RAIDX: RAID without striping

András Fekete
University of New Hampshire
afekete@wildcats.unh.edu

Elizabeth Varki
University of New Hampshire
varki@cs.unh.edu

***Abstract*—Each disk of traditional RAID is logically divided into stripe units, and stripe units at the same location on each disk form a stripe. Thus, RAID striping forces homogeneity on its disks. Now, consider a heterogeneous array of hard disks, solid state disks, and RAM storage devices, with various access speeds and sizes. If this array is organized as a RAID system, then larger disks have wasted space and faster disks are under utilized. This paper proposes RAIDX, a new organization for a heterogeneous array. RAIDX disks are divided into chunks; larger disks have more chunks. Chunks from one or more disks are grouped into bundles, and RAIDX bundles chunks of data across its disks. The heterogeneity of disks causes unbalanced load distribution with some under-utilized disks and some bottleneck disks. To balance load across disks, RAIDX moves most frequently accessed chunks to under-utilized, faster disks and least frequently used chunks to larger, slower disks. Chunk remapping is done at the RAIDX level and does not impact file system to storage addressing. Experiments comparing local and networked RAIDX against local RAID show that RAIDX has faster throughput than RAID when the array is composed of heterogeneous disks: local RAIDX is 2.5x faster than RAID; networked RAIDX is 1.6x faster than local RAID.**

I. Introduction

RAID technology transforms an array of small, cheap disks into a storage system with large capacity, low failure rate, and high performance. The acronym, RAID, stands for Redundant Array of Independent/Inexpensive Disks, but it could also stand for Redundant Array of Identical Disks since RAID configuration assumes that the disks have identical capacity and speed. The performance of a parallel system is dependent on the slowest component, so disk homogeneity allows for the superior performance of RAID. However, the homogeneous disk constraint limits the applicability of RAID. With the emergence of Solid State Disks (SSDs), it would be useful to put together a RAID consisting of expensive, fast SSDs and the cheaper, slower disks. Moreover, when an old failed disk is replaced by a larger disk, it would be advantageous to incorporate the greater capacity and speed of the new disk. This paper addresses the problem of integrating heterogeneous disks into a RAID array.

In RAID, storage data are striped (distributed row-wise) across all the disks in the array. A stripe unit is the smallest block of addressable data. A stripe refers to all the stripe units at the same location on each disk. When a file is stored sequentially, the file's data are written on consecutive stripes spanning all disks. There are several RAID levels, which differ in how data and redundancy are striped. Two common RAID levels are RAID5 and RAID10. In RAID5, every stripe has a parity stripe unit; in case a stripe unit is corrupted, the corresponding parity stripe unit is used to regenerate the data. In RAID10, data are striped across all the disks, and every stripe unit has a copy to protect against single disk failure.

A common feature in all RAID levels is that stripe units from a stripe are at identical locations on each disk. This feature facilitates quick mapping from logical to physical space and ensures load balancing across the disks. However, when a disk in an array is replaced by a larger disk, the extra space in the new disk is wasted since the number of stripe units on this new disk cannot exceed the number of stripe units on the smallest disk in the set. Another disadvantage of RAID striping is that faster disks in the array are under-utilized, so it is wasteful to add fast disks to a RAID array. Thus, RAID striping is inefficient for an array of heterogeneous disks.

This paper proposes RAIDX, a new RAID technology for disks with varying capacities and speeds. The design objectives of RAIDX are to ensure that capacity of larger disks be included in the array and to ensure that slower disks are not bottlenecks. RAIDX has no stripe units, and it does not stripe data across its disks. RAIDX bundles data across its disks. A bundle need not span all disks of the array. Bundles are composed of storage blocks called chunks, with at most one chunk per disk. Chunks from a bundle need not be at identical locations on each disk. The assignment of chunks to bundles may be dynamically remapped. This feature of RAIDX is used to balance the inherent imbalance of RAIDX disks. When a disk becomes a bottleneck, most frequently used chunks from the disk are dynamically remapped to an under-utilized, faster disk. The remapping of bundles does not impact the file system or its address mapping.

