RZ 3892 (#ZUR1512-013) 12/02/2015
Computer Science 12 pages

Research Report

SoftwAre Log-Structured Array (SALSA) – A unified stack for SSDs and SMR disks

Ioannis Koltsidas, Nikolas Ioannou, Kornilios Kourtis, and Thomas Weigold

IBM Research – Zurich
8803 Rüschlikon
Switzerland

SOFTWARE LOG-STRUCTURED ARRAY (SALSA)

A unified stack for SSDs and SMR disks

Ioannis Koltsidas <iko@zurich.ibm.com>
Nikolas Ioannou <nio@zurich.ibm.com>
Kornilios Kourtis <kou@zurich.ibm.com>
Thomas Weigold <twe@zurich.ibm.com>

1. Introduction

The explosion in data volumes as well as the emergence of new types of workloads (e.g., NoSQL) have increased the importance of cost-efficient, scale-out storage in modern data centers. To achieve cost-efficiency, new storage architectures take advantage of commodity hardware components. To accommodate the new requirements, storage device manufacturers are striving to provide higher capacity at a lower cost, often sacrificing performance as well as backwards compatibility of interfaces and ease of use to squeeze more capacity out of the storage media.

Low-cost SSDs. In the case of Flash Solid State Drives (SSDs), vendors are increasingly offering SSDs with higher densities (up to 4 TBs in a 2.5” drive) at lower prices (as low as $0.3/GB) by using consumer-level Flash and keeping the device controllers as simple as possible. However, the reduced price comes at the cost of lower write endurance and lower performance. In addition, traditional storage architectures are not a good fit for this type of devices: to deal with the low performance and reliability of disks, system designers have traditionally aggregated them in RAID arrays, with RAID5 and RAID6 being the most popular schemes in use. For low-cost SSDs, this approach only makes matters worse: the 2x and 3x write amplification penalty for small writes due to RAID5 and RAID6 read-modify-write, respectively, causes performance degradation and further accelerates wear-out. Amplified writes end up consuming most of the SSD resources, causing read operations to suffer long latencies: in our experiments with low-cost SSDs, we often observed that read operations wait for several milliseconds for writes to complete because of the high write amplification.

Host-Managed SSDs. Recently, SSD vendors have announced *host-managed* SSDs, which allow some of the internal operations of the SSD controller to be controlled by the host through software APIs and access the hidden Flash capacity that regular SSDs use as over-provisioning space. Although host-managed SSDs promise a higher potential for application-storage integration and lower cost, they come with a significantly higher complexity: they can no longer be accessed as regular block devices. Instead, the user has to access them over a specific interface through which low-level Flash operations, such as block erasures, are communicated to the device, meaning that a significant portion of the Flash management needs to be carried out by the user of the device.

SMR disks. In the case of magnetic Hard Disk Drives (HDDs), manufacturers started offering HDDs that use *Shingled-Magnetic Recording* (SMR). SMR is a technology that enables a higher grain density by using writes heads with stronger fields. SMR is particularly appealing as it does not require a dramatic rework of the magnetic media and can increase the capacity density without compromising the stability of the written bits. However, with SMR technology, the tracks on a platter overlap like rows of shingles on a roof, which prevents the user from modifying tracks independently of one another: modifying a disk sector results in corrupting sectors on overlapping tracks. To ensure that this never happens, specific access patterns need to be adopted and enforced, which entails the introduction of additional complexity before the user can access those drives using the conventional block interface.

Softw**A**re **L**og-**S**tructured **A**rray (or SALSA for short) is a unified software stack that enables the use of low-cost SSDs and SMR HDDs in existing systems. SALSA is targeted at cost-competitive, commodity storage devices and uses software intelligence to mitigate their limitations. By shifting the complexity from the hardware controller of the devices to software running on the host, SALSA not only reduces cost, but also takes advantage of the ample host resources to manage the device resources more effectively. For Flash-based SSDs, SALSA elevates their performance and endurance to meet the requirements of modern data centers, and in the case of host-managed SSDs, it provides a conventional block interface that hides the complexity of managing the resources of the SSDs. Similarly, for host-managed SMR HDDs, SALSA offers a conventional block interface and controls the data placement on the devices to improve their read and write performance. In addition, SALSA offers redundancy, storage virtualization and data reduction functionality, which allow the user to pool multiple devices and create storage volumes with improved performance, reliability and cost. Most importantly, SALSA exposes a standard block interface so that it can be used by file systems and applications without any modification. Using SALSA, we have achieved more than 1600% of performance improvement for a cloud database workload without any modifications to the database or the application.

In what follows, we present a brief overview of SALSA. In Section 2, we present the core technologies employed by SALSA when used with Flash-based SSDs and in Section 3 extensions that enable the use of SALSA on host-managed SMR disks. We provide some details about the implementation of the SALSA software stack in Section 4. In Section 5, we present the promising results of our experimental study with SALSA under various workloads, and conclude in Section 6.

