J Supercomput (2015) 71:3525–3548
DOI 10.1007/s11227-015-1447-3

Optimizing the Hadoop MapReduce Framework with high-performance storage devices

Sangwhan Moon[1] · Jaehwan Lee[2] · Xiling Sun[3] · Yang-suk Kee[3]

Published online: 29 May 2015

Abstract Solid-state drives (SSDs) are an attractive alternative to hard disk drives (HDDs) to accelerate the Hadoop MapReduce Framework. However, the SSD characteristics and today's Hadoop framework exhibit mismatches that impede indiscriminate SSD integration. This paper explores how to optimize a Hadoop MapReduce Framework with SSDs in terms of performance, cost, and energy consumption. It identifies extensible best practices that can exploit SSD benefits within Hadoop when combined with high network bandwidth and increased parallel storage access. Our Terasort benchmark results demonstrate that Hadoop currently does not sufficiently exploit SSD throughput. Hence, using faster SSDs in Hadoop does not enhance its performance. We show that SSDs presently deliver significant efficiency when storing intermediate Hadoop data, leaving HDDs for Hadoop Distributed File System (HDFS). The proposed configuration is optimized with the JVM reuse option and frequent heartbeat interval option. Moreover, we examined the performance of a state-

✉ Jaehwan Lee
jlee@kau.ac.kr

Sangwhan Moon
sangwhan@tamu.edu

Xiling Sun
xiling.sun@ssi.samsung.com

Yang-suk Kee
yangseok.ki@samsung.com

[1] Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77840, USA

[2] School of Electronics and Information Engineering, Korea Aerospace University, Goyang-si, Republic of Korea

[3] Advanced Datacenter Solution Group, Samsung Semiconductor Incorporation, Milpitas, CA 95036, USA

of-the-art non-volatile memory express interface SSD within the Hadoop MapReduce Framework. While HDFS read and write throughput increases with high-performance SSDs, achieving complete system performance improvement requires carefully balancing CPU, network, and storage resource capabilities at a system level.

Keywords Performance · Storage · SSD · Hadoop · MapReduce · HDFS

1 Introduction

The Hadoop MapReduce Framework [1,2] enables data scientists to analyze terabytes of data in parallel. Hadoop [3] is an Apache open source project that implements the Hadoop Distributed File System (HDFS) [4] and MapReduce. While many strategies can improve MapReduce performance, introducing SSDs is a particularly attractive strategy because SSDs exhibit superior performance—lower latency, higher IO rates, and higher throughput compared to HDDs [5]. The Hadoop MapReduce Framework has historically used HDDs as its only storage device and its design directly reflects that. Any benefits that SSDs presently offer are, therefore, largely incidental.

For example, because of associated HDD latencies, HDFS typically performs sequential I/Os versus random I/Os, a design consideration that can significantly improve HDD performance but is largely irrelevant for SSDs [6]. Since SSDs were never considered, MapReduce's original and present design marginalizes potential SSD performance benefits by default.

This paper is an extension of our previous work [7] which explored the benefits SSDs potentially offer to MapReduce. In addition to the previous work, we investigate different options such as frequent heartbeat interval and JVM reuse for optimizing Hadoop with SSDs. Moreover, we newly look into efficiently using faster SSDs (non-volatile memory express [NVMe] interface-based SSDs, NVMe SSDs in this paper) within Hadoop. To the best of our knowledge, this paper is the first work to investigate high-performance NVMe SSD performance with the Hadoop MapReduce Framework.

The contributions of this paper are:

- We investigate HDFS performance in different configurations using SSDs and HDDs. This reveals the present maximum SSD advantage is fulfilled when SSDs serve concurrent, multi-thread read/write requests. We develop a general network bandwidth guideline to balance systems: network (half duplex) bandwidth should be 1.33 times the storage throughput when using a replication factor of 3.
- We examine Hadoop workflows and identify various performance bottlenecks within different configurations. We observe that superior SSD random read and write performance accelerates MapReduce processing significantly when SSDs store intermediate data. Terasort benchmark results suggest using SSDs can accelerate the MapReduce shuffle phase by 1.75 on average. We study optional configurations such as frequent heartbeat and JVM reuse to optimize the efficiency of the proposed configuration in the map phase.
- We explore cost-effective and energy-saving Hadoop configurations. Our investigations show that properly using DRAM or SSDs for intermediate data can

significantly accelerate MapReduce performance. The most cost-effective and energy-efficient SSD application is storing intermediate data.

– We see the feasibility of next-generation storage devices (NVMe SSDs) in Hadoop. Our experimental results imply that other resources, such as CPU capability, should match the storage device's performance.

This paper has the following organization: Section 2 examines basic Hadoop File I/O (HDFS) performance. Section 3 discusses Hadoop MapReduce behavior and proposes a cost-effective, energy-efficient, high-performance storage configuration that judiciously incorporates SSDs. Section 4 explores optional configurations to maximize the benefit of the proposed configuration. Section 5 sees the potential benefit of next-generation storage devices in Hadoop. Section 6 introduces related work, and Sect. 7 is the conclusion.

2 HDFS characteristics

We conducted several experiments within different scenarios comparing SSD and HDD performance. This section discusses many investigative findings about Hadoop I/O performance.

2.1 Investigation environment

The goal of our first investigation was to examine the behavior of the Hadoop Datanode and HDFS. We configured a small Hadoop cluster described in Table 1.