Prior papers on RAID for heterogeneous disks have considered various data organization schemes that balance load and utilize faster disks efficiently. Several papers [1], [2] have proposed organizing multiple levels of RAID; lower RAID levels equalize the storage capacity of disks and higher RAID level perform the redundancy. Some papers [3]–[5] have proposed striping schemes for dissimilar sized disks. Recent papers [6]–[9] have studied RAID for arrays consisting of SSDs and HDDs - RAID is constructed on HDDs and a cache is constructed on SSDs. Prior papers have all used striping. We believe that we are the first to propose a non-striping organization for heterogeneous arrays. In earlier work [10], we proposed a chunk as the basic organizational block of a RAIDX array. In this paper, we extend this mechanism by bundling chunks of data across disks and developing a dynamic assignment of chunks to bundles. Our experiments show that RAIDX is faster than RAID when the array consists

of SSDs and HDDs.

II. RAIDX Architecture

RAIDX disks are divided into equal sized **chunks** - the basic storage unit. A RAIDX chunk is equivalent to a RAID stripe unit. The number of chunks on a disk depends on its capacity. Figure 1 shows a RAIDX array with 4 disks of unequal sizes; disk 1 has 7 chunks, disks 2 and 4 have 4 chunks each, and disk 3 has 5 chunks. Chunks are grouped into **bundles**. Each bundle consists of at most 1 chunk from each disk of the array, and bundles of an array have the same number of chunks. If a disk has two chunks in a bundle, then the failure of that disk would mean the loss of multiple pieces in a single-redundancy bundle. In Figure 1, each bundle consists of 3 chunks; bundle 1 consists of the first chunk (labeled B1 : B1-1, B1-2, B1-3) from disk 1 (B1-1), disk 3 (B1-2), disk 2 (B1-3); bundle 2 consists of chunks (labeled B2) from disk 1 (B2-1), disk 3 (B2-2), and disk 4 (B2-3), and so on. The label Bx-y represents the y^{th} chunk from bundle number x. A bundle is similar to a stripe, but bundles and stripes are fundamentally different. The number of chunks in a bundle may be less than the number of disks in the array, chunks of a bundle need not be at identical locations on each disk, and two bundles may contain chunks located on different set of disks within the array. Figure 2 shows a RAIDX array with 2 chunks per bundle.

RAIDX allows two types of redundancy, namely, **parity** and **copy**. In RAIDX-parity, each bundle contains a parity chunk; thus, RAIDX-parity is similar to RAID5. Assume that the RAIDX array of Figure 1 is RAIDX-parity with 3 chunks in a bundle: the third chunk of each bundle could be the parity chunk. In RAIDX-copy, each bundle consists of a chunk and its mirror; thus, RAIDX-copy is similar to RAID10. Assume that the RAIDX array of Figure 2 is RAIDX-copy: each bundle consists of 2 identical chunks - a chunk and its mirror. There is also RAIDX-zero which offers no redundancy and behaves like a mixture of RAID0 (horizontal striping) and JBOD (vertical striping) configurations.

For a RAIDX array, two input parameters are: 1) the size of a chunk and 2) the number of chunks in a bundle. The factors that determine the size of a stripe unit are relevant to determining the size of a chunk. A chunk size could be 4KB, 8KB, 16KB, 64KB, 256KB, or greater (similar to selection of stripe unit sizes).

The number of chunks in a bundle, **bundle length** L, could be a minimum of 1 and a maximum of D, the number of disks in the array. The allocation of chunks to bundles has more options when the value of L is small. The smallest L value depends on the redundancy level: for RAIDX-parity, L=3; for RAIDX-copy, L=2; for RAIDX-zero, L=1. There are three cases: L=1, L=D, $1 < L < D$. Below, we discuss the three cases.

The first case is 1 chunk per bundle (L=1). Here, the number of bundles on each disk is equal to the number of chunks on the disk (given by the disk's size divided by chunk size). This level of RAIDX has maximum capacity and no redundancy; we call it RAIDX-zero. The number of bundles, B, in the array is equal to the sum of number of chunks on all the disks. The bundles may be numbered as follows: bundle 1 (B1-1) consists of first chunk on disk 1, B2-1 consists of first chunk on disk 2, B3-1 consists of first chunk on disk 3, and so on.

The next case is D chunks per bundle (L=D) where D is the number of disks in the array. Here, the number of bundles in the array, B, is equal to the number of chunks on the smallest disk. Each bundle consists of 1 chunk from each disk. If each bundle consists of chunks at an identical location on each disk, then RAIDX is similar to RAID.

