Violin MEMORY

Technical Report

Benchmarking Hadoop & HBase on Violin

Harnessing Big Data Analytics at the Speed of Memory

Version 1.0

Abstract

The purpose of benchmarking is to show advantages in performance of running Hadoop and HBase on Violin flash Memory Arrays compared to conventional hard drives. The intuition is that Memory Arrays should provide better performance than disks for applications doing many random reads and reads mixed with writes.

Contents

1 Introduction: Hadoop, HBase and Violin Memory

Hadoop and HBase are the de facto leading software tools for solving Big Data problems.

Hadoop is a distributed platform, which provide a reliable scalable distributed file system HDFS, and a distributed computational framework Map-Reduce for the analysis and transformation of very large amounts of data. HBase is a distributed table-like key-value storage system on top of HDFS, which provides nearly linear time access to the data stored in sparse loosely structured tables.

The data is cached on many computers and served to the clients from RAM. Hadoop and HBase combine the computational power and local storage of hundreds or thousands of individual servers into a reliable and scalable cluster.

Violin Memory is a provider of one of the world's fastest and most scalable flash-based storage arrays. Typically the servers in a Hadoop and HBase cluster are equipped with several hard drives per node used to store the Hadoop data. Replacing the drives with Violin's Scalable Flash Memory Array is intended to boost the performance of small to medium size Hadoop/HBase clusters, which is the dominating use case for Hadoop today.

1.1 Benchmarks Used for This Report

The benchmarking is based on the two industry standard benchmarks: DFSIO and YCSB.

DFSIO is a standard Hadoop benchmark, which measures the IO throughput on Hadoop clusters. DFSIO is a pure HDFS benchmark known to be able to saturate hard drives' throughput, when disks are used as local storage. Flash Memory Arrays should be able to surpass the threshold bounding the performance of hard drives.

YCSB is a standard benchmark for evaluating the performance of key-value storage systems like HBase. YCSB allows one to define different workload scenarios for the system by mixing record reads, writes, updates, and table scans, and then measures the performance of the system on a particular workload.

Hadoop in general and HBase in particular are known to be low CPU usage systems. Some internal restrictions imposed by the implementation do not allow boosting the CPU usage to its maximum if a single instance of HBase server software is running on each node. To address this we use virtualization software to obtain high average CPU usage per node, which increases the cluster performance for the non IO bound workloads typical for Violin Arrays.

2 The Test Environment

Table 1 lists the key hardware and software components used for the benchmarks in this report.

Component	Insert %
Hadoop	1.0.3
HBase	0.92.1
CPU	Intel 8 core processor with hyper-threading
RAM	55GB
Disk Storage	Four 1TB 7200 rpm SATA drives
Network Interface	1 Gbps and 10 Gbps
Operating System	Linux kernel 3.3.0
Virtualization Software	VMware ESXi 5.0 Update 1
Flash Storage	Violin 6000 Series Array

3 Cluster Node Configuration

We assembled two 4-node clusters to run Hadoop and HBase benchmarks. Each cluster has a dedicated master-node to run Hadoop master services, which include NameNode, JobTracker, HBase Master, and Zookeeper. The master-node is always a physical node, no VM(s).

First cluster is composed of one master-node and three physical slave nodes. Each slave node runs DataNode, and TaskTracker or RegionServer. For DFSIO TaskTrackers are configured to have from 4 to 24 map slots per node and 1 reduce slot.

Physical slaves use hard drives as local storage. Total of 12 drives on the cluster.

Second cluster also has a master-node, but the three slave nodes run VM(s). We experimented with 2, 3, and 4 VM(s) per node. Each VM behaves the same way as a physical slave from the Hadoop cluster administering perspective. The difference is that each VM has less memory and restricted by ¼ of the node CPU capacity. That is in case of 4 VM(s) per node NameNode, JobTracker, and HBaseMaster see 12 DataNodes, TaskTrackers, and RegionServers, respectively, but the aggregate compute power of the virtualized cluster is the same as of the physical-node one.

VM(s) use Memory Arrays as local storage. Each VM is connected to a dedicated Array volume. With a total of 12 volumes used on the cluster.

