The Truth About MapReduce Performance on SSDs

Karthik Kambatla[†‡]*, Yanpei Chen*[†]
{*kasha, yanpei*}*@cloudera.com*
[†]*Cloudera Inc.*
[‡]*Dept. of Computer Science, Purdue University.*

Abstract

Solid-state drives (SSDs) are increasingly being considered as a viable alternative to rotational hard-disk drives (HDDs). In this paper, we investigate if SSDs improve the performance of MapReduce workloads and evaluate the economics of using PCIe SSDs either in place of or in addition to HDDs. Our contributions are (1) a method of benchmarking MapReduce performance on SSDs and HDDs under constant-bandwidth constraints, (2) identifying *cost-per-performance* as a more pertinent metric than *cost-per-capacity* when evaluating SSDs versus HDDs for performance, and (3) quantifying that SSDs can achieve up to 70% higher performance for 2.5x higher cost-per-performance.

Keywords:. MapReduce, Analytics, SSD, flash, performance, economics.

1 Introduction

Solid-state drives (SSDs) are increasingly used for a variety of performance-critical workloads, thanks to their low latency (lack of seek overheads) and high throughput (bytes-per-second and IOPS). The relatively high *cost-per-capacity* of SSDs has limited their use to smaller datasets until recently. Decreasing prices [16] and low power-consumption [13] make them a good candidate for workloads involving large volumes of data. The lack of seek overhead gives them a significant advantage over traditional hard disk drives (HDDs) for random-access workloads such as those in key-value stores. The gains are less clear for sequential access workloads.

In this paper, we investigate the economics of using SSDs to improve the performance of MapReduce [8], a widely-used big-data analytics platform. As MapReduce represents an important software platform in the datacenter, the performance tradeoffs between SSDs and HDDs for MapReduce offers critical insights for designing both future datacenter server architectures and future big-data application architectures.

MapReduce is traditionally considered to be a sequential access workload. A detailed examination of the MapReduce IO pipeline indicates that there are IO patterns that benefits from the hardware characteristics of SSDs. Past studies on MapReduce SSD performance have not yet accurately quantified any performance gains, mostly due to previous hardware limits constraining studies to be simulation based, on unrealistic virtualized environments, or comparing HDD and SSD setups of different bandwidths (Section 2).

Our MapReduce benchmarking method seeks to compare HDDs and PCIe SSDs under constant bandwidth constraints (Section 3). We selected our hardware to answer the following questions: (1) when setting up a new cluster, how do SSDs compare against HDDs of same aggregate bandwidth, and (2) when upgrading an HDDs-only cluster, should one add SSDs or HDDs for better performance. We measured performance for a collection of MapReduce jobs to cover several common IO and compute patterns.

Our results quantify the MapReduce performance advantages of SSDs and help us identify how to configure SSDs for high MapReduce performance (Section 4). Specifically, we find that

1. For a new cluster, SSDs deliver up to 70% higher MapReduce performance compared to HDDs of equal aggregate IO bandwidth.
2. Adding SSDs to an existing HDD cluster improves performance if configured properly. SSDs in hybrid SSD/HDD clusters should be divided into multiple HDFS and shuffle local directories.

Beyond the immediate HDD versus SSD tradeoffs, a broader implication of our study is that the choice of storage media should consider *cost-per-performance* in addition to the more common metric of *cost-per-capacity* (Section 5). As a key benefit of SSDs is performance, one can argue that cost-per-performance is the more important metric. Our results indicate SSDs have 2.5x higher cost-per-performance for MapReduce, while delivering up to 70% higher performance. This gap is far smaller than the orders-of-magnitude difference in cost-per-capacity.

2 Background and Related Work

2.1 SSDs vs HDDs

The biggest advantage of SSDs over HDDs is the high IOPS. SSDs achieve this by avoiding the physical disk rotation and seek time. The sequential IO bandwidth is