The final, most common case is $1 < L < D$: when the bundle length is greater than 1 but less than the number of disks in the array. There may be different strategies for assigning chunks to bundles. A simple allocation strategy that maximizes utilization of all disks is as follows: calculate the maximum number of chunks for each disk (disk size divided by chunk size). Initially, all chunks are unassigned to any bundle. Bundle to chunk assignment: B1 is assigned chunks from the L disks with greatest number of non-assigned chunks; next B2 is assigned chunks from the L disks with the greatest number of remaining unassigned chunks, and so on, until there is no space for further complete bundle assignment. For example, consider Figure 1 where L=3, and Figure 2 where L=2. In Figure 1: B1 consists of chunks from the three disks (D1, D2, D3) with maximum unassigned chunks; B2 consists of chunks from the three disks (D1, D3, D4) with the maximum remaining unassigned chunks, and so on. In Figure 2: B1 consists of chunks from the disks with the maximum number of unassigned chunks (D1, D3); B2 consists of chunks from the disks with the maximum number of remaining unassigned chunks (D1, D2), and so on.

The number of bundles in an array, B, is inversely proportional to the bundle length L. In Figure 1, B=6; in Figure 2, B=10. Intuitively, the degree of parallelism seems directly proportional to the bundle length, and L=D results in maximum parallelism. Reconsider Figures 1 and 2: L=4 with chunks bundled across all disks has parallelism of 4. However, L=2, can also result in maximum parallelism depending on how bundles are assigned to chunks. For example, suppose B1 is assigned to chunks from D1, D2, and B2 is assigned to chunks from D3, D4, and assignment continues in this manner. Thus, the degree of parallelism is determined both by the bundle length and the bundle allocation policy.

III. RAIDX lookup table

In RAID, stripe numbers are assigned to stripe units implicitly: stripe 1 consists of the first stripe unit from each disk, stripe 2 consists of the second stripe unit from each disk, and stripe i consists of the i^{th} stripe unit from each disk. In RAIDX, bundle numbers are assigned to chunks by a bundle allocation strategy, so a data structure that explicitly maps bundle numbers to chunks is required. The chunks in bundle i (Bi) are found by using a lookup table that maps bundle numbers to disk numbers and to the chunk number within the disk.

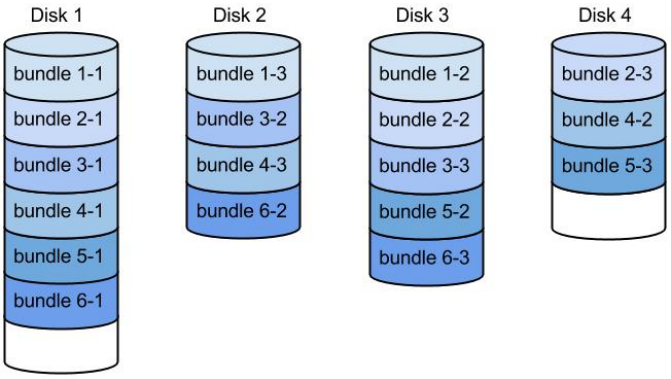

Fig. 1: Sample allocation of a 3 chunk per bundle array

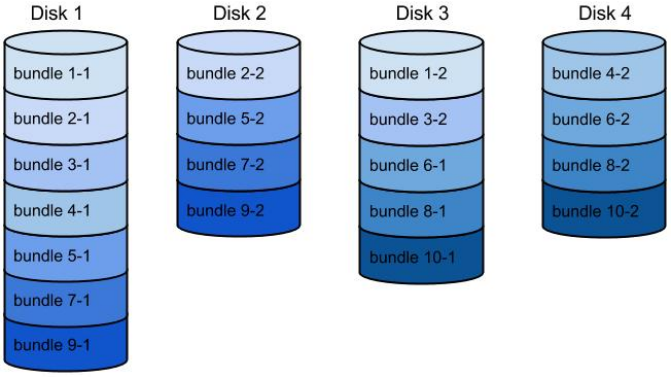

Fig. 2: Sample allocation of 2 chunk per bundle array

Each disk has an associated array that maps chunk numbers to bundle numbers. In Figure 1, disk 1's lookup array, D1-lookup is as follows: c[1]=1, c[2]=2, c[3]=3, c[4]=4, c[5]=5, c[6]=6, c[7]=0. The last chunk of D1 is not assigned to any bundle, so c[7] is set to 0. Disk 2's lookup array, D2-lookup, is as follows: c[1]=1, c[2]=3, c[3]=4, c[4]=6. Disk 3's lookup array, D3-lookup, is as follows: c[1]=1, c[2]=2, c[3]=3, c[4]=5, C[5]=6. Disk 4's lookup array, D4-lookup, is as follows: c[1]=2, c[2]=4, c[3]=5, c[4]=0.