2. SALSA for low-cost SSDs

SALSA intercepts the user I/O requests and remaps them in ways that transform the user access pattern to be as efficient as possible for the underlying storage medium. For instance, for a low-cost SSD, SALSA transforms the user access patterns to be as Flash-friendly as possible: SALSA is designed from the ground up for low write amplification. Thereby, the device controller hardly ever needs to relocate data internally, and is therefore relieved from a significant burden. As a result, the user experiences significantly improved performance and endurance.

The main benefits of SALSA come from the way data is organized on the SSDs. SALSA, at its core, follows a Log Structured Array (LSA) architecture: data is packed into logical segments appended to a global log structure that may span multiple devices. As data is written out-of-place, a dynamic mapping is maintained to keep track of the current data placement. Because of the out-of-place writes required for an efficient management of the log structure, Garbage Collection (GC) of invalidated data is required: SALSA uses a holistic data placement and GC scheme that reduces the probability of encountering invalid data upon a GC operation and thus reduces Write Amplification (WA). Using intelligent scheduling of foreground and background operations, SALSA guarantees an excellent read performance that is not disturbed by write traffic. The various technologies used by SALSA are outlined in the subsections below.

2.1. Log-Structured Data Placement

The goal of the log-structured data placement is to implicitly 'force' the device controller to not do any garbage collection internally. This is achieved by ensuring that SALSA will always write data to the SSDs using large sequential I/O operations. In particular, as write I/O comes down to SALSA from the user, SALSA packs the written data into large *segments*. Each segment is typically a multiple of the logical erase block size of the underlying device. SALSA always writes full segments to storage and, effectively, overwrites multiple logical erase blocks in the underlying device. As a result, when the underlying device attempts to perform garbage collection, all the data in a garbage-collected logical erase block are found to be invalid and no data relocation is required. In turn, this means that no write amplification is generated in the device.

This enables the use of SSDs with very simple controllers, coarse mappings and simple GC schemes. SALSA performs garbage collection at the host level above the device, where it not only has plenty of the host resources available (CPU, memory, etc.), but also can coordinate GC operations across multiple devices. To facilitate this higher-level GC, SALSA reserves a small portion of the SSD capacity as SALSA over-provisioning.

For host-managed SSDs, the situation is even more favorable: the host-managed device exposes an API by which SALSA can explicitly control some of the internal SSD functionality. With host-managed SSDs, SALSA can use the entire physical Flash capacity, completely disabling the internal SSD GC and data placement decisions.

2.2. Redundancy without performance penalties

SALSA can be configured to manage multiple devices in an array configuration, adding redundancy (parity) to guard against drive failures. SALSA provides RAID5-equivalent protection without the Read-Modify-Write penalty of RAID5. In particular, SALSA creates stripes out of multiple segments (there are n data segments and 1 parity segment per stripe in an array of $n + 1$ devices) and always performs full stripe writes, in which the parity is written only once after the data. Thus, write amplification is minimal ($1/(n + 1)$), in contrast to RAID5, which results in WA = 2 for small random writes. Moreover, SALSA performs array-wide wear leveling and load balancing, ensuring that workload hot spots and skew do not become performance bottlenecks.

2.3. Heat Segregation

SALSA continuously monitors the user accesses to logical data and maintains access statistics. This information is used to segregate hot data from cold data to reduce write amplification even further. In particular, SALSA keeps track of the data *heat*, that is, it classifies data as *hot* or *cold* (and various degrees in between), depending on the rate at which they get updated (overwritten) by the user. Data-placement decisions entail segregating hot data from cold data to ensure that data with a different level of heat do not get mixed up in the same segment. Thereby, we avoid the situation of continuously relocating cold data just because it happens to be collocated with hot, frequently changing data in the same segment. As a result, a considerable amount of write amplification is avoided, especially also because user access patterns tend to be skewed.

2.4. Garbage Collection

SALSA employs state-of-the-art garbage collection, which, together with the data placement logic, reduces the probability of encountering invalid data upon a GC operation. In deciding which segment to garbage-collect, the GC subsystem takes into account a) the logical origin of data blocks, b) their age in the system, and c) their frequency of updates to make optimal decisions.

Using a technique called *Recurring Pattern Detection*, SALSA can detect user write patterns that repeat in time or exhibit temporal locality. Because of the log-structured writes, temporal locality will translate into spatial locality in the physical domain, causing SALSA segments to eventually be invalidated. The GC process then takes special care not to interrupt such recurring patterns so that, upon relocation, segments will have as many invalid blocks as possible (and therefore incur lower write amplification).