Table 1 Investigation environment

Machines	Machine #1 runs a Namenode and a JobTracker
	Machines #2–#5 run Datanodes and TaskTrackers
Processor	Two Intel Xeon E5-2630 2.3 GHz CPUs per machine
	- 6 cores (12 threads with hyperthreading) per CPU
Memory	16 GB DRAM (DDR3 1600 with ECC)
Network	All machines connected through
	(1) two 10 Gbit Ethernet NIC (full duplex)
	- 20 Gbit total, using link aggregation
	(2) 1 Gbit Ethernet (half duplex)
Storage	2 HDDs RAID0, (2TB, SATA)
	2 Samsung SM1625 SSDs (400 GB, MLC, SAS)
	1 Samsung XS1715 SSD (1.6TB, NVM Express)
Software	Native Hadoop 0.20.205.0
	(replication factor = 3, block size = 128 MB)
	ubuntu 12.04 desktop
	openJDK 1.6.0_27, 64-bit

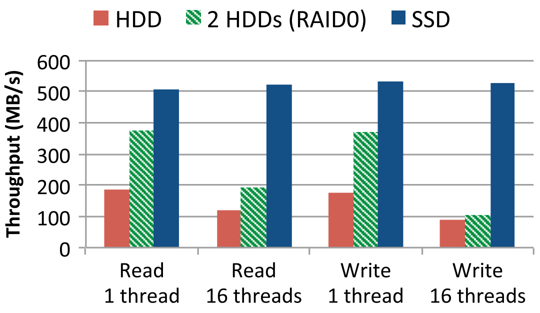


Fig. 1 Storage I/O throughput under fio with queue depth: 32, block size: 128 KB, sync I/O, direct I/O

We exploit the Linux sysstat (sar) and mpstat tools to measure, gather, and show various Datanode resource utilizations. These tools observe CPU, memory, storage I/O and network utilizations.

Raw storage device performance HDFS is a user-level distributed file system superimposed on a native file system. HDFS performance, therefore, highly depends both on the native file system and storage media performance. We first reviewed the native file system and storage media performance. In our investigations, the OS was Linux and the native file system was ext4, which is a widely used configuration. The impact of different filesystems on Hadoop is investigated in the related work [6] in detail.

Datanode storage media was an HDD, 2 HDDs (RAID0), and one Serial Attached (SAS) SSD. The NVMe interface is designed for enterprise level SSDs and uses the PCI Express (PCIe) interface. Section 5 discusses NVMe SSD details.

Flexible IO Tester (FIO) [8] is an open source I/O benchmark supporting a plenty of configurations. We used the fio benchmark with the following options: (1) queue depth = 32, (2) 128 KB block size, (3) sync I/O, and (4) direct I/O. System engineers typically select a sufficiently deep queue depth to prevent significant block device driver I/O performance delays. We found that configurations with queue depths greater than 32 or with block sizes greater than 128 KB marginally changed the result.

Figure 1 shows measurement results. This measurement provides a baseline for other experiments in this paper. In the figure, HDD throughput decreases as the concurrent thread count increases. In contrast, SSD throughput slightly increases.

SSD random write performance significantly decreases with a nearly full capacity utilization. In this paper, in all experiments, SSDs have enough space such that performance degradation is barely observed. Enterprise class SSDs such as NVMe SSDs are an order of magnitude faster than client class SSDs. Section 5 discusses the performance of NVMe SSDs in detail.

Test Distributed File System I/O (TestDFSIO, hereafter DFSIO) [9] is a widely used HDFS performance benchmark. It distributes Map tasks that read/write dummy files on Datanodes. Each Map task reads the local dummy file and writes a few statistical output lines. Reduce tasks simply gather the statistics for output.

While conducting DFSIO benchmarks on the Hadoop cluster, we encountered various problems. First, in real workloads, some HDFS read operations experience occasional Linux kernel page cache hits which affect (improve) performance. However, when running consecutive, identical DFSIO benchmarks, DFSIO instances experienced exaggerated page cache hit rates, producing results that are not representative in typical usage. Therefore, to avoid amplified page cache hits, we halted Hadoop after each DFSIO run, flushed the page cache, and restarted Hadoop.

Second, we found that DFSIO control file occasionally identifies incorrect locations where actual replicas are stored. This possibly leads Datanodes to read data from remote storage even though it has the data in local storage. We, therefore, necessarily modified the original DFSIO read-function source code to prevent unwanted remote accesses.

2.2 Storage vs. network

We performed HDFS performance measurements using 1 Gbit Ethernet links, 20 Gbit links constructed by combining two 10 Gbit Ethernet links via link aggregation (channel bonding), HDDs, and SSDs. This enabled us to vary storage devices and network bandwidth to determine how various SSD and high network bandwidth combinations could improve HDFS throughput.

Figure 2 depicts DFSIO data processing throughput for various cluster configurations. The figure shows DFSIO throughput when it reads/writes a file with different file size in each node. We performed DFSIO benchmark five times[1] for each configuration. The associated plots indicate the average throughput for these configurations. Figure 2a, b shows the HDFS read performance using 1 and 20 Gbit Ethernet links, respectively. As expected, HDFS read performance largely depends on the backing storage device's sequential read performance. In the figures, SSD performance is higher than the two-HDD RAID0 configuration's performance and much higher than a single HDD's performance. It is noted that read performance is independent of network bandwidth since map tasks typically read data from local storage. In contrast, write performance directly depends on network bandwidth because DFSIO write includes file replications through network. While one Datanode's replication traffic traverses the network, replicas from other Datanodes simultaneously contend for network access, creating significant network bandwidth contention. Figure 2c, d compares HDD and SSD write performance using 1 and 20 Gbit Ethernet links. In Fig. 2c, because an 1 Gbit Ethernet network provides significantly lower bandwidth than any storage device's transfer rate in any configuration, HDFS throughput is identical in all cases, regardless of the storage device. However, using 20 Gbit Ethernet reveals a noticeable HDFS throughput improvement when using SSDs versus HDDs. An interesting observation in Fig. 2d is that HDD and HDD RAID0 throughput decreases as file size increases. The reason is that a considerable amount of write (here, up to 4–5 GB for original data and 8–10 GB for replica in 16 GB DRAM) is kept in Linux kernel page cache and is delayed until the page cache is full. A huge performance gap between SSD and HDD results in relatively less degradation in SSD throughput.