At boot time, a bundle-lookup table (or RAIDX-lookup table) is constructed from the disk-lookup arrays; a RAIDX-lookup maps bundle numbers to chunk addresses. The RAIDX-lookup associated with Figure 1 is explained here. Bundle 1 is mapped to the first chunk in disks 1, 3, 2, so it may be written as: b[1]=[(1,1),(3,1),(2,1)] where (1,1) refers to disk 1, chunk 1; (2,1) refers to disk 2, chunk 1; (3,1) refers to disk 3 chunk 1. Similarly, for bundle 2, b[2]=[(1,2),(3,2),(4,1)], bundle 3, b[3]=[(1,3),(2,1),(3,3)], and so on.

Every request arriving at a RAIDX array is mapped to the correct disk location via the lookup table which is sorted and has a fixed size. Mapping request addresses to disk locations can be done directly in RAID; address mapping in RAIDX has a lookup overhead.

IV. Performance tuner

A RAIDX array is inherently unbalanced; the disks differ in sizes and speeds. In RAIDX, data are bundled across the heterogeneous disks of the array. A request that is bundled across several disks is completed only when all its chunks are read/written. A goal of RAIDX is to reduce mean response time by balancing load across the disks.

A strategy to lower response time of a multiple server system is the following: an arriving job should be directed to a faster server if the system is idle; on the other hand, an arriving job should be directed to an idle slower server if the faster servers are busy. The problem with storage is that requests can only be serviced from the disk on which the data resides. A larger disk has more chunks and it is natural that it will get a proportionally larger number of requests. Similarly, slower disks will have more outstanding requests waiting in the queue than faster disks.

The inherent variance in sizes and speeds of RAIDX disks would result in unbalanced load with some disks being under utilized while other disks are bottlenecks. The performance of RAIDX would be closer to the performance of the bottleneck disks. To prevent disk bottlenecks, the workload to the disks must be unbalanced too - faster disks should get proportionally more read/write requests. In RAIDX, we balance disk load by unbalancing the RAIDX workload. Next, we explain how RAIDX unbalances the workload.

Several papers [11]–[13] have shown that only a small percentage of storage space is accessed. Miranda and Cortes [12] showed that in various popular traces, 90% of I/O requests access less than 40% of the storage device. For some traces, less than 0.05% of the storage device is accessed. EMC states that up to 50% of allocated storage is unused [14]. This implies that only a small percentage of storage data are accessed frequently. If frequently accessed data are moved to under utilized disks, then the performance would improve.

It should be noted that under utilized disks are not necessarily the fastest disks. If all frequently accessed data are moved to the faster disks, then they would become the bottleneck when arrival rates increase. The workload is dynamic - the arrival rates change during the day and the data access patterns change during the day. The goal is to ensure that load is balanced dynamically across disks. The RAIDX performance tuner must address three issues: 1) tracking disk utilization, 2) shuffling (moving) storage data between disks, and 3) tracking frequently accessed chunks.

A. Disk utilization

Disk utilization is a measure of a disk's busy time - the proportion of busy time over total running time. If all the disks of RAIDX contained identical data, then an arriving request should be directed to the disk with no outstanding requests or an SSD with a short queue. (The SSD is an order of magnitude faster than the HDD, so an SSD with a short queue of outstanding requests may service an arriving request faster than an idle HDD.) To balance load, we require a measure of the load at each disk when a request arrives at the array.

We estimate arrival instant disk utilization from arrival instant queue lengths. Each time a request is submitted to RAIDX, we record the number of outstanding requests at each disk and compute a moving average of arrival instant queue lengths.

The disk with the greatest queue length is a bottleneck and frequently accessed chunks from this disk should be moved to disks with smaller queue lengths. If the average arrival instant queue length of all disks is close to 0, then it is best to move frequently accessed data to the fastest disk; if not, it is best to move frequently accessed data to under utilized disks.