2.5. Storage Virtualization

SALSA can virtualize a device or an array of devices, that is, a user can create logical volumes that are backed by an array of physical devices. What is even more important, SALSA takes advantage of the logical separation between different volumes to perform workload isolation at the physical level, meaning that data belonging to a logical volume will never get mixed up with data from other logical volumes. This not only results in separation of I/O streams in the data path, but also ensures that the write amplification generated by one volume will be isolated and thus will not interfere with that of other volumes. In this way, the system achieves an overall lower write amplification, and the administrator can easily monitor the effects of different workloads and find out which workloads generate a high load (because of high write amplification).

2.6. In-memory Caching

By using buffers in the host memory, SALSA can cache data in the user data path. Thereby, it can achieve very low write latency. In addition, SALSA achieves lower CPU load by batching write operations together and performing fewer, but larger writes. The user can limit the exposure to data loss (due to the volatility of main-memory buffers) by using consistency barriers along with the I/O requests.

2.7. Throttling

SALSA supports a sophisticated mechanism for throttling read and write I/O requests to ensure that a) read requests will not starve while waiting for high-WA write operations to complete and b) the garbage collection can advance at a pace that allows the device to remain responsive even at high loads.

2.8. Data Reduction

The LSA structure of SALSA enables inline compression of data as data is being packed into the LSA segments, but also allows SALSA to store objects efficiently without additional overhead other than the processing to compress and decompress data blocks. The SALSA architecture makes de-duplication a good fit for SALSA: we de-duplicate data in-line, using a fixed-chunk de-duplication scheme based on SHA1 fingerprints. The data reduction achieved with compression and de-duplication increases the effective capacity of the system, resulting in a total system cost similar to that of disk or lower, for certain workloads. Data reduction in SALSA is currently an experimental feature.

2.9. RDMA Interface

In addition to the conventional block interface, SALSA exposes an RDMA interface for accessing data. Using this interface, the user can use industry-standard RDMA interfaces to access data at byte granularity and has a straightforward way of integrating Flash-based storage with RDMA-enabled network stacks.

3. SALSA for host-managed SMR disks

Host-managed SMR disks typically have their storage space divided into a number of sequential *zones* that have no overlap with one another (i.e., they are physically separated by guard bands). A typical size of such a zone is on the order of 256 MB. Within a given zone, writes have to follow a strict sequential order because there is overlap between the tracks that make up a zone. Different sequential zones, in contrast, can be written to independently. In some cases, SMR disks also have a number of *conventional zones*, that is, non-shingled regions whose blocks can be written to and updated in any order. These zones typically amount to less than 1% of the total drive capacity.

SALSA offers the necessary extensions to also manage host-managed SMR disks, in addition to Flash SSDs. Although being completely different storage media, both Flash and SMR disks require out-of-place writes, which result in a data layout that necessitates space management, reclamation of invalid data and garbage collection. The main differences are that a) SMR disks do not need their zones to be erased before they can be written to (they only require the resetting of a pointer); they exhibit symmetric read and write speeds (for Flash, programming a page takes much longer than reading it), and c) having mechanical moving parts, they suffer from high rotational positioning and seek latencies. However, if one abstracts both device types as storage spaces divided into logical segments that can only be written to sequentially, then the idea and the overall goals remain the same. In particular, the most important goal is that data placement and garbage collection incur as low a write amplification as possible.

With SMR disks, SALSA reuses the same techniques for space management, data placement and garbage collection as for Flash SSDs. For SMR disks, the SALSA segment size is aligned to the zone size of the SMR disk. SALSA uses heat-segregating data placement, efficient parity protection without read-modify-writes, the same garbage collection algorithms, and the same schemes for storage virtualization, in-memory caching and throttling. In addition, SALSA uses the two techniques described below, which are specific to the geometry and characteristics of host-managed SMR drives.

3.1. Geometry-aware Data Placement

In addition to segregating data across segments based on its heat, SALSA takes into account the data heat to decide to which physical zone a particular SALSA segment should be written. Specifically, segments that hold data that is frequently read is stored in the outer tracks of the disk where the bandwidth is higher and seek times can be reduced. Overall, this results in improved read performance, even more so as workloads tend to be skewed.

3.2. Acceleration using Conventional Zones

SALSA can take advantage of the conventional zones in an SMR HDD to accommodate small writes efficiently. Small writes can be generated by the system either as a result of fine-grained updates to user data or of file-system metadata. By absorbing small writes in the conventional zones, SALSA can avoid a significant amount of write amplification and improve the overall system performance.