To see the network and I/O resource consumption during DFSIO operations, we profiled the resource utilization for all configuration variations. Figure 3 shows the resource utilization with the DFSIO write benchmark. This figure shows the average resource utilization of four Datanodes. In Fig. 3a, b, network throughput does not

[1] We repeated five times since variance of results is small enough.

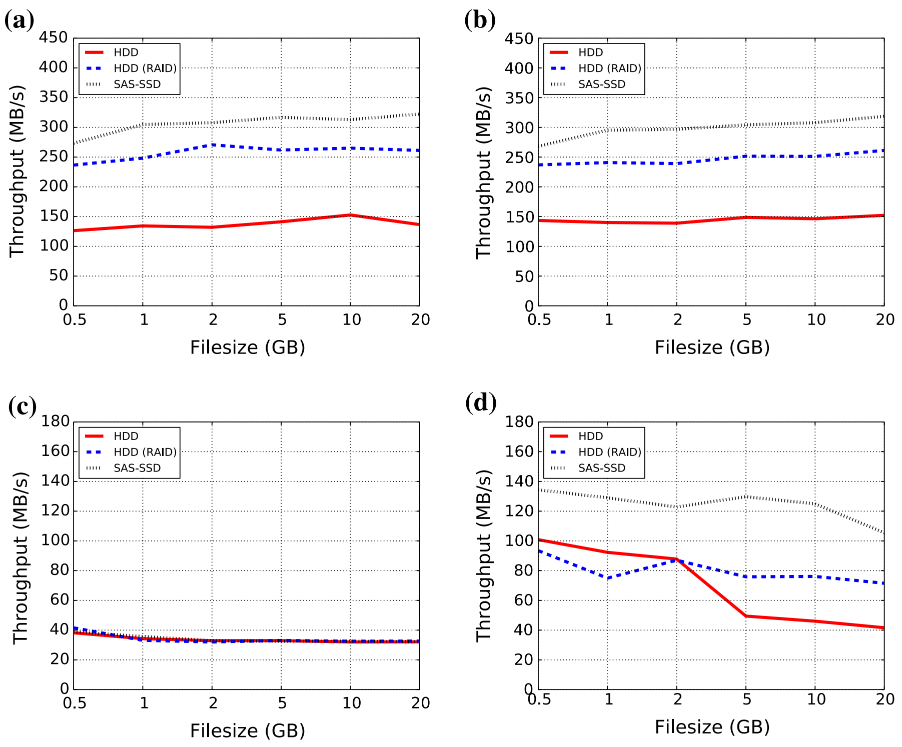


Fig. 2 DFSIO throughput vs. file size (I/O, network, and storage). **a** READ, 1GE; **b** READ, 20GE; **c** WRITE, 1GE; **d** WRITE, 20GE

exceed (is saturated at) 120–180 MB/s (1 Gbps approximately), while I/O utilization[2] still has room to grow.

When comparing network and I/O resource utilizations in 20 Gbit Ethernet configurations depicted in Fig. 3c, d, a very different scenario emerges. Both SSD and HDD I/O utilizations are saturated while network bandwidth is not. Employing more HDDs or SSDs can exploit the potential of high-performance networks. Consequently, matching network bandwidth and I/O throughput is essential when attempting to build cost-efficient systems.

2.2.1 Network bandwidth requirement for high-performance storage devices

These observations and considerations enable us to estimate the required network bandwidth per link to maximize I/O throughput of storage devices per Datanode.

1. HDFS replication amplifies I/O throughput by N times where (N) is the HDFS replication factor:

$$K \cdot T = N \cdot D$$

[2] I/O utilization is defined as the percentage of CPU time passed during I/O requests were issued [10].

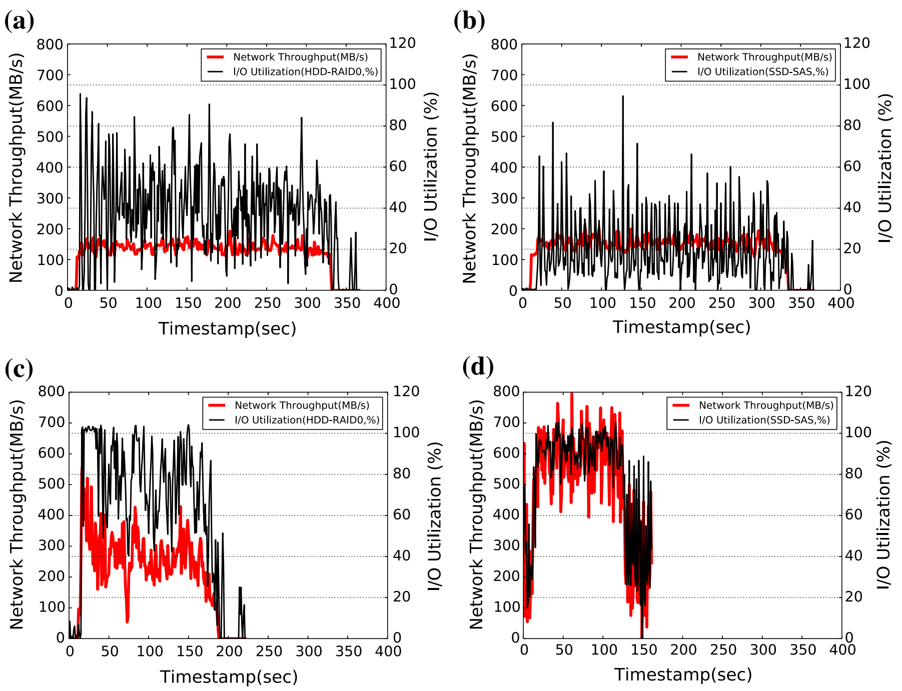


Fig. 3 DFSIO write (network throughput and I/O utilization). **a** 1GE, HDD; **b** 1GE, SSD; **c** 20GE, HDD; **d** 20GE, SSD

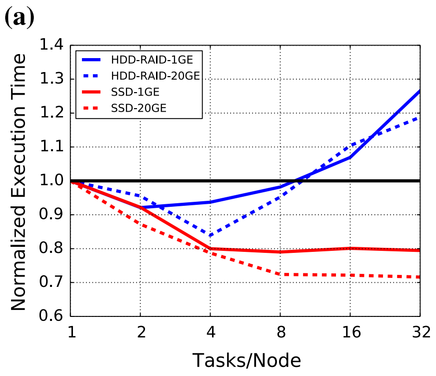
In the equation, (K) is the number of Datanodes in a cluster, (T) is I/O throughput of storage devices per Datanode, and (D) is the HDFS I/O throughput per cluster.