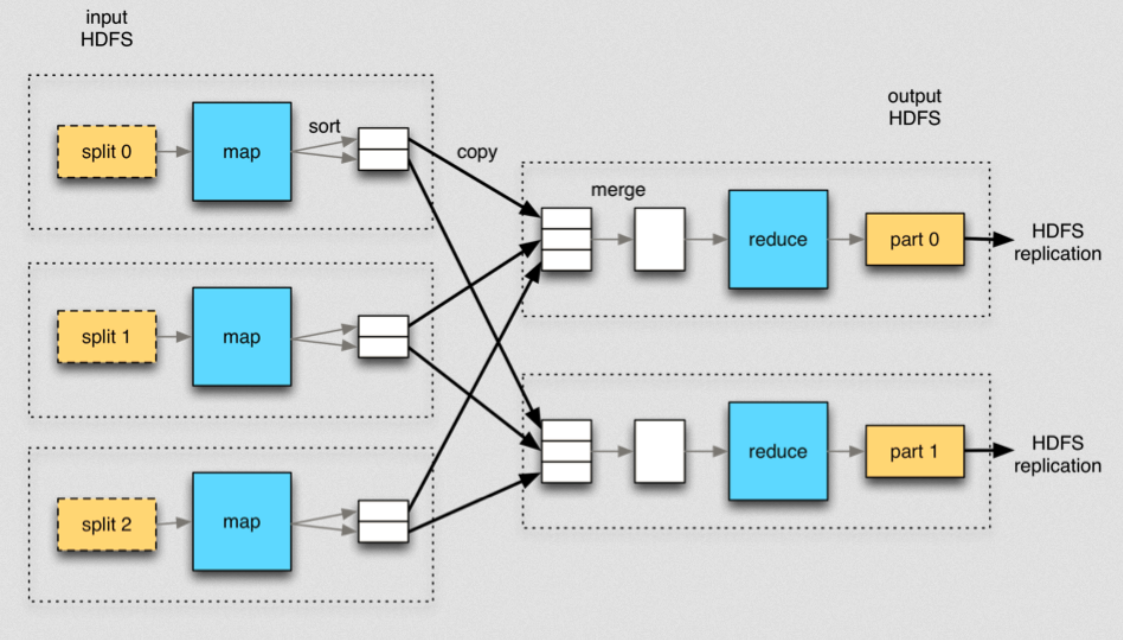


Figure 1: MapReduce Dataflow. Source: [19]

also higher: by measuring the time taken to copy a large file, we found our HDDs could each support ∼120MBps of sequential read or write, while our SSDs were each capable of ∼1.3GBps sequential read or write.

The performance benefits of SSD compared to HDDs depend on the workload. For sequential I/O workloads, one can use multiple HDDs in parallel (assuming the application allows parallel access) to extract bandwidth comparable to SSD. For example, one can use 10 HDDs of 120MBps to match the 1GBps bandwidth of an SSD. On the other hand, realizing comparable bandwidth for random I/O workloads can be far more expensive, as one would need many more HDDs to offset the seek latency. For example, if an HDD delivers an effective bandwidth of 10 MBps for random accesses of a few MBs of data each IO, one needs to use at least 100 of them to achieve the same 1GBps aggregate bandwidth.

2.2 MapReduce – Dataflow

MapReduce [8] is a data-parallel processing framework designed to process large volumes of data in parallel on clusters of machines. In Apache Hadoop [1], a widely-used open-source MapReduce implementation, the execution is split into map, shuffle, reduce phases. Map and reduce phases are split into multiple tasks, each task potentially running on a different machine. Each map task takes in a set of key-value pairs ($< key, value >: list$), applies the map function to each pair and emits another set of key-value pairs ($< key, value >: list$). The shuffle phase partitions the map output from all map tasks such that all values corresponding to a key are in the same partition ($< key, value : list >: list$), each partition can be on a different node. Each reduce task picks these partitions, applies the reduce function per key, and writes the output to HDFS. Figure 1 captures the flow of data in a typical MapReduce job.

The effect of storage media, particularly SSD versus HDD, depends on the average read/write size and the randomness of data accesses. A typical MapReduce job exhibits two kinds of data accesses:

Large, sequential HDFS accesses. The job reads input splits from HDFS initially, and writes output partitions to HDFS at the end. Each task (dotted box) performs relatively long sequential IO of 100s of MBs. When multiple tasks are scheduled on the same machine, they can access the disks on the machine in parallel, with each task accessing its own input split or output partition. Thus, an HDD-only configuration of 11 disks of 120MBps each can potentially achieve HDFS read/write bandwidth comparable to a SSD drive of 1.3GBps.

Small, random reads and writes of shuffle intermediate data. MapReduce partitions each map output across all the reduce tasks. This leads to significantly lower IO size. For example, suppose a job has map tasks that each produces 1GB of output. When divided among, say, 1,000 reduce tasks, each reduce task fetches only 1MB. Analysis of our customer traces indicate that many deployments indeed have a per-reduce shuffle granularity

Table 1: Storage configurations used

Setup	Storage	Capacity	Sequential R/W Bandwidth	Price (USD)
HDD-6	6 HDDs	12 TB	720 MBps	2,400
HDD-11	11 HDDs	22 TB	1300 MBps	4,400
SSD	1 SSD	1.3 TB	1300 MBps	14,000
Hybrid	6 HDDs + 1 SSD	13.3 TB	2020 MBps	16,400

of just a a few MBs or even lower.