4. Implementation

SALSA has been implemented in Linux® as a library that provides interfaces that can be used by programs in both user space and the kernel. In the kernel, SALSA provides the standard block interface, i.e., users see SALSA volumes as normal block devices on top which they can deploy their *unmodified* file systems and applications. SALSA is implemented as a pluggable kernel module that can be loaded dynamically, and has been tested on the kernels of most modern Linux distributions, including Red Hat Enterprise Linux® 6 & 7, SUSE Linux Enterprise® 11, Ubuntu 14.04 - 15.10, for both the Intel® x86_64 and the IBM® Power® (ppc64) architectures. SALSA uses the Device Mapper framework of the Linux kernel to intercept requests and remap them to appropriate locations. After remapping, a request is forwarded to the device driver to be served as a normal I/O request to the device. SALSA does not modify any kernel subsystem or device driver. In the user space, SALSA can be linked to any user-space storage stack that needs to access low-cost SSDs or host-managed drives (SSDs or HDDs) efficiently.

SALSA consumes memory on the host to store its internal data structure and buffers during runtime. Its footprint is almost exclusively due to its mapping tables. When high performance is required with small I/O operations (e.g., in the case of SSDs), then typical SALSA configurations consume 200 MB – 1.5 GB of memory per user TB of storage. When not optimizing for small I/O operations (e.g., in the case of SMR HDDs), a typical SALSA configuration consumes 64 – 128 MB of memory per user TB of storage.

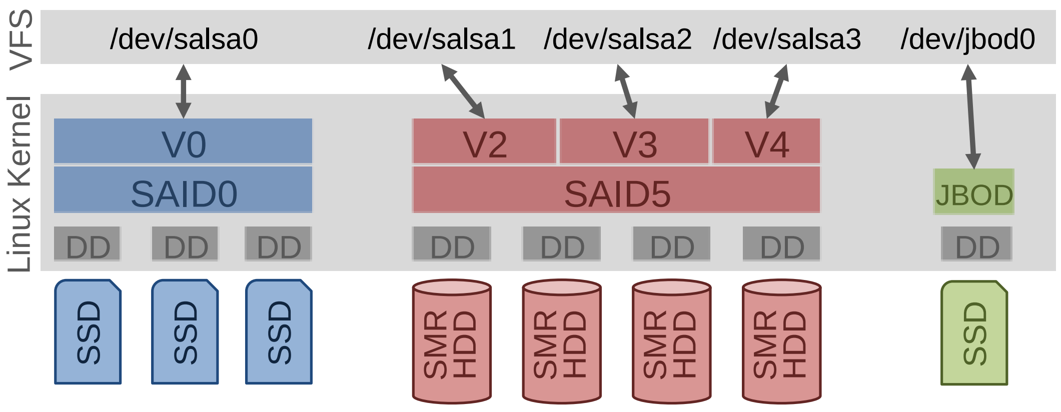

Figure 1: SALSA manages SSDs and SMR disks

Figure 1 shows an example of SALSA in the kernel. In that specific instance, the user has created a) a SAID0 (i.e., a RAID0-equivalent SALSA array) of three SSDs and a single volume on top of them (V0), which can be accessed by the user as /dev/salsa0; b) a SAID5 (i.e., a RAID5-equivalent SALSA array) of four SMR HDDs (i.e., an array equivalent in terms of redundancy to a 3+1 RAID5), and three logical volumes, V2, V3 and V4, on top of it that can be accessed by the user as /dev/salsa1, /dev/salsa2, /dev/salsa3, respectively, and c) a JBOD SALSA device, where a single physical SSD is exposed by SALSA as a single volume (which the user can access as /dev/jbod0).

Users can directly access the volumes as they would access any other block device, for example, to deploy a file system or a database. For instance, we have deployed local file-systems, such as ext4 and XFS, on top of SALSA, as well as distributed file-systems, such as IBM Spectrum Scale™ and Apache™ Hadoop® HDFS, and object storage systems, like Ceph and OpenStack® Swift. In addition, we have successfully deployed and run database systems, such as IBM DB2®, MySQL, and Oracle® Database 11g, and key-value stores, like RocksDB.

5. Experimental Results

We now present some experimental results with SALSA under micro-benchmarks as well as real application workloads. For the experiments, we used a standard commodity server equipped with two Intel® Xeon® E5-2630v3 CPUs and 128 GB of main memory. In all SSD experiments, we used low-cost SSDs made of TLC Flash, which were directly attached to the host as JBOD devices over SATA. In the SMR HDD experiments, we used a 10-TB host-managed HDD directly attached to the host over SATA.

5.1. Micro-benchmarks using SALSA on SSDs

5.1.1. Performance

In the first experiment, we used five identical low-cost SSDs in three different configurations: a) in a 5-drive software RAID0, b) in 4+1 software RAID5, and c) in a 4+1 SALSA SAID5 array, which is equivalent to RAID 5 in terms of redundancy and protection. After pre-conditioning the SSDs, we used a standard open-source I/O workload generator (fio) to generate a uniformly random read workload across the entire array using an I/O block size of 4 kB and doing direct (unbuffered) I/O. We increased the offered load by varying the queue depth of outstanding requests from QD = 1 to QD = 32. At each queue depth, we measured the IOPS and the average completion latency of the requests. The results are shown in Figure 2. As can be seen, all three systems achieved good performance: more than 250 kIOPS at approx. 200 – 250 µsec of latency.