2. Network bandwidth capability $(C \cdot K \cdot B$, where (C) is a coefficient for (half/full) duplex and (B) is network throughput per link) should sufficiently cover the I/O traffic written to other Datanodes $(D \cdot (N - 1))$:

$$C \cdot K \cdot B \geq D \cdot (N - 1) \tag{1}$$

The $(N - 1)$ term considers I/O traffic which is not written to local storage since local writes are not transmitted. The coefficient C in Eq. (1) is (0.5) for the half duplex and (1.0) for the full duplex network.

For example, 100 MB/s write in an HDFS cluster (D) results in 300 MB/s write $(K \cdot T)$ to storage devices in the cluster with 4 Datanodes (K) when the replication factor (N) is 3. Each storage device in a Datanode serves 75 MB/s, and of the 75 MB/s write activity, 25 MB/s is original data produced by the Datanode, and 50 MB/s is replicas of other Datanode data. Each half duplex network link should support at least 100 MB/s bandwidth (B) to maximize the performance of HDFS to cover both the incoming (50 MB/s) and the outgoing (25 MB/s $\times$ 2 replications) traffic.



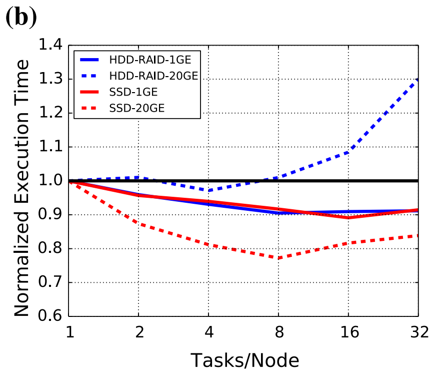


Fig. 4 Normalized DFSIO execution times. The standard deviations are **a** 6.7 % , and **b** 3.21 % on average. **a** READ, **b** WRITE

In other words, the network bandwidth required per link ($B_{\min}$) is:

$$B_{\min} = \frac{1}{C} \cdot T \cdot \frac{N-1}{N} \tag{2}$$

This implies that the network bandwidth more than $B_{\min}$ should be guaranteed to fully utilize storage device throughput. Otherwise, it would bound the storage throughput to less than those shown in Fig. 1.

2.3 Multiple readers/writers

The previous investigation considered HDFS performance when each Datanode runs a single map task. However, in real Hadoop clusters, Datanodes concurrently run multiple, often numerous, map tasks. Observing the effect of multiple concurrent HDFS readers and writers requires measuring DFSIO execution time with multiple tasks per Datanode. Figure 4 shows the DFSIO execution time normalized to the execution time of a single task per Datanode. Figure 4a shows the DFSIO read execution time of HDDs increases significantly as the number of tasks increases, while that of SSDs decreases. As shown in Fig. 4b, since 1 Gbit Ethernet limits the number of concurrent access to storage, I/O randomness is bounded. Consequently, HDDs and SSDs exhibit similar write performance. However, 20 Gbit Ethernet provides enough bandwidth for a number of concurrent I/Os. This implies that high-speed networks can exploit potential performance of high-performance storage.

Figure 5 demonstrates I/O throughput and I/O utilization with varying numbers of DFSIO read tasks. Note that I/O utilization does not exactly match a certain throughput value because it depends on the utilization of the block device I/O queue. We found that HDD I/O utilization is approximately 100 % when the task count is 1 or 32. In contrast, the total throughput is lower when the task count is 32 than when it is 1. This is due to poor HDD random read performance. When the number of tasks accessing an individual HDD increases, I/O throughput significantly decreases due to

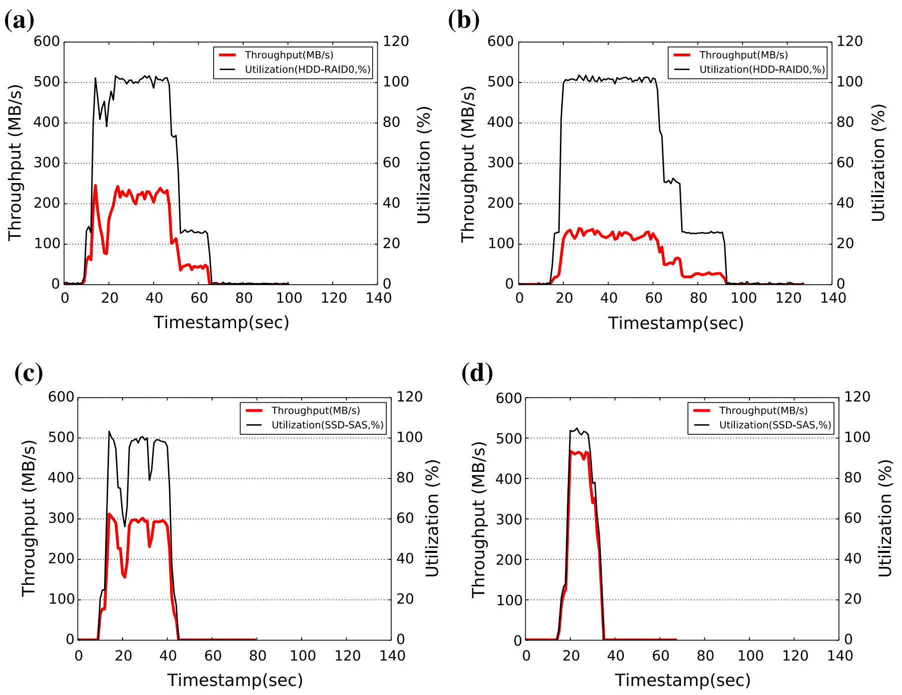


Fig. 5 Single/multiple task I/O throughput (MB/s) and utilization (%) (READ). **a** 2 HDD (RAID0), 1 task/node; **b** 2 HDD (RAID0), 32 tasks/node; **c** SSD (SAS), 1 task/node; **d** SSD (SAS), 32 tasks/node