B. Shuffling chunks between disks

In RAIDX, each bundle spans L disks of the array. It is possible that an entire bundle is frequently accessed; it is also possible that a chunk in a bundle is frequently accessed. When all the chunks of a bundle are frequently accessed, then the entire bundle must be moved to under-utilized disks. When a chunk is frequently accessed, only the chunk in the bundle needs to be moved.

In RAID, storage blocks cannot be moved. It must be handled at the file system level. Assume that this is not the case. If data from one stripe is moved to another stripe, then the new data address must be transmitted upward to the file system, which updates the logical block numbers. Without this update, the requests from file system would map to incorrect disk blocks. Therefore, in RAID, data may not be moved from one stripe to another stripe by the RAID controller. (Note that a disk may remap physical blocks to other disk locations, since this movement does not affect the logical block numbering.)

In RAIDX, however, it is possible to move storage data without updating logical block numbers. RAIDX moves data by moving the corresponding chunks and bundles where the data are stored. When chunks and bundles are moved, the only data structure that has to be updated is the RAIDX-lookup table (and the associated disk-lookup arrays). Changing the address of chunks and bundles does not impact the file system's logical block numbers since the file system maps to the same bundle.

For example, Figure 3 is the RAIDX array of Figure 1 with chunk B6-2 of bundle 6 (b[6]=[(1,6), (2,4),(3,5)]) moved to a faster disk 4. This change would be updated in the RAIDX lookup table as: b[6]=[(1,6),(4,4),(3,5)]. Suppose disk 1 is the bottleneck and chunk B6-1 is frequently accessed; if disk 2 is under utilized, the chunk B6-1 on disk1 may be moved onto disk 2, so that b[6]=[(2,4),(4,4),(3,5)]. The corresponding changes are also made in the disk-lookup arrays.

Thus, RAIDX moves data between disks by shuffling chunk addresses. The shuffling of chunks and bundles is an atomic (all-or-nothing) operation. Initially, all disks in RAIDX will have free (unallocated) chunks used as temporary chunks for shuffling. A chunk may only be moved to an unused chunk. This ensures atomicity as the chunk is not considered moved until the lookup table is updated. The shuffling occurs during RAIDX's idle time. If a request arrives during shuffling, the shuffling stops and the request is serviced.

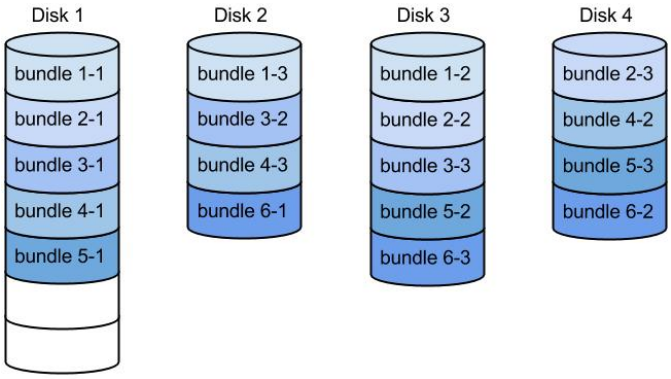

Fig. 3: Disk array of Figure 1 with some chunks relocated.

C. Tracking chunk access frequency

The objective is to balance the imbalance of RAIDX hardware. We achieve this by unbalancing the workload so that more requests are serviced by faster disks. To do so, frequently accessed chunks must be moved to faster disks, and less frequently accessed chunks must be moved to larger, slower disks. Thus, RAIDX must track the rates at which chunks are accessed.

RAIDX uses the frequency of access of a chunk to determine the chunk's placement on disk. This is similar to how cache replacement algorithms determine what data should remain in cache and what data should be evicted. Cache replacement policies such as LRU (Least Recently Used) and LFU (Least Frequently Used) evict blocks from the cache. RAIDX uses similar algorithms to determine chunk placement on disk.

RAIDX tracks the Most Frequently Used (MFU) chunks of the array. When there is a lull in array traffic, RAIDX starts chunk shuffling by moving the most frequently accessed chunk from a bottleneck disk to an under-utilized faster disk. If the fast disk is full, then the least frequently used chunk on the fast disk is swapped out. During chunk shuffling, the algorithm must ensure that there is at most 1 chunk from a bundle on each disk.