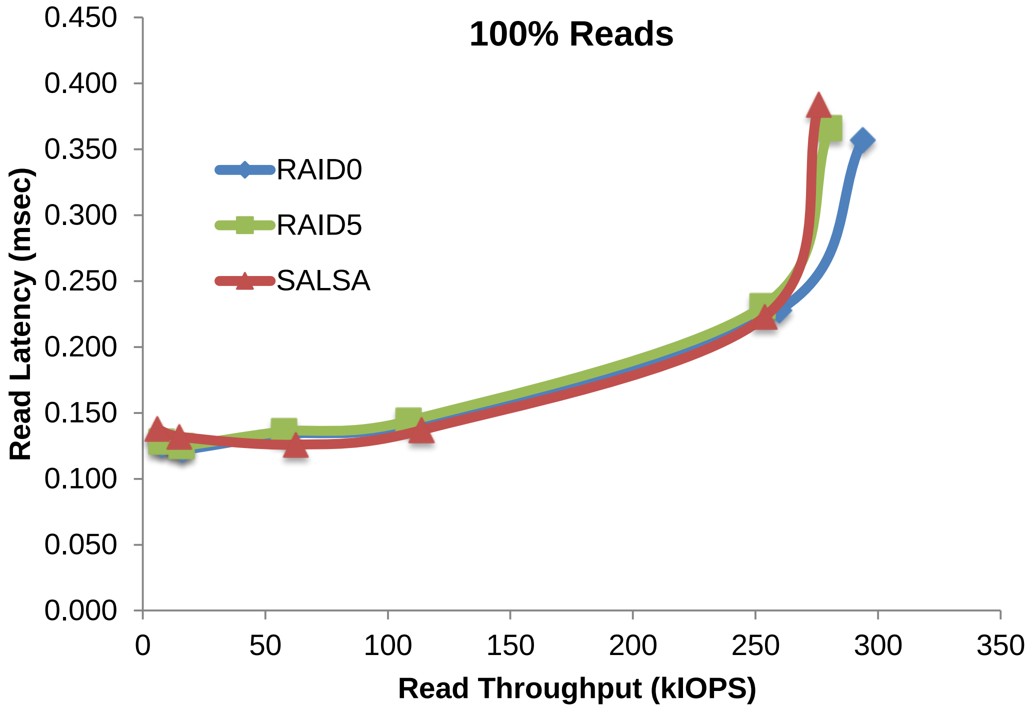

Figure 2: Performance for 100% reads (random I/O)

Next, we repeated the same experiment with the same uniform random access pattern, but doing 80% reads and 20% writes instead of 100% reads. The results are shown in Figure 3. In this case, the picture is very different for the three systems. The performance for RAID0 and RAID5 dropped down to 8.5 kIOPS and 4 kIOPS respectively: adding 20% of writes in the I/O mix resulted in the performance dropping by more than an order of magnitude! The reason for this drop is the significant write amplification that the random writes incur, which consumes most of the SSD internal bandwidth, leaving only very little resources to serve the reads. Thus, read operations are stuck waiting behind time-consuming writes, and the overall performance suffers severely. For RAID5, the random writes incur read-modify-write operations for the parity at the array level, which double the write amplification and therefore halve the performance of the system. In contrast, by shifting the burden of garbage collection to the host software, using the host resources for throttling and scheduling, and avoiding the read-modify-write penalty due to the log-structured data layout, SALSA was able to maintain a performance of more than 110 kIOPS, that is, about **13 times** better than RAID0 and **28 times** better than RAID5.