HDD seek and rotational latencies. However, SSDs perform random reads much faster than HDDs. Moreover, single tasks typically cannot consume all SSD bandwidth. This typically requires multiple tasks. As shown in Fig. 5c, d, with SSDs, I/O throughput increases as the number of tasks increases.

An interesting observation is that for both HDDs and SSDs, single-task I/O throughputs are less than the maximum value indicated in Fig. 1 when sysstat (sar) reports device I/O utilization is approximately 100 %. This is because DFSIO uses synchronous reads. Hence, single tasks can only have one outstanding request to an I/O queue. This means single tasks cannot hide the IO bus transaction latency but do not receive disk scheduling benefits such as SATA Native Command Queueing (NCQ).

HDFS performance can further improve using techniques such as pipelining, where a mapper processes data while another thread loads the next block. Unlike when using HDDs, increasing the concurrent thread count improves HDFS performance when using SSDs because of their faster random read performance.

In this section, we have explored the HDFS performance as a baseline of the experimental results in this paper. We have shown that an SSD has significantly better performance than an HDD especially with multiple streams of random workload. We have also revealed that network bandwidth should satisfy a requirement to utilize SSD's full throughput.

Table 2 Terasort investigation configuration

Property	Configuration
Storage	Two HDDs (RAID-0)
	2 Samsung SM1625 SSDs (SAS)
DRAM	16 or 48 GB (DDR3 1600 ECC)
Data size	100 GB
Compression	Java native compressor on map output
HDFS block size	128 MB
HDFS I/O buffer size	128 KB
Speculative exc.	No speculative execution
JVM heap size	512 MB for map, 1,000 MB for reduce
Slow start	Reduce starts at 95 % completion of map
Max. # of maps	12 per datanode
# of reduces	6 per datanode

3 MapReduce from a storage perspective

This section discusses how differing storage device performance affects MapReduce performance. Here, we run the Terasort benchmark [11] to evaluate Hadoop performance.

Map output size varies depending on user applications. With Terasort, map output always has the same size as the input data. *Intermediate data* are a portion of map output temporally stored in local storage. The place to write intermediate data is configurable. We call the local storage for intermediate data *intermediate storage*. Compressing map output is often used to reduce the amount of data stored in intermediate storage and transferred through the cluster network [12].

3.1 Terasort performance analysis

This section explores the effects HDDs and SSDs have on Terasort performance. In addition, it discusses a case where SSDs are intermediate data storage and HDDs provide HDFS storage, and a case where a large DRAM resource caches the majority of intermediate data. Table 2 describes configuration details.

Terasort consists of common and essential MapReduce procedures with the lightest mappers and reducers. Figure 6 briefly describes how Terasort works. Terasort reads unsorted key value pairs data in files, sorts the key value pairs, and writes the sorted data to HDFS. Each Terasort mapper reads a block to get a number of key value pairs. The pairs are sorted and stored in temporary files in the configured intermediate storage location. Reducers read the stored map output from intermediate files. Each reducer has its own key sub-space and only reads the portion of map output it needs using an http connection. When a reducer completes copying map output, it merges the copied chunks as a sub-step of merge sort. The merged data are stored in HDFS.

As depicted in Fig. 6 above, Terasort consists of three phases: (1) map phase, (2) shuffle phase, and (3) reduce phase.

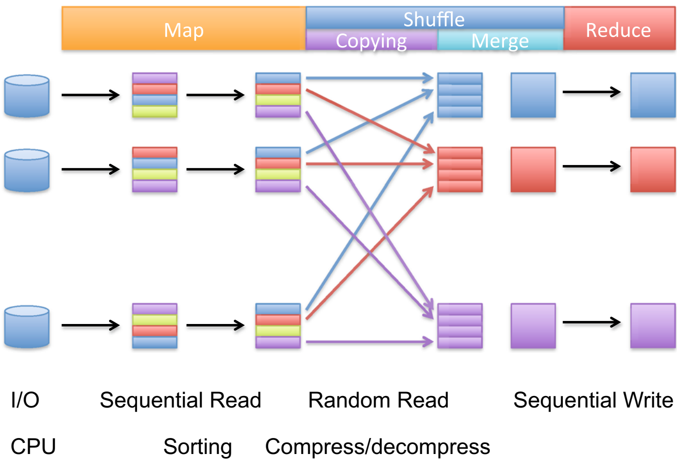


Fig. 6 Terasort workflow overview

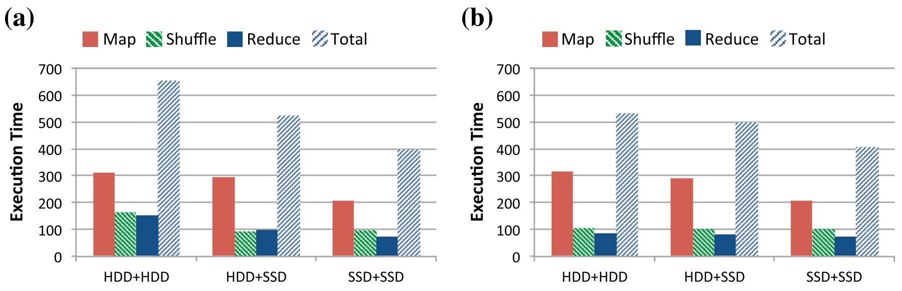


Fig. 7 Terasort investigation execution times with varying DRAM amounts. Performance gain from SSD comes from execution time reduction in shuffle phase in HDD + SSD, and in all phase in SSD + SSD configuration. **a** DRAM: 16 GB/node **b** DRAM: 48 GB/node