Virtualization serves two main functions:

1. **Resource utilization** by running more server processes per node.
2. **Resource isolation** by designating certain percentage of resources to each server and not letting them starve each other.

4 DFSIO Benchmark

The primary goal of DFSIO benchmarks is to saturate flash arrays with random read workloads in order to determine a cluster configuration optimal to achieving maximum performance out of the Memory Arrays.

The DFSIO benchmark measures average throughput for read and write operations. DFSIO is a MapReduce program, which reads/writes random data from/to large files. Each map task within the job executes the same operation on a distinct file, transfers the same amount of data, and reports its transfer rate to the single reduce task. The reduce task then summarizes the measurements. The benchmark runs without contention from other applications, and the number of map tasks is chosen to be proportional to the cluster size. It is designed to measure performance only during data transfers, and excludes the overheads of task scheduling, start up, and the reduce task.

The standard DFSIO benchmark for Hadoop 1.0 supports only sequential reads and writes. We ran both of them on both clusters.

In addition to the standard DFSIO functionality three new variants of DFSIO were developed: Random, Backward, and Skip Reads.

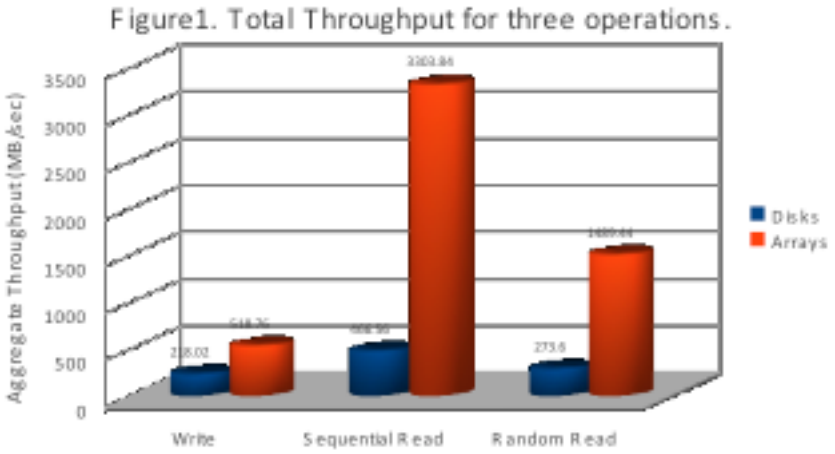
1. *Random Read DFSIO* randomly chooses an offset in a big file and then reads the specified number of bytes (1 MB in our experiments) from that offset.
2. *Backward Read DFSIO* reads files in reverse order. This is to avoid the read-ahead advantage and to prevent from scanning the same bytes twice.
3. *Skip Read DFSIO* reads the specified portion of the input file, then seeks ahead and then reads again. The jump between the reads is supposed to be large enough to avoid data read-ahead by the underlying local file system.

All three variants are intended to measure random read performance of HDFS. The new benchmarks reveal similar performance if the parameters are chosen reasonably. We will further refer to either of them as Random Read DFSIO benchmark. Benchmarks were executed on a large enough dataset to avoid system caching so that they measured IO(s) rather than memory copy. We discarded the system buffer cache before each run in order to avoid reading data cached on the previous runs. We showed that turning off read-ahead feature increases the performance of the Arrays for random reads.

For the cluster that uses Memory Array volumes as local storage read-ahead was disabled. The benchmarks were run with varying number of threads, slots, from 3 threads – one per node, and up to 72 threads. The dataset consisted of 72 files of size 9600 MB each. Sequential read benchmark reads whole files. In random read benchmarks each thread reads a total of 6400 MB of the dedicated file, with a single read operation reading a 1 MB chunk of data.