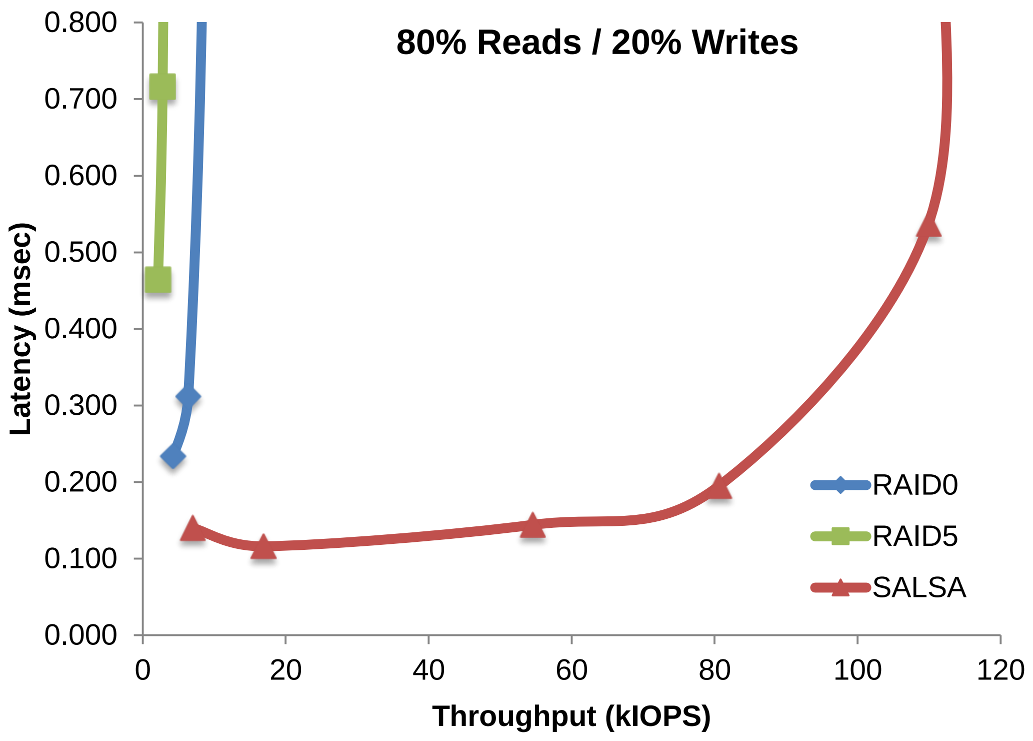

Figure 3: Performance for 80% reads and 20 writes (random I/O)

5.1.2. Endurance

Next, we experimented with SALSA in JBOD mode to measure its effect on the device endurance. After low-level formatting the device, we exercised one full device worth of random writes and measured the wear on the device, as reported by the relevant counter in the device's S.M.A.R.T attributes (the counter increases linearly with the number of Program-Erase cycles performed to the device's Flash cells). We then repeated the procedure 10 times with SALSA and 10 times on the raw device (without SALSA), and plotted the results in Figure 4. As expected, the lower end-to-end write amplification of SALSA translated into less wear to the device. Specifically, after the 10 iterations, SALSA had incurred 4.6 times less wear to the device, which would translate into a 4.6 times longer device lifetime.

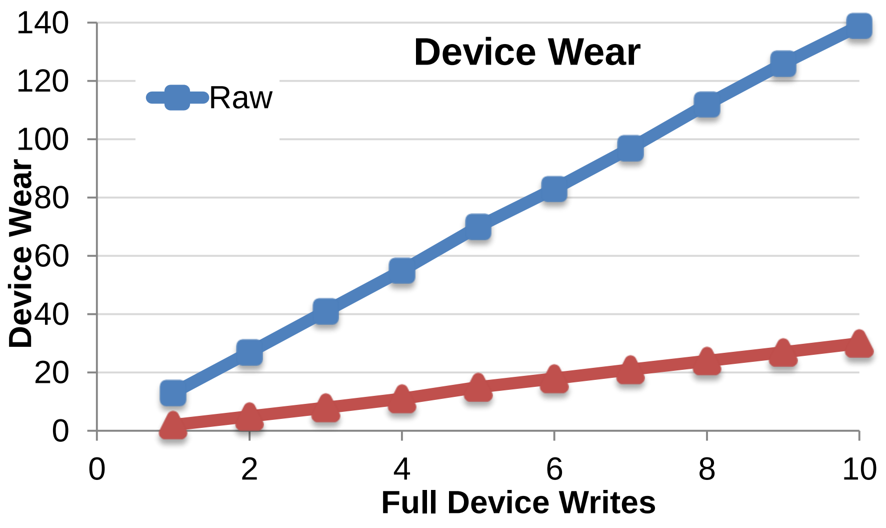

Figure 4: Device wear for 10 full-device writes

5.1.3. CPU utilization

We measured the CPU overhead of SALSA to assess to whether it can scale with the number of SSDs. For this, we ran SALSA in a server with 14 SSDs, creating an individual SALSA volume for each SSD. We ran a random write workload, as under random writes, SALSA consumes the most CPU cycles compared with reads or sequential writes: this is due to the high write amplification incurred by random writes, which in turn means heavy garbage collection and additional I/O for relocations. We ran the same workload with and without SALSA and measured the system CPU utilization. As the throughput of the two systems was different, we normalized the CPU utilization measured by the number of user IOPS served in the two cases and plotted the results in Figure 5. On average, SALSA only consumed ~7% more CPU cycles than the raw device per I/O operation.