Figure 7 shows Terasort execution times for various cluster configurations. Key Terasort performance observations are: (1) map phase performance depends on HDFS read throughput (primarily sequential) performance. (2) Shuffle phase performance mainly depends on the random I/O performance for reading map output and writing intermediate data. To explicitly identify performance degradation sources, we profiled CPU and I/O utilizations during investigations, depicted in Figs. 8 and 9. (3) Reduce phase performance depends more on random read performance of intermediate data storage than the HDFS storage media performance.

In our investigations, each phase slightly overlaps since we specified reducers start when the map phase is 95 % complete. The number of map tasks is typically more than the number of map tasks that Datanodes can handle in one parallel round. Namenode assigns as many map tasks as possible to a Datanode which can run in parallel, waits for map task completions, and then assigns a number of map tasks, repeatedly. This makes the map phase consist of a succession of task groups called *waves*.

In the map phase, the number of concurrent cluster map tasks is set to 48. Hence, we need 17 [(800/48) + 1] map task execution waves (each with 48 map tasks) to

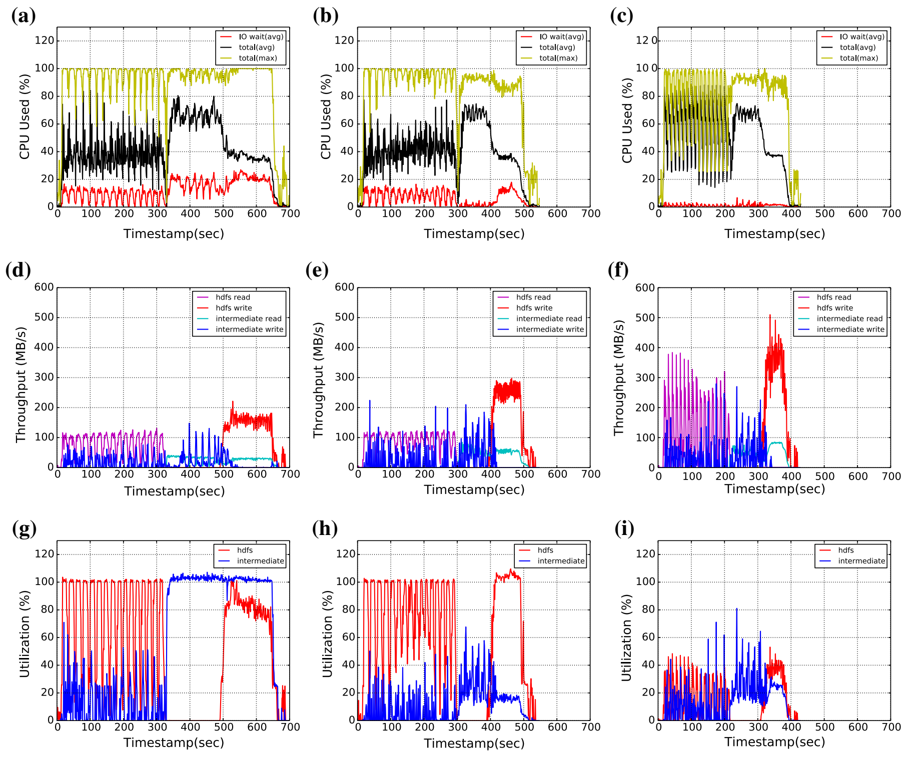


Fig. 8 MapReduce CPU utilization and I/O throughput, DRAM: 16 GB. HDD performance determines overall performance of HDD + HDD (**a**, **d**, **g**) configuration in all phase, while it does of HDD + SSD (**b**, **e**, **h**) configuration only in map (0–300 s) and reduce (400–500 s). Neither CPU or storage is saturated in SSD + SSD configuration. **a** CPU, HDD + HDD; **b** CPU, HDD + SSD; **c** CPU, SSD + SSD; **d** I/O (MB/s), HDD + HDD; **e** I/O (MB/s), HDD + SSD; **f** I/O (MB/s), SSD + SSD; **g** I/O (%), HDD + HDD; **h** I/O (%), HDD + SSD; **i** I/O (%), SSD + SSD

initiate the required 800 map tasks [100 GB/(128 MB block)] since each map task normally processes approximately 128 MB of data. Thus, due to coarse-grained task scheduling (3 s in our investigations), consecutive peaks occur over the map phases in Figs. 8 and 9.

In Fig. 8, when HDFS uses HDDs, I/O becomes the map phase bottleneck. In contrast, when HDFS uses SSDs, CPUs are heavily utilized to handle user processes. During the shuffle phase, we see about 50 MB/s read throughput and 100 % of I/O utilization when HDDs store intermediate data. I/O utilization is much lower (20–40 %) when SSDs store intermediate data. During the shuffle phase, HDD intermediate data read throughput is much less than the maximum HDD capability, while I/O utilization is 100 % because of higher shuffle phase random read seek times. When using SSDs to store intermediate data, the shuffle phase CPU I/O wait time significantly decreases (to nearly zero) compared to using HDD storage (up to 20 %). This reduces overall execution time, and provides more opportunities for the system to increase concurrent job and task counts. These resource utilization findings indicate that shuffle phase execution time highly depends on storage device’s random read performance.