4.1 DFSIO Results

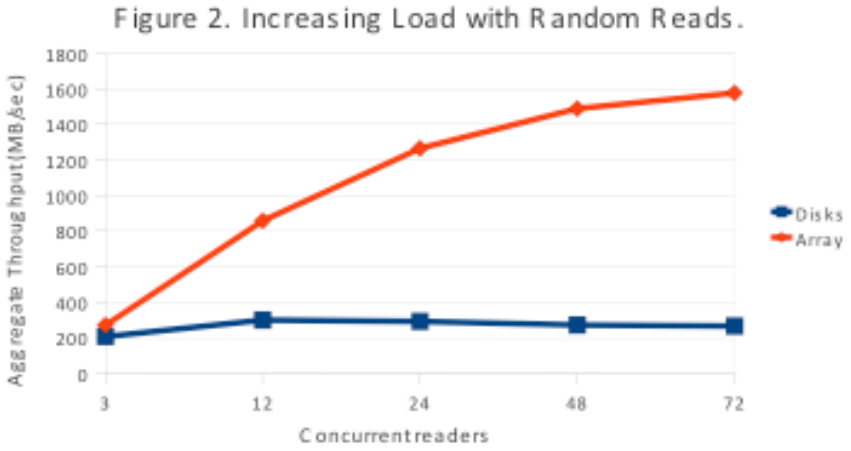
The results for maximum aggregate cluster throughput for each of the three operations: write, sequential read, and random read are summarized in Figure 1.



The results show that the aggregate cluster throughput for Memory Arrays is higher

- 2.5 times on writes
- 7 times on sequential reads
- 6 times on random reads

When the load on the cluster increases with more threads doing random reads, the cluster with Disks gets quickly saturated peaking at 300 MB/sec, while the Arrays aggregate throughput keep increasing to 1600 MB/sec. See Figure 2.



Our benchmarks showed that unlike conventional storage devices Violin Arrays cannot be saturated using single node running multiple I/O threads or processes or even multiple VM(s) on that single physical node.

- The benchmarks revealed that running variable number of threads on three or less nodes does not provide enough I/O load on the Array. The maximum aggregate throughput we could get is 1.5 GB/sec, compared to the Array throughput capacity of 4.8 GB/sec.
- We believe that increasing the node count on the cluster will raise the throughput up to the maximum.
- We estimate an optimal cluster size for this workload is 9-12 physical slaves.

- We ran DFSIO benchmarks on virtualized and physical clusters both using Arrays as local storage, and have not seen any significant improvements in saturating flash Arrays as a result of running more VM(s) per node.

Overall Violin Arrays perform substantially better than disks. The results show 5-8 times better random read throughput on Arrays compared to disk drives.

5 YCSB Benchmark

The purpose of the benchmark is to show that using multiple VM(s) per node utilizes the node resources in larger extent under the assumption that the system is not I/O bound as should be the case with Violin flash Arrays.

The YCSB benchmark allows one to define a mix of read / write operations and measure the performance in terms of latency and throughput for each operation included in the workload. The data is semantically represented as a table of records with a certain amount of fields representing record values and a key unique identifying each record.

YCSB supports four operations:

1. *Insert*: Insert a new record.
2. *Read*: Read a record.
3. *Update*: Update a record by replacing the value of one field.
4. *Scan*: Scan a random number of records in order, starting at a randomly chosen record key.

We apply YCSB benchmark to evaluate HBase performance, although the YCSB framework allows one to compare performance for a variety of databases.

Table 1 summarizes the types of workloads that were chosen for benchmarking.

Workload	Insert %	Read %	Update %	Scan %
Data Load	100			
Short range scans: workload E	5			95
Reads with heavy insert load	55	45		
Scans with heavy insert load	55			45
Scans with heavy update load			55	45

Table 1. YCSB Workloads.

We used varying number of threads generating the workload. Smaller numbers of threads provide very light utilization of cluster resources especially for physical nodes. On the other hand very large thread count hits the limitations of YCSB implementation: the YCSB client can run out of memory in such cases.