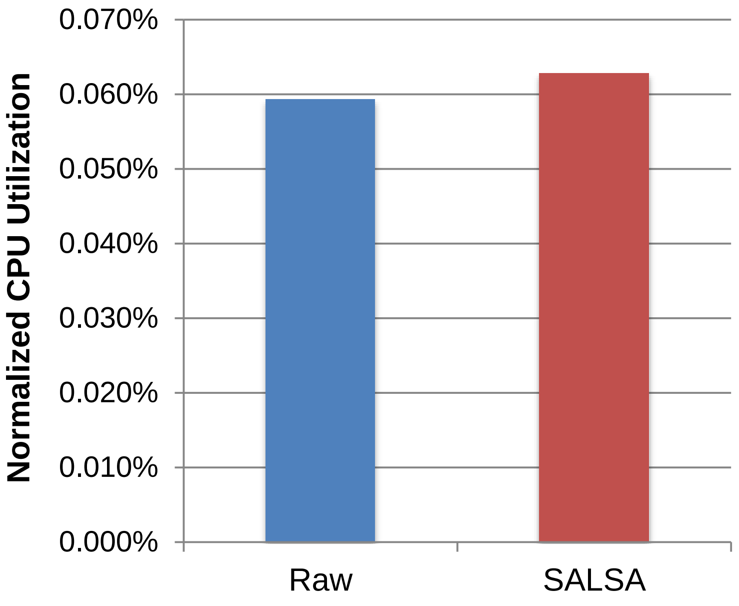

Figure 5: Normalized CPU Utilization

5.2. Cloud Database Workload using SALSA on SSDs

In the next experiment, we tested SALSA under a typical cloud workload: multiple database servers running in cloud VMs that share the same hardware resources. Specifically, we deployed multiple VMs that run the MySQL database on Ubuntu Linux 14.04, and created database load using the Sysbench benchmarking suite. Multiple database users submit queries to the database servers: the query processing stresses the storage and, as a result, the end-to-end performance is primarily determined by the storage performance. We ran the benchmark in two different configurations: one (‘baseline’) in which four low-cost 1-TB SSDs were used in software RAID5 and one (‘SALSA’) in which we created a SALSA SAID5 array (which is equivalent to RAID5 in terms of redundancy and protection). In each case, we measured the application-level transaction response time as well as the total throughput of the system (i.e., the total number of transactions executed per second). We increased the load by adding more VMs and more database users per VM. The results are shown in Figure 6. As evident, the system was able to realize the I/O performance improvement offered by SALSA all the way up to the application. With SALSA running on top of the SSDs, the system achieved a **16 times higher throughput** at ~100 msec of transaction response time and a 42 times lower transaction response time for a constant throughput of ~50 transactions per second.