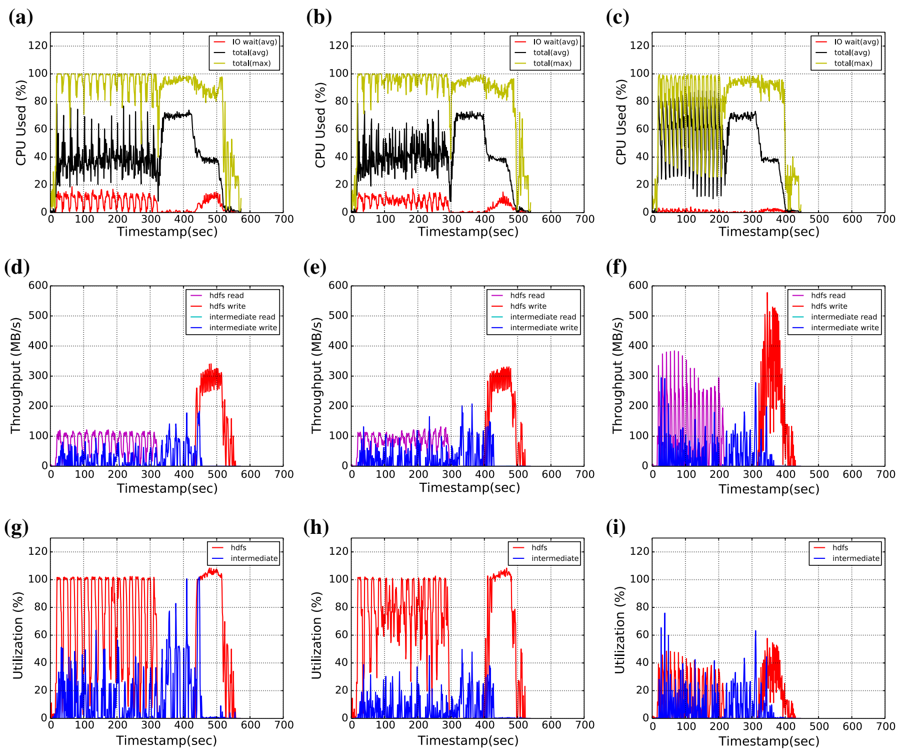


Fig. 9 MapReduce CPU utilization and I/O throughput, DRAM 48 GB. Large-size DRAM has a similar role to SSD intermediate storage. HDD utilization does not saturated in shuffle phase (300–400 s) in HDD + HDD (**a**, **d**, **g**) and HDD + SSD (**b**, **e**, **h**) configurations but in map (0–300 s) and reduce (400–500 s). Neither CPU or storage is saturated in SSD + SSD configuration. **a** CPU, HDD + HDD; **b** CPU, HDD + SSD; **c** CPU, SSD + SSD; **d** I/O (MB/s), HDD + HDD; **e** I/O (MB/s), HDD + SSD; **f** I/O (MB/s), SSD + SSD; **g** I/O (%), HDD + HDD; **h** I/O (%), HDD + SSD; **i** I/O (%), SSD + SSD

Finally, in the reduce phase, I/O utilization using HDDs for intermediate data storage still shows 100 % utilization. Hence, this can be bottleneck even for HDD HDFS because HDD random read throughput is significantly less than HDD sequential write throughput. The use of SSDs as the intermediate data storage provides HDFS with adequate throughput, which makes HDFS utilization close to 100 %. When SSDs provide all storage, no I/O or CPU resource bottleneck is apparent.

Figure 9 shows resource utilization when Datanode DRAM size expands from 16 to 48 GB. Since most intermediate data are cached in the Linux kernel page cache, total execution time highly depends on map phase storage media native read performance, and reduce phase write performance. Shuffle phase performance is essentially the same for all cases. Note that in all three cases, the intermediate storage read throughput is zero in the shuffle phase since most intermediate data are cached in DRAM, allowing it to be read from memory. Therefore, expanded DRAM increases total performance by reducing shuffle phase execution time, but it incurs significant DRAM capital costs and operation costs due to energy consumption. The next section analyzes the performance and cost relationship to determine the most cost-efficient deployment strategy.

Table 3 Commercial price per unit of major Datanode components

	Description	HH16	HS16	SS16	HH48	HS48	SS48	Price($)
CPU	Intel Xeon E5-2630 2.3 GHz	2	2	2	2	2	2	540
Board	Intel W2600CR	1	1	1	1	1	1	600
Power	SC5650DPNA	1	1	1	1	1	1	270
NIC	Intel X520-DA2	1	1	1	1	1	1	430
RAM	DDR3 4 × 4 GB	1	1	1	3	3	3	150
HDD	Seagate 2TB 7200 rpm	3	2	0	3	2	0	83
SSD-1	Samsung 840 Pro 256 GB	0	1	1	0	1	1	150
SSD-2	Samsung 840 Pro 512 GB	0	0	8	0	0	8	289

3.2 Cost analysis

The previous section's Terasort evaluation shows that using DRAM or an SSD as intermediate data storage significantly reduces shuffle phase execution time. When deploying SSDs as HDFS storage, Hadoop map and reduce phase performance considerably improves. However, because of SSD cost-per-bit, it is presently not economically feasible to replace all HDDs, though the SSD cost-per-bit is still rapidly decreasing. In this section, we considered both Datanode configuration cost and Datanode performance to estimate Datanode cost effectiveness.

First, we determined the cost to build the various machine configurations our investigations used. Table 3 shows components counts and prices. We note that a 10GE network is employed but the network interface bandwidth larger than 1GE barely impacts on the throughput of Terasort in all configurations in the table.

From the component prices, we can calculate unit price of Datanodes in Fig. 10. In the figure, A + B (C) represents the storage A is used for HDFS, the storage B is used for intermediate data, and the amount of DRAM is C GB. All values in the figure are normalized to the HDD + HDD (16) result values. For fair comparisons, we assumed the SSD + SSD (16) and SSD + SSD (48) configuration had as many SSDs as required to provide the same amount of HDFS storage capacity as with the HDDs (4TB). Providing intermediate data storage requires one 256 GB SSD. We find that the SSD performance is stable regardless of its capacity because its capacity is provided enough to accommodate intermediate data. The effectiveness (higher values are better) is computed by dividing throughput by price.