V. Experimental evaluation

The goal of our experimental evaluation is to compare the performance of RAIDX against the performance of RAID. Our implementation of RAIDX, shown in Figure 4, uses the ZeroMQ framework [15], a high speed bus/network transmission protocol. ZeroMQ reduces the number of memory copies and has a minimal packet header size, thereby lowering transmission overhead. In Figure 4, our application, ZMQ-Device, connects the virtual device to a raw block device. We created ZMQ-RAIDX to combine multiple ZMQ-Devices together to form a RAID. Our ZMQ-ToNBD application creates a virtual device that can be mounted on a Linux filesystem. Each of the ZMQ devices connects to a disk. The ZMQ RAIDX stores the RAIDX-lookup table and the performance tuning data structures. We used Multiple Device RAID (MD-RAID) [16]

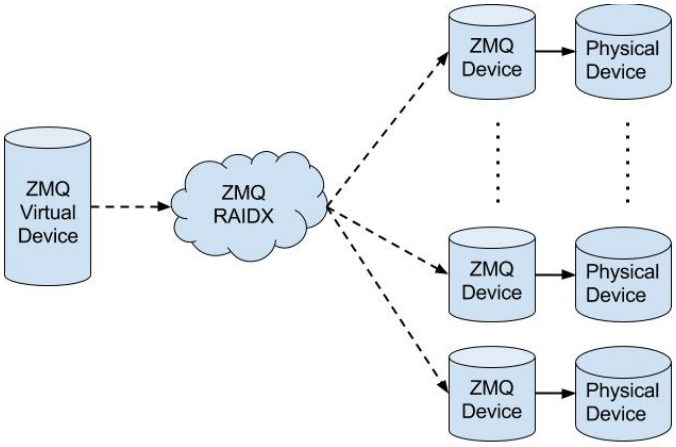

Fig. 4: ZeroMQ RAIDX architecture: dashed arrows show connections that can be either local or networked, solid arrows are local connections

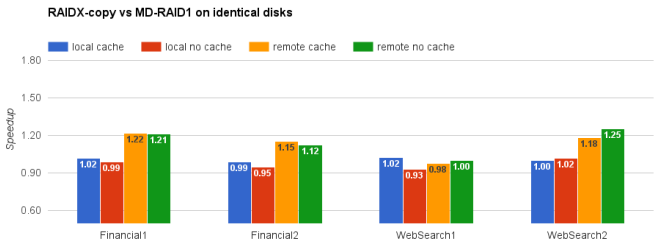

Fig. 5: Speedup comparison between traditional RAID and RAIDX when array has identical disks

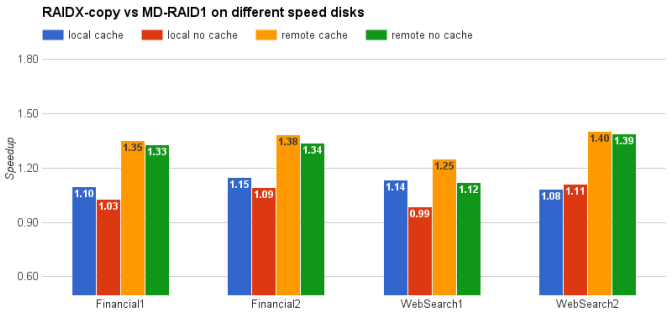

Fig. 6: Speedup comparison between traditional RAID and RAIDX when array has disks of varying speeds

the Linux software RAID, as the experimental control RAID platform.

Our first two experiments evaluate the overhead of lookup tables in RAIDX. For these experiments, we did not run the performance tuner in RAIDX. The disk array consists of two disks where each disk is a copy of the other - for RAID the redundancy is RAID1, for RAIDX the redundancy is RAIDX-copy. The RAIDX configuration can be local or remote, where local implies that storage and testing computer are in the same box and remote implies that storage is on the network. The RAIDX configuration can have storage cache activated or disabled. Thus, we tested four RAIDX configurations: 1) local with cache, 2) local with no cache, 3) remote with cache, 4) remote with no cache. We tested only one RAID configuration - local with cache activated. The workloads are financial and websearch trace files [17]–[19]. Each workload is executed on a disk array configuration (RAIDX - local with cache, local with no cache, remote with cache, remote with no cache; RAID - local with cache) and the time to complete execution is noted. We ran each workload on a given disk array configuration several times and found that the results were statistically identical.