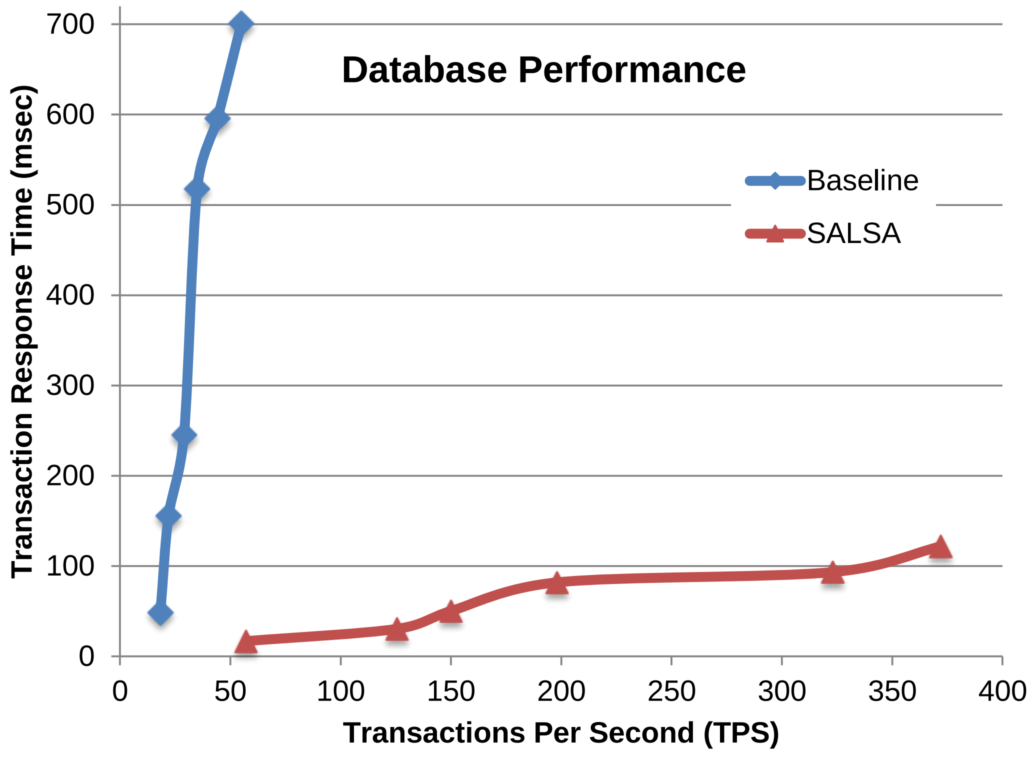

Figure 6: Database performance in a cloud environment

5.3. Micro-benchmarks using SALSA on SMR HDDs

Next, we experimented with SALSA on a host-managed SMR HDD ('SALSA on HM') and compared its performance with that of a drive-managed SMR HDD ('DM'), that is, one in which the indirection layer that manages the space on the drive, reclamation and garbage collection is implemented on the drive controller. As both SALSA and the drive-managed SMR HDD expose a block interface, we used the same I/O workload generator as in the previous experiments to generate I/O load to the device. Note that in this experiment we only used the sequential zones of the HDD; the optimizations involving the conventional zones are therefore not used, and the same holds for the geometry-aware data placement optimizations.

In the first experiment, we performed 64 kB sequential writes to the host-managed SMR HDD with SALSA ('SALSA on HM – Sequential') and measured the bandwidth. The results are shown in Figure 7, where we have also included the maximum performance of the formatted drive (which corresponds to the drive performance before it fills up and garbage collection is needed). As shown, SALSA on the host-managed HDD achieves a write performance that is within 90% of the 'ideal' drive performance (when no GC would be required). Next, we performed 64 kB random writes to both the host-managed HDD with SALSA ('SALSA on HM – Random') and to the drive-managed HDD ('DM – Random') and measured the total throughput. The results are also shown in Figure 7: SALSA achieved a steady-state performance of about ~9 MB/s, whereas the write performance of the drive-managed SMR disk dropped to almost 0, averaging at less than 150 kB/s at steady state. The SALSA performance was **64 times** higher than the performance of the drive-managed SMR drive.

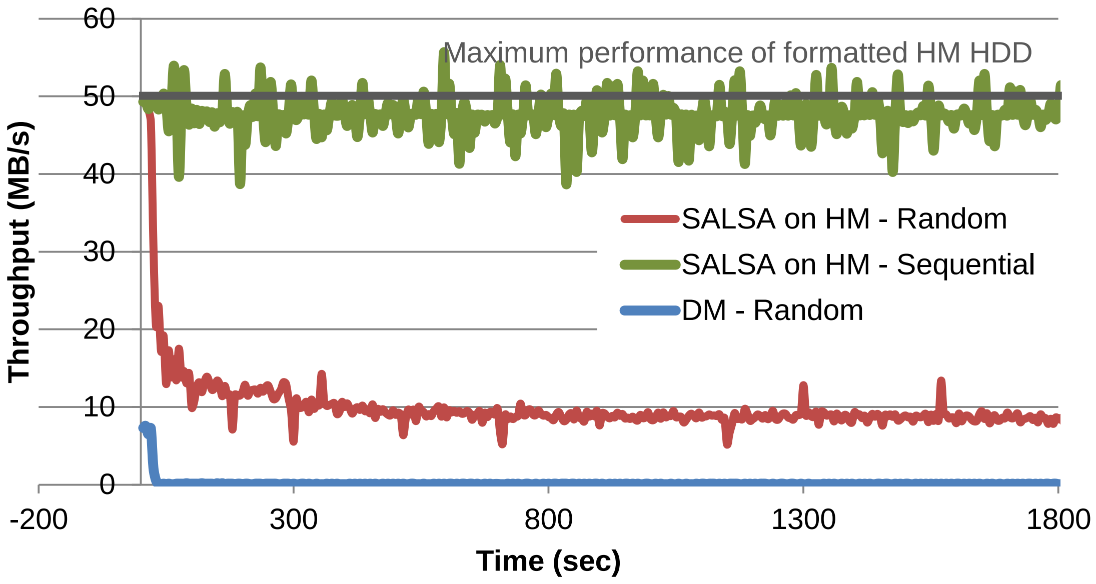

Figure 7: Write I/O Bandwidth with SALSA on SMR HDDs

6. Conclusion

Softw**A**re **L**og-**S**tructured **A**rray (or SALSA for short) is a unified storage stack for low-cost SSDs and SMR disks. Being a pure software technology that runs on Linux, SALSA can take advantage of commodity server hardware and commodity storage devices. By exploiting the software intelligence on the host, SALSA moves the complexity of managing Flash and SMR disks from hardware to software. SALSA elevates the performance and endurance of commodity Flash-based SSDs to meet the requirements of modern data centers.

For host-managed SMR disks, SALSA provides a traditional block interface that hides the complexities of SMR technology from the user, and dramatically improves write performance. Our results using commodity SSDs suggest that SALSA is a promising approach that, for certain workloads, could yield order-of-magnitude improvements, thereby enabling the use of commodity SSDs and SMR disks in modern data centers.