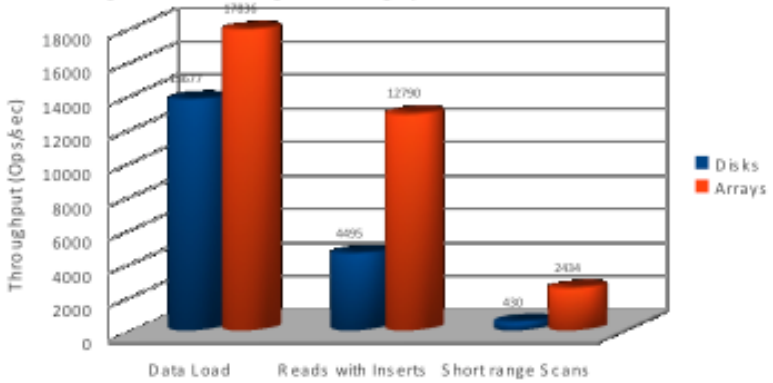
We experimented with different number of VM(s) on a physical node, running from 2 to 4 VM(s) per node. Total up to 12 VM(s) on the three-node cluster. We ran YCSB for datasets of two different sizes. One of 10 million records and another 3 times larger set of 30 million records. While running the benchmarks we ran dstat on every node to collect system level resource statistics such as CPU and memory usage, disk and network stats.

Below we present results for the first three workloads: load, predominant reads, and reads with inserts. While running the other two workloads on heavy volume we encountered problems with HBase that did not allow the benchmark to complete on the physical cluster, while the virtual cluster completed successfully. We decided to abandon the two latter workloads, even though the cases are interesting from practical view point, because there are no comparison points.

5.1 YCSB Results

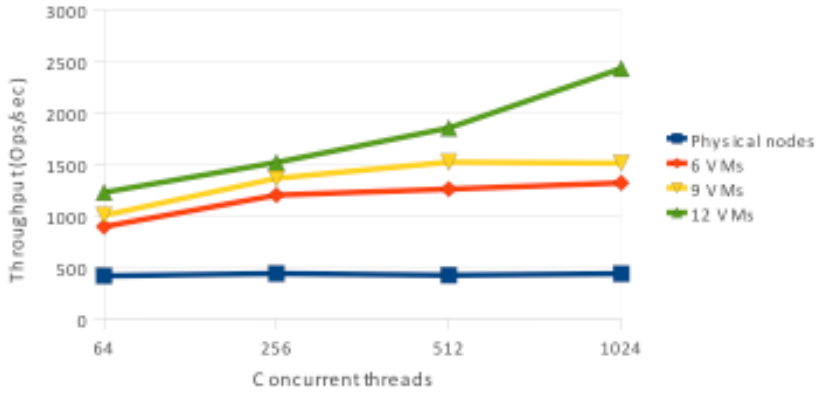
The throughput for the three workloads is summarized in Figure 3.

Figure 3. Average Throughput for three Workloads



For the *write-only workload* (100% inserts) we see that Arrays are 30% better than Disks. As expected the gain is not big. The main advantages should be seen for reads. But it is important that Arrays delivered better performance on writes.

Figure 4.1. Short range Scans. Throughput

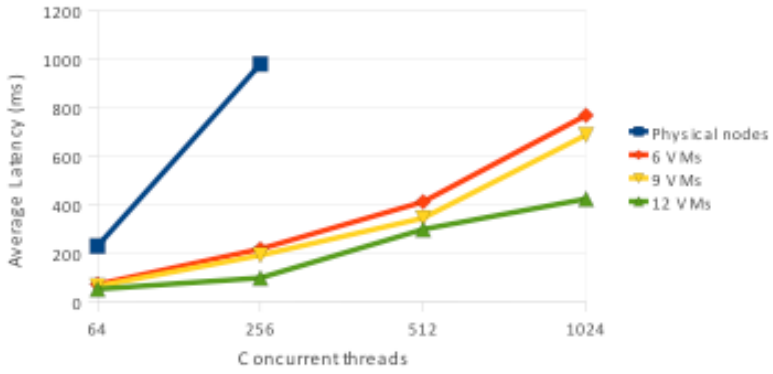


YCSB Predominant Reads

YCSB workload E generates 95% random Short range Scans and 5% Inserts. These results were collected on the 10 million record dataset. On the two clusters we ran the same workload with different number of threads from 64 to 1024 and different number of VM(s) per node, see Figure 4.