The number of concurrent accesses on a node determines the extent of I/O multiplexing on the disks, which in turn depends on the stage of job execution. The number of map or reduce tasks per node determines the number of concurrent HDFS read/write accesses. During the shuffle phase, the map-side sort IO concurrency is determined by the total number of merge-sort threads used across all map tasks on a node. The network copy concurrency comes from the number of map-side threads serving the map outputs and the number of reduce-side threads remotely fetching map outputs. The reduce-side sort IO concurrency is also determined by the number of merge-sort threads on the node. In practice, independent of IO concurrency, there is negligible disk I/O for intermediate data that fits in memory, while a large amount of intermediate data leads to severe load on the disks.

A further dimension to consider is compression of HDFS and intermediate data. Compression is a common technique to shift load from IO to CPU. Map output compression is turned on by default in Cloudera's distribution including Apache Hadoop (CDH), as most common kinds of data (textual, structured numerical) are readily compressible. Job output compression is disabled by default in CDH for compatibility with historical versions. Tuning compression allows us to examine tradeoffs in storage media under two different IO and CPU mixes.

Based on the MapReduce dataflow and storage media characteristics, we hypothesize that:

1. SSDs improve performance of shuffle-heavy jobs.
2. SSDs and HDDs perform similarly for HDFS-read-heavy and HDFS-write-heavy jobs.
3. For hybrid clusters (both SSDs and HDDs), using SSDs for intermediate shuffle data leads to significant performance gains.
4. All else equal, enabling compression decreases performance differences by shifting IO load to CPU.

2.3 Prior work

A body of work on SSD performance for MapReduce and other big data systems is still emerging. To date, progress on this area has been limited by the cost and (un)availability of SSDs.

An early work on HDFS SSD performance used OS buffer cache to simulate a fast SSD [4]. The study focused on Apache HBase [2] performance, and found various code path bottlenecks in HDFS that prevented the full potential of SSDs to be realized. Some of the bottlenecks have since been eliminated from HDFS.

Another study used real SSDs, but on a virtualized cluster, i.e., multiple virtualized Hadoop workers on a single physical machine [12]. The experiments found that Hadoop performs up to 3x better on SSDs. It remains unclear how the results translate to non-virtualized environments or environments where every virtualized node is located on a separate physical node.

A recent follow up to [4] simulated SSD performance as a tiered cache for HBase [9]. It found that under certain cost and workload models, a small SSD cache triples performance while increasing monetary cost by 5%.

The closest work to ours compared Hadoop performance on actual SSDs and HDDs [14], albeit on hardware with non-uniform bandwidth and cost. The study runs the *Terasort* benchmark on different storage configurations and found that SSD can accelerate the shuffle phase of the MapReduce pipeline, as we already hypothesized based on MapReduce IO characteristics.

Working with Cloudera's hardware partners, we designed our experiments to cover gaps in prior studies. We compare MapReduce performance on actual HDDs and SSDs, without virtualization, using storage hardware of comparable bandwidths, with MapReduce configurations optimized via experience at Cloudera's customers.

3 Experimental setup

Our choice of hardware and MapReduce benchmarks is guided by the following considerations:

- We compare SSDs vs HDDs performance under equal-bandwidth constraints. An alternative is to compare performance for equal-costs. We selected the equal-bandwidth setup because it reveals the intrinsic features of the technology without the impact of variable economic dynamics.
- We considered both SSDs/HDDs as storage for a new cluster, and SSDs/HDDs as additional storage

Table 2: Descriptions of MapReduce jobs

Job	Description
Teragen	HDFS write job, with 3-fold replication that heavily uses the network.
Terasort	Job with 1:1:1 HDFS read, shuffle, and HDFS write.
Teravalidate	HDFS read-heavy job that also does sort order validation (mostly HDFS read), with some small IO in shuffle and HDFS write.
Wordcount	CPU-heavy job that heavily uses the map-side combiner.
Teraread	HDFS read-only job, like Teravalidate, except no sort order validation and no reduce tasks.
Shuffle	Shuffle-only job, modified from randomtextwriter in hadoop-examples.
HDFS Data Write	HDFS write-only job, like Teragen, except with 1-fold replication.

Table 3: Data size and CPU utilization for the MapReduce jobs. Values are normalized against Terasort.

Job	Input size	Shuffle size	Output size	CPU utilization
Teragen	0	0	3	0.99
Terasort	1	1	1	1.00
Teravalidate	1	0	0	0.60
Wordcount	1	0.07	0.09	1.47
Teraread	1	0	0	1.23
Shuffle	0	1	0	1.01
HDFS Data Write	0	0	1	0.93

to enhance an existing HDD cluster. Both scenarios are relevant and of interest to enterprise customers.

- Our benchmark includes a series of MapReduce jobs to cover common IO and compute patterns seen in customer workloads. We deliberately deferred a more advanced method of measuring performance for multi-job workloads [5, 6]. The standalone, one-job-at-a-time method allows us to more closely examine MapReduce and storage media interactions without the impact of job scheduling and task placement algorithms.