In the first experiment, we used two identical Samsung 850 EVO 250GB SSD disks. Figure 5 shows the results of our first experiment with similar disks. The goal of this experiment is to evaluate the performance slowdown caused by lookup tables in RAIDX. Each bar graph plots RAIDX speedup: the time to run a workload on RAID divided by the time to run the same workload on RAIDX. A speedup of 1 is obtained when RAID and RAIDX have similar times; a value less than 1 is obtained when RAID is faster than RAIDX; a value greater than 1 is obtained when RAID is slower than RAIDX. Our results show that the overhead of lookup tables is not significant. The graph shows that when RAIDX is remote, it is slightly faster than RAID (which is always local); this result is surprising and we conjecture that the superior remote RAIDX performance is a result of storage computation being carried out on a separate computer in remote RAIDX whereas storage and workload computation are carried out on the same computer in local RAID and local RAIDX.

For the next experiment, we used two disks, a 300GB HDD and a 250GB SSD. The goal of this experiment is to evaluate the overhead of RAIDX lookup tables without the performance tuner. The results (see Figure 6) show that RAIDX performed slightly better than RAID. The reason for RAIDX's performance edge is its queuing algorithm, which determines which disk copy should service a read request: when both disk queues are empty RAIDX selects the faster disk while RAID randomly chooses either disk.

Our next experiment evaluates the impact of the performance tuner in RAIDX. For this experiment, we used the same hardware platform as the previous experiment with a 250GB SSD and 300GB HDD disk. We removed redundancy from the array, so the array is configured as RAID0 and RAIDX-zero. Instead of using standard workload traces, we generated a workload. We developed diskSpotcheck [20], a tool that generates random I/O requests on the disks. DiskSpotcheck mimics a multi-user system with various users using various files; the random workload ensures that performance does not include caching effects. The workload consists of write requests, later followed by read requests to the written data. With each each array configuration - RAID0 and RAIDX-zero - we ran the workload four times. For each of the four runs, we measured the total time to execute the workload. We generated a total of three random workloads using diskSpotcheck with

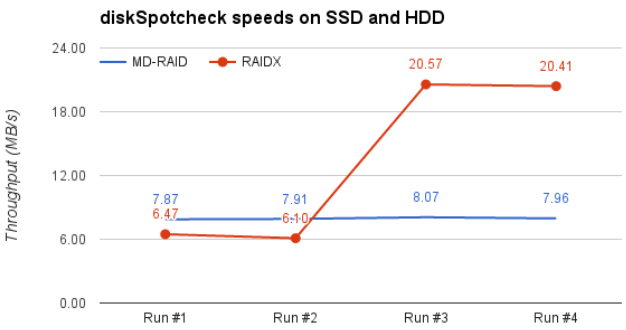

Fig. 7: Comparing RAID0 to RAIDX-zero

different random seeds. Figure 7 plots the results of our experiment. The X-axis specifies the four runs and the Y-axis specifies the mean throughput (for the three workloads). The 99% confidence interval is worst case ± 0.006 for the values in the figure. As expected, the run time is constant for RAID. For RAIDX, however, the run time decreases since the performance tuner executes between runs. The performance tuner moves frequently accessed chunks to the faster disk. For RAIDX, throughput increases with each run; there is a 2.5x speedup after the second run.

VI. Conclusions

This paper develops RAIDX, a new RAID for heterogeneous arrays of disks with varying speeds, sizes, and technologies. RAIDX is unique in that it does not stripe data across disks; RAIDX bundles data across disks. Bundles are composed of chunks, with at most one chunk per disk. The number of chunks per disk varies according to size. Therefore, RAIDX incorporates the capacities of larger disks.

A novel feature of bundles is that chunks are dynamically assigned to bundles. At boot time, a bundle may be assigned chunks from disks, say 1, 2, 3, 4; at a later time, the same bundle may be assigned chunks from disks, say 2, 4, 5, 6. This dynamic assignment of chunks allows RAIDX to balance load across disks and maximize performance. Frequently accessed chunks are moved to faster disks on-the-fly, thereby ensuring that there are no under-utilized disks nor bottleneck disks.

RAIDX provides two types of redundancy, namely RAIDX-parity, which is similar to RAID5, and RAIDX-copy, which is similar to RAID10. The goals of RAIDX are to provide redundancy and parallelism of traditional RAID, and to provide the access speeds of the faster disks in the array. Our experiments suggest that RAIDX satisfies these goals. RAIDX is a suitable technology for arrays composed of slower hard disks and faster solid state disks.