Figure 4.2 summarizes latency of scans and shows strong correlation of latency with throughput. We intentionally did not plot latency for physical node cluster for 512 and 1024 threads, because the latency doubles for each of these points and goes to 4,418 ms at 1024 threads, which makes all lower values unrecognizable.

Figure 4.2. Short range Scans. Latency



We see that virtualized cluster scales the throughput when the load increases, while the physical cluster shows essentially flat performance. With varying number of VM(s) per node, the performance is gradually increasing.

- Every step of adding one VM per node increases the overall performance on the scan benchmark about 20% on average for the busiest VM of the cluster.
- Under low load (64 YCSB threads) it doesn't matter how many VM(s) we run per node. The throughput and latency remain the same.
  The advantage of more VM(s) per node is obvious under heavy load (1024 YCSB threads).

We observed that with certain high enough number of threads the virtual cluster performance maxes out, so that HBase cannot yield higher throughput, which results in latency dropping down proportionally. The threshold is higher with more VM(s) per node.

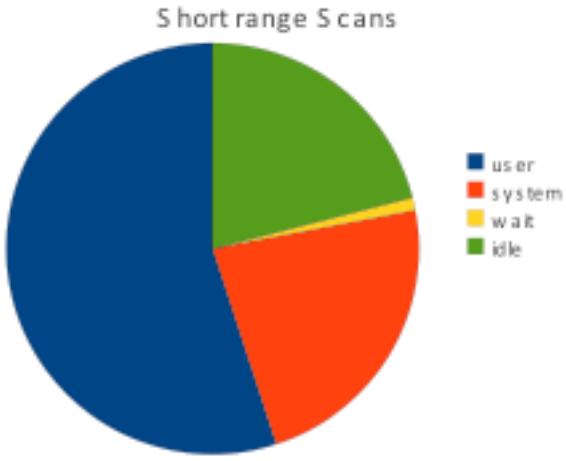
5.2 CPU Utilization

Dstat was used to collect system level statistics. CPU utilization was our main focus.

Physical three-node cluster generates very light CPU load on the nodes with 92% idle on average, see Figures 5.1 and 5.2. While with VM(s) the CPU can be drawn close to 100% at peaks, resulting in better cluster performance.

Figure 5.1. CPU Physical node cluste

Figure 5.2. CPU Virtualized cluster



- On the physical node cluster with standard configuration parameters we see very low memory utilization (2%) and low utilization of other resources.
- With optimized configuration the CPU on physical node cluster CPU utilization goes up, but still remains under 4% usr, 3% sys.
- 6, 9, and 12 VM cluster has much higher utilization, which is also reflected in overall execution-time throughput and latency.
- With 12 VM(s) (4 per node) and 1024 threads average CPU utilization goes up to 55% usr, 23% sys on the busiest VM of the cluster. The peak total CPU in the experiment with optimized configuration is reaching 99%.

Similar problems with CPU utilization for physical nodes using SSD(s) were observed in the community and reported in the following blog.

http://hadoopblog.blogspot.com/2012/05/hadoop-and-solid-state-drives.html

Internal locking prevents HBase from using more CPU, which leads the author to the conclusion that HBase "will not be able to harness the full potential that is offered by SSD(s)". The author did not go the rout of splitting physical nodes into virtual machines acting as independent nodes. And therefore could not observe the advantages of the approach.

Overall we see that setting up multiple VM(s) with attached Violin Array volumes provides higher utilization of the node resources and improves the predominant read performance 6 - 7 times both throughput- and latency-wise compared to three physical node cluster.

5.3 YCSB Reads mixed with Writes

The purpose of the benchmark is to evaluate random read performance of Violin flash Arrays when read workload is mixed with substantial write load.

5.3.1 45% scans, 55% updates

We consider it the most natural benchmark to show read performance benefits. Unfortunately this benchmark breaks under heavy load. The condition is known in HBase community, discussed in https://issues.apache.org/jira/browse/HBASE-4890

To summarize the problem, the benchmark fails at the very end when all the updates are flushed into HBase Servers, causing multiple timeouts in the RPC layer and failure of YCSB operations. Increasing RPC timeout is not a typical use case. We decided to avoid it, restricting the run to rather light workload of 16 YCSB threads, on which flash arrays outperformed disks about 30%.