3.1 Hardware

We used PCIe SSDs with 1.3TB capacity costing $14,000 each, and SATA HDDs with 2TB capacity costing $400 each. Each storage device is mounted with the Linux ext4 file system, with default options and 4KB block size. The machines are Intel Xeon 2-socket, 8-cores, 16-threads, with 10Gbps Ethernet and 48GB RAM. They are connected as a single rack cluster.

To get a sense of the user-visible storage bandwidth without HDFS and MapReduce, we measured the duration of copying a 100GB file to each storage device. This test indicates the SSDs can do roughly 1.3GBps sequential read and write, while the HDDs have roughly 120MBps sequential read and write.

Table 1 describes the storage configurations we evaluate. The SSD and HDD-11 setups allow us to compare SSDs vs HDDs on an equal-bandwidth basis. The HDD-6 setup serves as a baseline of IO-constrained cluster. The HDD-6, HDD-11, and Hybrid setups allow us to investigate the effects of adding either HDDs or SSDs to an existing cluster.

3.2 MapReduce jobs

Table 2 describes the MapReduce benchmark jobs that we use. Each is either a common benchmark, or a job constructed specifically to isolate a stage of the MapReduce IO pipeline.

More details on the jobs and our measurement method:

- Each job is set to shuffle, read, write, or sort 33GB of data per node.
- Where possible, each job runs with either a single wave of map tasks (Teragen, Shuffle, HDFS Data Write), or a single wave of reduce tasks (Terasort, Wordcount, Shuffle).
- We record average and standard deviation of job duration from five runs.
- We clear the OS buffer cache on all machines between each measurement.
- We used collectl to track IO size, counts, bytes, merges to each storage device, as well as network and CPU utilization.

Note that the jobs here are IO-heavy jobs selected and sized specifically to compare two different storage media. In general, real-world customer workloads have a

Table 4: Performance-relevant MapReduce 2 configurations.

Configuration	Value
mapreduce.task.io.sort.mb	256
mapreduce.task.io.sort.factor	64
mapreduce.task.io.sort.spill.percent	0.8
mapreduce.reduce.shuffle.parallelcopies	10
HDFS blocksize	128 MB
yarn.nodemanager.resource.memory-mb	RAM size
yarn.nodemanager.resource.cpu-vcores	# of CPU threads
mapreduce.map.memory.mb	1024
mapreduce.reduce.memory.mb	1024
mapreduce.map.cpu.vcores	1
mapreduce.reduce.cpu.vcores	1
mapreduce.map.java.opts	-Xmx1000
mapreduce.reduce.java.opts	-Xmx1000
yarn.scheduler.minimum-allocation-mb	256
mapreduce.job.reduce.slowstart.completedmaps	0.8
mapreduce.map.output.compress	both true and false
mapreduce.output.compress	false

variety of sizes and create load for multiple resources including IO, CPU, memory, and network.

Table 3 shows, for each job, the data size at a given stage of the MapReduce IO pipeline as well as the CPU utilization for the HDD-6 setup. The data in the table is normalized with Terasort as baseline. It quantifies the MapReduce job descriptions in Table 2.

3.3 MapReduce configurations

Our experiments are run on MapReduce v2 on YARN, which does not have the notion of map and reduce slots anymore. Map and reduce tasks are run within "containers" allocated by YARN. Most of the MapReduce configurations used in our tests come from the defaults in CDH5b1. Table 4 lists performance-related configurations and the values we used. These little-known parameters are often neglected in various studies, including SSD-related prior work [12, 14] and dedicated MapReduce configuration auto-tune systems [10].

Note that these configuration are intended to be performance safe. For each particular customer use case and hardware combination, we expect there is room for further tuning using the values here as a starting point. Further details about MapReduce performance tuning can be found in references such as [19, 18, 17].

4 Results

We present the results of our benchmarking in the context of these two questions: (1) for a new cluster, should one prefer SSDs or HDDs of same aggregate bandwidth, and (2) for an existing cluster of HDDs, should one add SSDs or HDDs.

