6* (2009).

[4] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM* (2009).

[5] Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. In *SIGCOMM* (2009).

[6] Hu, Y., and Yang, Q. DCD- Disk Caching Disk: A New Approach for Boosting I/O Performance. In *ISCA* (1996).

[7] Kistler, J. J., and Satyanarayanan, M. Disconnected Operation in the Coda File System. In *ACM Transactions on Computer Systems* (1992).

[8] Meyer, D. T., Aggarwal, G., Cully, B., Lefebvre, G., Feeley, M. J., Hutchinson, N. C., and Warfield, A. Parallax: Virtual disks for virtual machines. In *EuroSys* (2008).

[9] Muthitacharoen, A., Chen, B., and MaziŕRes, D. A Low-Bandwidth Network File System. In *SOSP* (2001).

[10] Narayanan, D., Donnelly, A., Thereska, E., Elnikety, S., and Rowstron, A. Everest: Scaling Down Peak Loads through I/O Off-loading. In *OSDI* (2008).

[11] Soundararajan, G., Prabhakaran, V., Balakrishnan, M., and Wobber, T. Extending SSD Lifetimes with Disk-Based Write Caches. In *FAST* (2010).