RAIDX has a performance tuner that tracks chunk access frequency and determines what chunks should be moved, when they should be moved, and where they should be moved. Thus, RAIDX has computational overhead and traffic overhead. The movement of chunks between disks causes traffic at the disks. We plan to study the overhead of RAIDX performance tuner. Our preliminary implementation of the performance tuner uses an approximate LFU algorithm. We plan to analyze alternative algorithms for the performance tuner.

References

[1] A. Thomasian, "Disk arrays with multiple RAID levels," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 5, pp. 6–24, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2641364

[2] N. Jeremic, H. Parzyjegla, and G. Mühl, "Improving Random Write Performance in Heterogeneous Erasure-Coded Drive Arrays by Offloading Code Block Requests," pp. 2007–2014, 2015.

[3] T. Cortes and J. Labarta, "A Case for Heterogenenous Disk Arrays," *Proceedings of the IEEE International Conference on Cluster Computing (Cluster'2000)*, pp. 319–325, 2000.

[4] J. L. Gonzalez and T. Cortes, "Adaptive Data Block Placement Based on Deterministic Zones (AdaptiveZ)," pp. 1214–1232, 2007.

[5] M. D. Flouris and A. Bilas, "Violin: A framework for extensible block-level storage," *Proceedings - Twenty -second IEEE/Thirteenth NASA Goddard Conference on Mass Storage Systems and Technologies*, no. i, pp. 128–142, 2005.

[6] S. Im and D. Shin, "Flash-aware RAID techniques for dependable and high-performance flash memory SSD," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 80–92, 2011.

[7] N. Jeremic, G. Mühl, A. Busse, and J. Richling, "The pitfalls of deploying solid-state drive RAIDs," in *Proceedings of the 4th Annual International Conference on Systems and Storage - SYSTOR '11*. ACM Press, 2011, p. 1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1987816.1987835

[8] J. Ren and Q. Yang, "I-CASH: Intelligently coupled array of SSD and HDD," *Proceedings - International Symposium on High-Performance Computer Architecture*, vol. 02881, pp. 278–289, 2011.

[9] A. Miranda and T. Cortes, "CRAID: Online RAID Upgrades Using Dynamic Hot Data Reorganization," *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, pp. 133–146, 2014. [Online]. Available: http://blogs.usenix.org/conference/fast14/technical-sessions/presentation/miranda

[10] A. Fekete and E. Varki, "RAID-X : RAID eXtended for Heterogeneous Arrays," in *30th International Conference on Computers and Their Applications*, 2015, pp. 157–162. [Online]. Available: http://www.bandilabs.com/2015/03/12/cata-2015-presentation-of-raid-x-raid-extended-for-heterogeneous-disks/

[11] T. Gibson, E. L. Miller, and D. D. E. Long, "Long-term File Activity and Inter-Reference Patterns University of California," *International Business*, no. December, pp. 976–987, 1998.

[12] A. Miranda and T. Cortes, "Analyzing Long-Term Access Locality to Find Ways to Improve Distributed Storage Systems," *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on*, pp. 544–553, 2012.

[13] D. Roselli, J. Lorch, and T. Anderson, "A Comparison of File System Workloads," *In Proceedings of the 2000 USENIX Annual Technical Conference*, pp. 41–54, 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.195

[14] EMC, "EMC VNX Virtual Provisioning," Tech. Rep. December, 2013.

[15] "ZeroMQ," 2015. [Online]. Available: http://www.zeromq.org

[16] G. Oxman, I. Molnar, and M. de Icaza, "The linux raid-1,-4,-5 code," *Linux Journal*, no. 44, 1997. [Online]. Available: http://www.linuxjournal.com/article/2391

[17] J. Wan, J. Wang, Q. Yang, and C. Xie, "S2-RAID: A new RAID architecture for fast data recovery," *Mass Storage Systems and ...*, vol. c, 2010. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5496980

[18] S. Wu, D. Feng, H. Jiang, B. Mao, L. Zeng, and J. Chen, "JOR: A journal-guided reconstruction optimization for RAID-structured storage systems," *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, pp. 609–616, 2009.

[19] F. Ye, J. Chen, X. Fang, J. Li, and D. Feng, "A Regional Popularity-Aware Cache Replacement Algorithm to Improve the Performance and Lifetime of SSD-based Disk Cache," pp. 45–53, 2015.

[20] A. Fekete, "diskSpotcheck," 2016. [Online]. Available: https://github.com/bandi13/diskSpotcheck