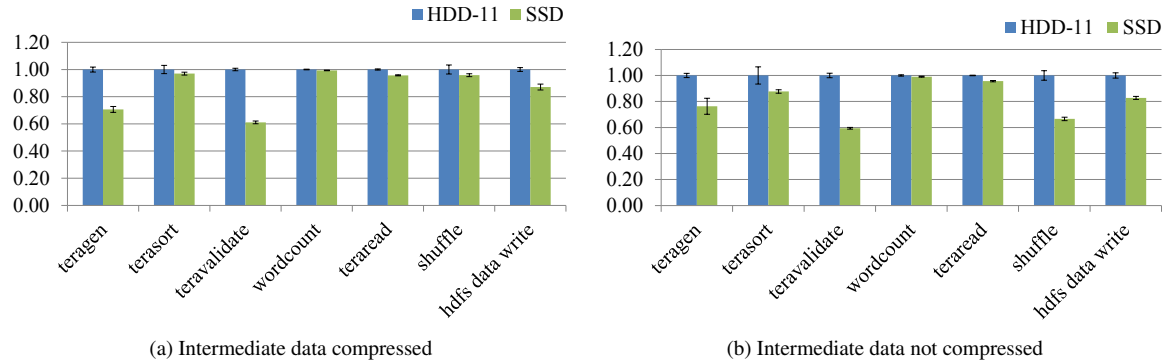
4.1 SSDs vs HDDs for a new cluster

Our goal here is to compare SSDs vs HDDs of the same aggregate bandwidth. Let us look at a straight-forward comparison between the SSD (1 SSD) and HDD-11 (11 HDDs) configurations. Figure 2 plots the job durations for the two storage options; the SSD values are normalized against the HDD-11 values for each job. The first graph shows results with intermediate data compressed, and the second one without.

General Trend. SSD is better than HDD-11 for all jobs, both with and without intermediate data compression. However, the benefits of using SSD vary across jobs.

Shuffle size determines the improvement due to SSDs. SSD does benefit shuffle, as seen in Terasort and Shuffle for uncompressed intermediate data. Shuffle read and write IO sizes are small compared to that of HDFS, as shown in Figure 3, and in agreement with our discussion of MapReduce IO patterns earlier.

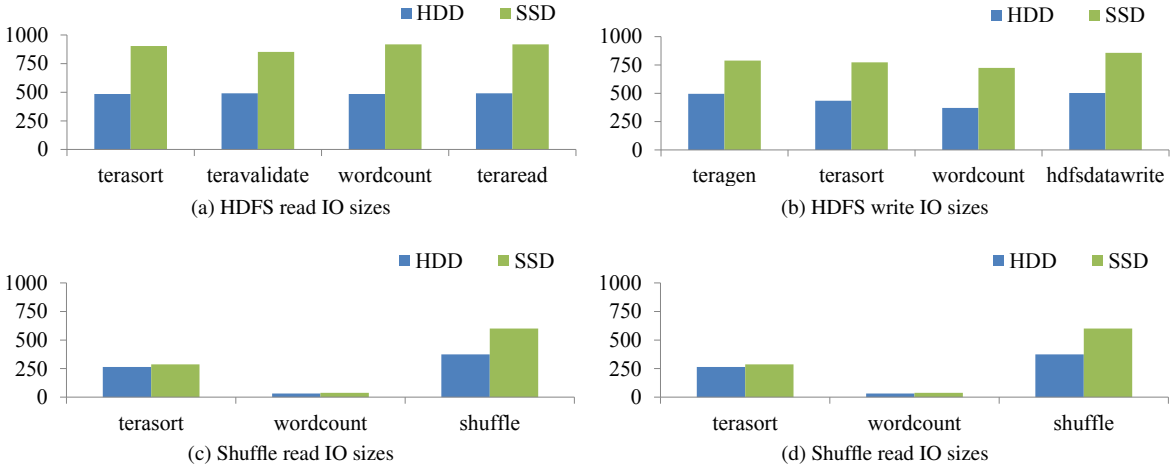
Map output compression masks any improvement due to SSDs. This is evident in the data for Terasort and Shuffle jobs in Figure 2. We believe this is due to shuffle data being served from buffer cache RAM instead of disk. The data in Terasort and Shuffle are both highly compressible, allowing compressed intermediate data to fit in the buffer cache. When we increase the data size per job 10x, the SSD benefits are visible even with compressed intermediate data.



(a) Intermediate data compressed

(b) Intermediate data not compressed

Figure 2: SSD vs HDD. Normalized job durations, lower is better.



(a) HDFS read IO sizes

(b) HDFS write IO sizes

(c) Shuffle read IO sizes

(d) Shuffle read IO sizes

Figure 3: SSD vs HDD read and write IO sizes for HDFS and shuffle data.

SSD also benefits HDFS read and write. A surprising result was that SSD also benefits HDFS read and write, as indicated by Teragen, Teravalidate, Teraread, and HDFS Data Write. Turns out that our SSD is capable of roughly 2x the sequential IO size of the hard disks (see Figure 3). Also, note that these jobs do not involve large amounts of shuffle data, so compressing intermediate data has no visible effect.

CPU heavy jobs not affected by choice of storage media. Wordcount is a CPU-heavy job that involves text parsing and arithmetic aggregation in the map-side combiner. The CPU utilization was higher than that for other jobs, and at 90% regardless of storage and compression configurations. As the IO path is not the bottleneck, the choice of storage media has little impact.