5.3.2 2. 45% scans, 55% inserts

Table scans provide higher load on HBase than row reads. The heavy load version, 1024 threads, of the benchmark successfully completed on virtualized cluster, but failed about 70% through on the physical node cluster because of timeouts and lease expirations due to insufficient parallelization of the resources.

5.3.3 3. 45% reads, 55% inserts

Running 45% reads and 55% inserts allowed us to reach high load on both systems without crashing them. With this workload Violin Arrays performed 3 times better than drives in overall throughput (Figure 6.1) and read latency (Figure 6.2).

Figure 6.1. Reads with Inserts. Throughput

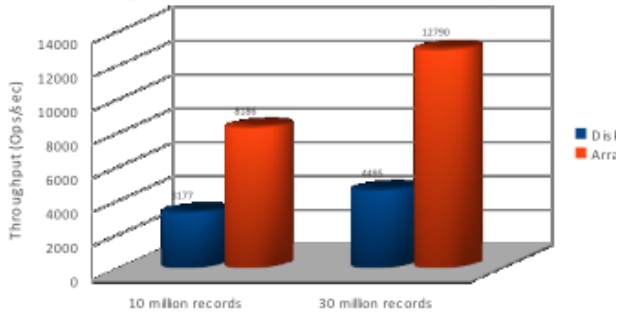
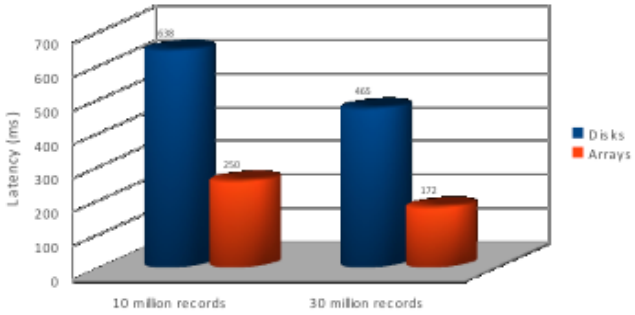


Figure 6.2. Reads with Inserts. Latency



The same ratio is confirmed on smaller 10 million records dataset, and the larger one of 30 million records. As with predominant read workload we see higher CPU utilization on virtualized cluster. The difference is less drastic because write load involves more processing on HBase side than reading. See Figures 7.1 and 7.2.

Figure 7.1. CPU Physical node cluster.

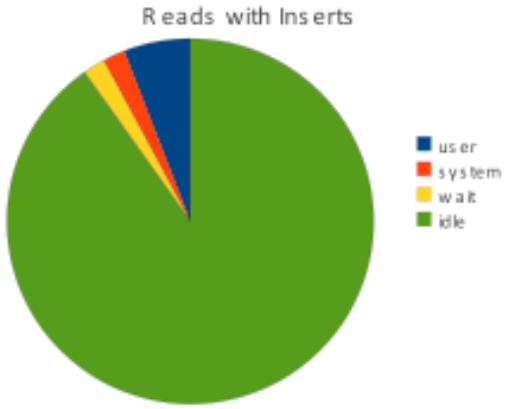


Figure 7.2. CPU Virtualized cluster

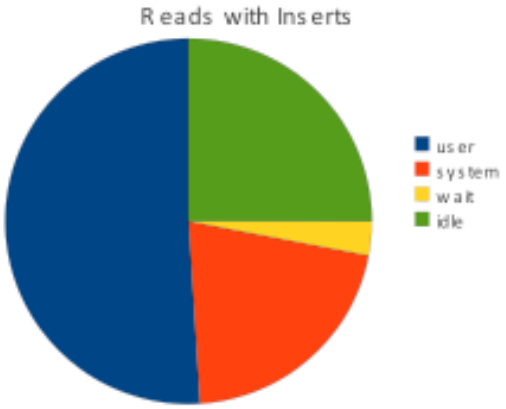


Figure 8.1 summarizes utilization of individual local storage devices per disk and per ISCSI LUN for Arrays. This shows that for the same workloads disk drives are utilized 72% on average and almost to their capacity at peaks, while Array LUNS use only 1/5 of their capacity on average.