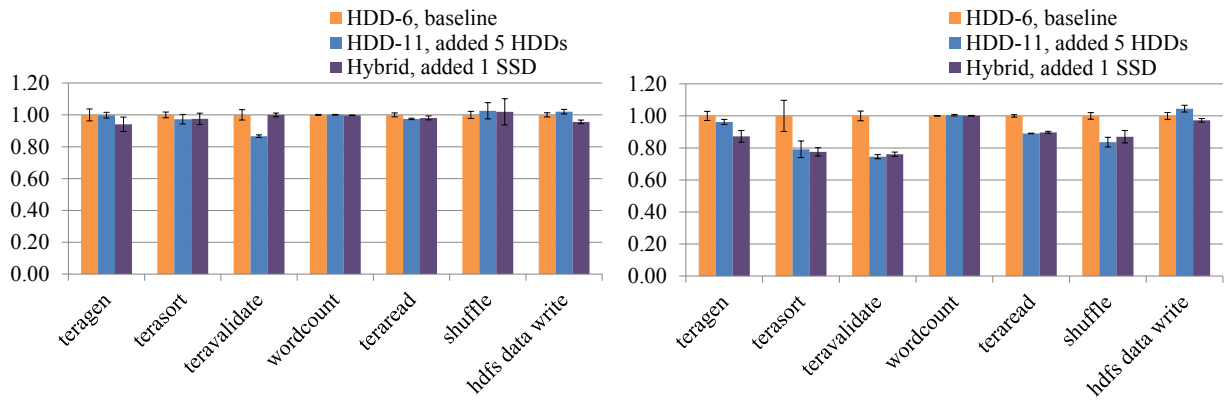
4.2 Adding SSDs to an existing cluster

Our goal here is to compare adding an SSD or many HDDs to an existing cluster, and to compare the various configurations possible in a hybrid SSD-HDD cluster.

We use a baseline cluster with 6 HDDs per node (HDD-6). Adding an SSD or 5 HDDs to this baseline results in the Hybrid and HDD-11 setups. On an equal bandwidth basis, adding one SSD should ideally be compared to adding 11 HDDs. Our machines do not have 17 disks; however, we believe the setups we have are enough to give us helpful insights as discussed below.

Default configurations - Hybrid cluster sees lower than expected benefit. Figure 4 compares job durations for the HDD-6, HDD-11, and Hybrid setups; intermediate data is compressed in the first graph and not compressed in the second. HDD-11 and Hybrid both give visible improvement over HDD-6. However, even with its additional hardware bandwidth (add 1 SSD vs. add 5 HDDs), the Hybrid setup leads to no improvement over HDD-11. This observation triggered further investigations as discussed below.

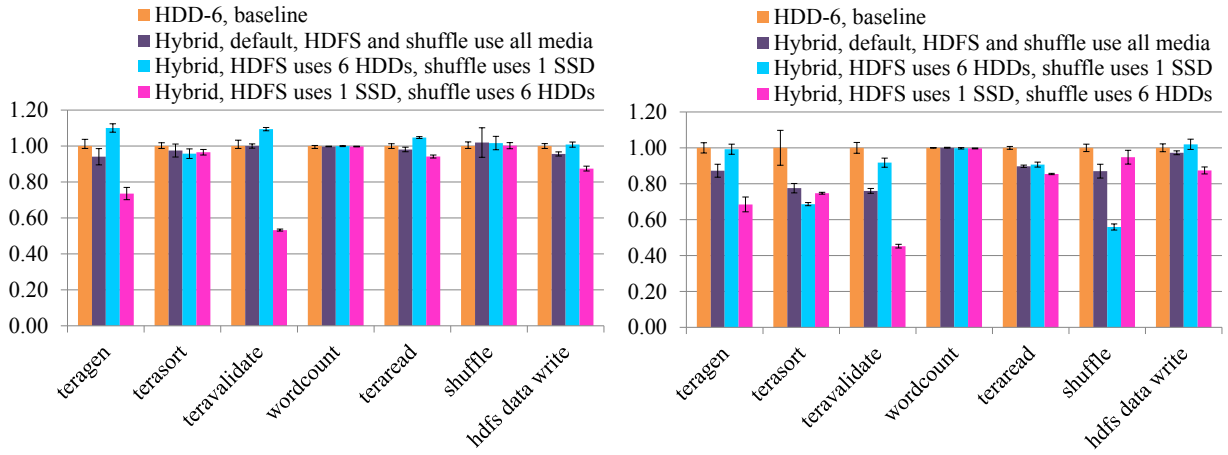
Hybrid - when HDFS and shuffle use separate storage media, benefits depend on workload. The default



(a) Intermediate data compressed

(b) Intermediate data not compressed

Figure 4: Add SSD/HDD to an existing HDD-6 cluster. Normalized job durations, lower is better.



(a) Intermediate data compressed

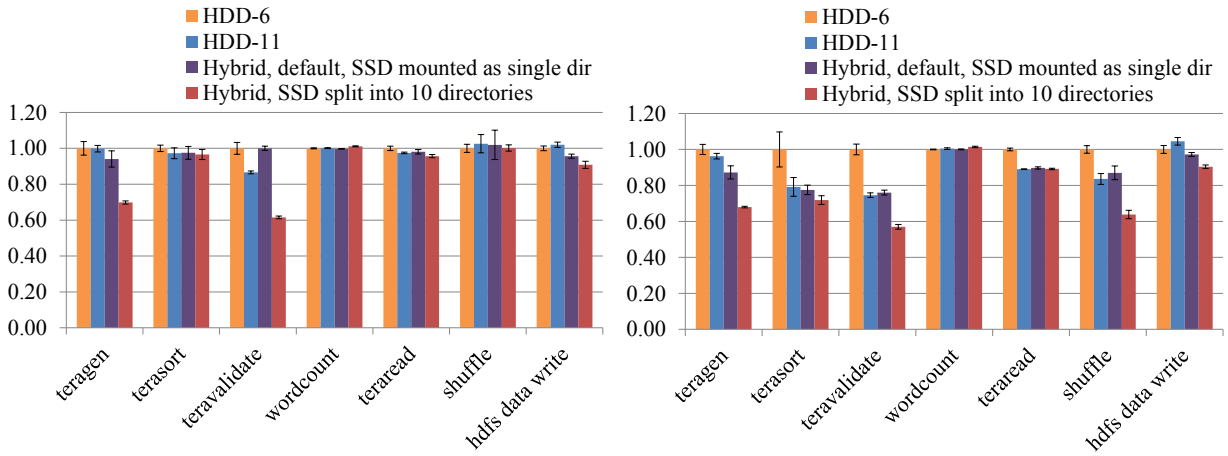
(b) Intermediate data not compressed

Figure 5: Hybrid modes. Normalized job durations, lower is better.

Hybrid configuration assigns HDDs and SSD to both the HDFS and shuffle local directories. We test whether separating the storage media gives any improvement. Doing so requires two more cluster configurations - HDDs for HDFS with SSD for intermediate data, and vice versa. Figure 5 captures the results of this experiment.

From the results, we see that the shuffle-heavy jobs (Terasort and Shuffle) benefit from assigning SSD completely to intermediate data, while the HDFS-heavy jobs see a penalty (Teragen, Teravalidate, Teraread, HDFS Data Write). We see the opposite when the SSD is assigned to only HDFS. This is expected, as the SSD has a higher bandwidth than 6 HDDs combined. However, one would expect the simple hybrid to perform half way between assigning SSD to intermediate data and HDFS. This led to the next set of tests.

Hybrid - SSD should be split into multiple local directories. A closer look at HDFS and MapReduce implementations reveals a critical point — both the DataNode and the NodeManager pick local directories in a round-robin fashion. A typical setup would mount each piece of storage hardware as a separate directory, e.g., /mnt/disk-1, /mnt/disk-2, /mnt/ssd-1. HDFS and MapReduce both have the concept of local directories; HDFS local directories store the actual blocks and MapReduce local directories store the intermediate shuffle data. One can configure HDFS and MapReduce to use multiple local directories, e.g, /mnt/disk-1 through /mnt/disk-11 plus /mnt/ssd-1 for our Hybrid setup. When writing the intermediate shuffle data, the NodeManager picks the 11 HDD local directories and the single SSD directory in a round-robin fashion. Hence, when the job is optimized for a single wave of map tasks, each local directory receives the same amount of data, and faster progress on



(a) Intermediate data compressed

(b) Intermediate data not compressed

Figure 6: Hybrid with 10 data directories on SSD. Normalized job durations, lower is better.

Table 5: Cost Comparison

Setup	Cost (US$)	Capacity	Bandwidth	US$ per TB	Cost per performance
Disk	400	2 TB	120 MBps	200	1x baseline
SSD	14,000	1.3 TB	1300 MBps	10,769	2.5x baseline

the SSD gets held up by slower progress on the HDDs.

So, to fully utilize the SSD, we need to split the SSD into multiple directories to maintain equal bandwidth per local directory. In our case, SSDs should be split into 10 directories. In our single-wave map output example, the SSDs would then receive 10x the data directed at each HDD, written at 10x the speed, and complete in the same amount of time. Note that splitting the SSD into multiple local directories improves performance, but the SSD will fill up faster than the HDDs.

Figure 6 shows the performance of the split-SSD setup, compared against the HDD-6, HDD-11, and Hybrid-default setups. Splitting SSD into 10 local directories invariably leads to major improvements over the default Hybrid setup.

5 Implications and Conclusion

Choice of storage media should also consider cost-per-performance. Our findings suggest SSD has higher performance compared to HDD-11. However, from an economic point of view, the choice of storage media depends on the cost-per-performance for each.