Figure 8.1. Local Storage Devices

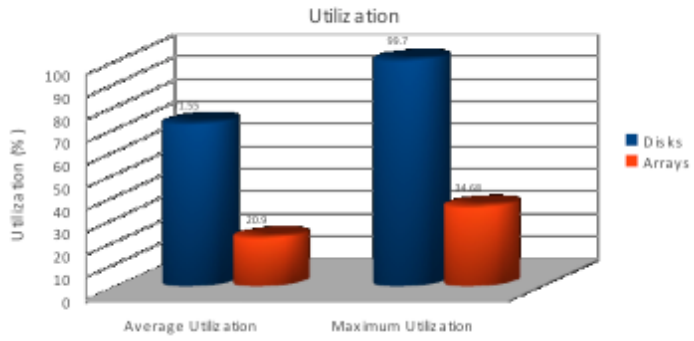
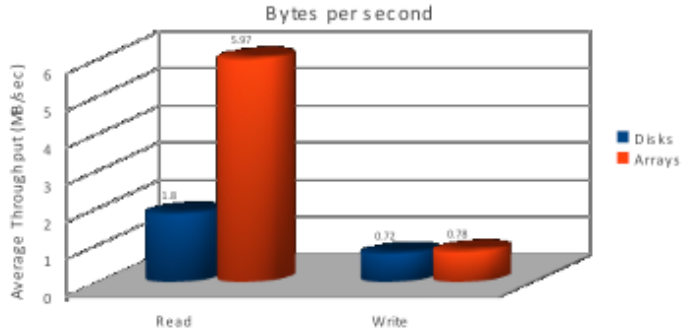


Figure 8.2 summarizes the number of bytes per second read or written to disk drive or an Array LUN. We see that Arrays performance is marginally better for writes, but 3.5 times higher for reads.

Figure 8.2. Local Storage Devices



6 Conclusion

These benchmarks confirm superior performance of Violin flash Arrays under a variety of workloads provided by Hadoop and HBase. Random reads and reads mixed with writes are the workloads handled by Violin Arrays exceptionally well. Writes are marginally better with arrays.

Our findings showed that the read-ahead feature is not beneficial for Violin as it slows down random reads.

Running multiple VM(s) as cluster nodes increases utilization of the physical node resources and stresses the system with more load, making it a preferred way of using Violin flash Arrays for Big Data systems.

The following is the summary of the main results of the accomplished benchmarking

DFSIO benchmark results show 5-8 times better *random read throughput* on Violin Memory Arrays compared to disk drives.

For YCSB *write-only workload* we see that Arrays are 30% better than Disks.

Setting up multiple VM(s) with attached Violin Array volumes provides close to *100% utilization* of the node resources – CPU and local storage IO, while on the same size physical node cluster with local disks the resource utilization is only a fraction of the capacity.

The latter improves the *predominant read* performance 6 - 7 times on Violin Arrays both *throughput*- and *latency*-wise compared to physical node cluster.

For *mixed read/write workloads* the same setup makes Violin Arrays performed 3 times better than drives in overall read throughput and latency.

About Violin Memory

Violin Memory is pioneering a new class of high-performance flash-based storage systems that are designed to bring storage performance in-line with high-speed applications, servers and networks. Violin Flash Memory Arrays are specifically designed at each level of the system architecture starting with memory and optimized through the array to leverage the inherent capabilities of flash memory and meet the sustained high-performance requirements of business critical applications, virtualized environments and Big Data solutions in enterprise data centers. Specifically designed for sustained performance with high reliability, Violin's Flash Memory Arrays can scale to hundreds of terabytes and millions of IOPS with low, predictable latency. Founded in 2005, Violin Memory is headquartered in Mountain View, California.

For more information about Violin Memory products, visit www.vmem.com.

Violin MEMORY

Violin Memory, Inc.
685 Clyde Ave, Mountain View, CA 94043
Ph: 1-888-9VIOLIN (984-6546)
Email: sales@vmem.com