This differs from the cost-per-capacity metric ($-per-TB) that appears more frequently in HDD vs SSD comparisons [16, 15, 11]. Cost-per-capacity makes sense for capacity-constrained use cases. As the primary benefit of SSD is high performance rather than high capacity, we believe storage vendors and customers should also track $-per-performance for different storage media.

From our tests, SSDs have up to 70% higher performance, for 2.5x higher $-per-performance (Table 5, average performance divided by cost for the SSD and HDD-11 setups). This is far lower than the 50x difference in $-per-TB. Customers can consider paying a premium cost to obtain up to 70% higher performance.

One caveat is that our tests focus on equal aggregate bandwidth for SSDs and HDDs. An alternate approach is to compare setups with equal cost. That translates to 1 SSD against 35 HDDs. We do not have the necessary hardware to test this setup. However, we suspect the performance bottleneck likely shifts from IO to CPU for our hardware. The recommended configuration is 2 containers per core for MR2, and roughly one container per local directory. On our hardware of 8 cores, having 35 HDDs means either there would not be not enough containers to keep all disks occupied, or there would be too many containers that the CPUs are over-subscribed.

Choice of storage media should also consider the targeted workload. Our tests here show that SSD benefits vary depending on the MapReduce job involved. Hence, the choice of storage media needs to consider the aggregate performance impact across the entire production workload. The precise improvement depends on how compressible the data is across all datasets, and the ratio of IO versus CPU load across all jobs.

Future work. MapReduce is a crucial component of Enterprise data hubs (EDHs) that enable data to be ingested, processed, and analyzed in many ways. To fully understand the implications of SSDs for EDHs, we need to study the tradeoffs for other components such as HBase, SQL-on-HDFS engines such as Impala [7], and enterprise search platforms such as Apache Solr [3]. These components are much more sensitive to latency and random access. They aggressively cache data in memory, and cache misses heavily affect performance. SSDs could potentially act as a cost-effective cache between memory and disk in the storage hierarchy. We need measurements on real clusters to verify.

Overall, SSD economics involves the interplay between ever-improving software and hardware, as well as ever-evolving customer workloads. The precise trade-off between SSDs, HDDs, and memory deserves regular reexamination over time.

References

[1] Apache Software Foundation. Apache Hadoop. http://hadoop.apache.org, .

[2] Apache Software Foundation. Apache HBase. http://hbase.apache.org, .

[3] Apache Software Foundation. Apache Solr. http://lucene.apache.org/solr/, .

[4] D. Borthakur. Hadoop and solid state drives. Blog, http://hadoopblog.blogspot.com/2012/05/hadoop-and-solid-state-drives.html.

[5] Y. Chen, S. Alspaugh, A. Ganapathi, R. Griffith, and R. Katz. Statistical workload injector for mapreduce. https://github.com/SWIMProjectUCB/SWIM/wiki, .

[6] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The Case for Evaluating MapReduce Performance Using Workload Suites. In *MASCOTS 2011*, .

[7] Cloudera Inc. Cloudera Impala. http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html.

[8] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI 2004*.

[9] T. Harter, D. Borthakur, S. Dong, A. Aiyer, L. Tang, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Analysis of hdfs under hbase: A facebook messages case study. In *FAST 2014*.

[10] H. Herodotou and S. Babu. Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. In *VLDB 2011*.

[11] J. Janukowicz. Worldwide solid state drive 20132017 forecast update. IDC Research Report, 2013.

[12] S.-H. Kang, D.-H. Koo, W.-H. Kang, and S.-W. Lee. A case for flash memory ssd in hadoop applications. *International Journal of Control and Automation*, 6(1), 2013.

[13] T. Kgil, D. Roberts, and T. Mudge. Improving nand flash based disk caches. In *ISCA 2008*.

[14] J. Lee, S. Moon, Y. suk Kee, and B. Brennan. Introducing SSDs to the Hadoop MapReduce Framework. In *Non-Volatile Memories Workshop 2014*.

[15] J. Monroe and J. Unsworth. Market Trends: Evolving HDD and SSD Storage Landscapes. Gartner Analyst Report, 2013.

[16] PriceG2 Research Report. When Will SSD Have Same Price as HDD. http://www.priceg2.com/.

[17] S. Ryza. Getting MapReduce 2 Up to Speed. Cloudera blog, 2014. http://blog.cloudera.com/blog/2014/02/getting-mapreduce-2-up-to-speed/, .

[18] S. Ryza. Migrating to MapReduce 2 on YARN (For Operators). Cloudera blog, 2013. http://blog.cloudera.com/blog/2013/11/migrating-to-mapreduce-2-on-yarn-for-operators/, .

[19] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009. ISBN 0596521979, 9780596521974.