The results show that using SSDs as intermediate data storage is the best option in terms of calculated effectiveness. The effectiveness is 15 % better than HDD + HDD (16). However, while expanded DRAM is also beneficial, simultaneously expanding DRAM and introducing SSDs as temporary storage does not help. Finally, replacing all HDFS HDDs with SSDs is only appropriate when it is imperative to increase

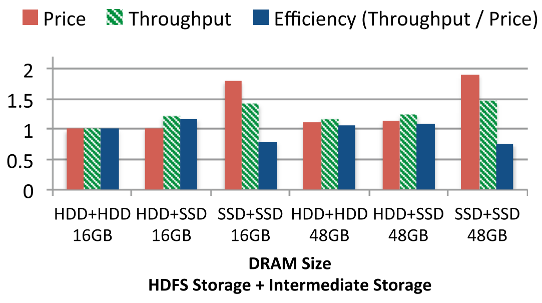


Fig. 10 Cost of a Datanode running Terasort normalized to the HDD + HDD/16 GB value

Table 4 Power consumption statistics of a Datanode running Terasort

	Voltage (V)	Current (A)	Power (W)	P.Factor	Run time (s)	Energy (Wh)
HDD + HDD	119.20	1.75	204.82	−0.98	1885	107.25
HDD + SSD	119.30	1.77	208.09	−0.98	1727	99.83
SSD + SSD	118.62	1.84	215.37	−0.98	1518	90.81

performance despite obvious significant cost differentials. Its effectiveness is 34 % worse than HDD + HDD (16)'s effectiveness.

3.3 Energy consumption analysis

The previous section's cost analysis is a one-time expense until device replacement occurs. In contrast, energy consumption is a primary recurring expense that can be more important in large-scale data center. In this section, the energy consumption of different Hadoop configurations is discussed.

We installed a power-meter device directly to a Datanode running Terasort. Table 4 shows observed current, voltage, power, and energy consumption of the Datanode.

It is counter intuitive that SSD + SSD configuration requires more power than HDD + HDD and HDD + SSD. We found that in the HDD + HDD and HDD + SSD configurations, CPU and memory are less utilized because HDDs are a system bottleneck. Hence, configurations with HDDs demand less average power than configurations with SSDs. However, resulting Terasort run time energy consumption totals are larger with HDD configurations than with SSD configurations. The total energy consumption of the HDD + HDD configuration is 18 % higher than the SSD + SSD configuration and 7 % higher than the HDD + SSD configuration.

This suggests many interesting topics to consider in large-scale MapReduce cluster designs. First, energy consumption should be the amount of energy used to complete a job instead of instantaneous power demands. We show the gap between HDDs and SSDs in instant power is marginal while the gap in total energy consumption is considerably larger. Second, most SSD energy consumption benefit comes from reduced run times.

In this section, we have shown that SSD + HDD is the best option in terms of performance, cost, and energy. However, SSD + HDD configuration only optimizes

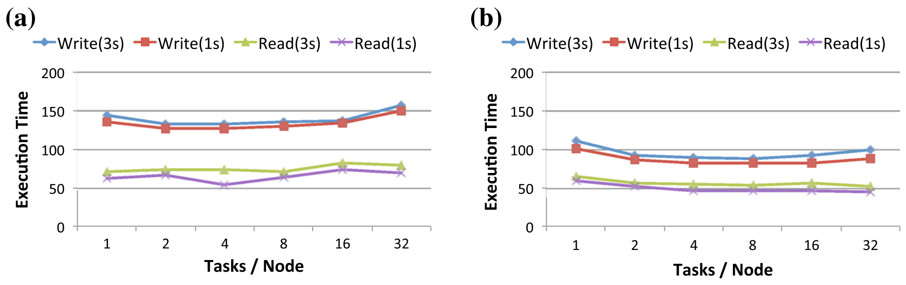
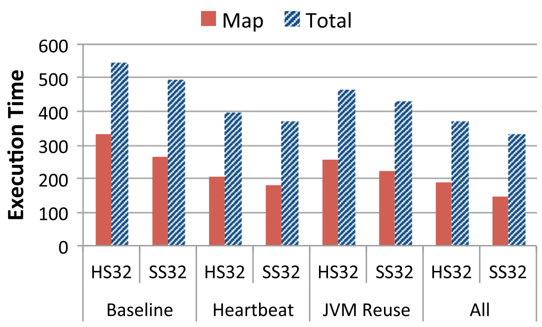


Fig. 11 DFSIO execution time vs. the number of concurrent tasks per node with different heartbeat intervals. **a** HDD, DFSIO 10 GB, 10GE; **b** SSD, DFSIO 10 GB, 10GE

Fig. 12 Terasort run time (s) with/without JVM reuse option and a 1-s heartbeat interval



shuffle phase, while map and reduce phase have a considerable portion in a MapReduce framework. In the next section, we will explore different optimization techniques for map/reduce phases.

4 Optimize map performance

According to our experiment results, introducing SSDs for intermediate data improves Hadoop performance significantly by eliminating shuffle phase delay. The proposed HDD + SSD configuration is optimal when considering performance, cost, and power consumption together. However, the HDD + SSD configuration has less performance than the SSD + SSD configuration because map phase performance highly relies on storage media sequential read performance (HDDs in the HDD + SSD configuration and SSDs in the SSD + SSD configuration). We conducted several additional experiments to reduce the performance gap between HDDs and SSDs in the map phase. Figures 11 and 12 show DFSIO and Terasort execution time with different Hadoop options to boost map phase performance. This section discusses the details.

4.1 Heartbeat interval

There are two types of heartbeat in Hadoop. First, Hadoop Datanodes send a heartbeat signal to the Namenode to indicate Datanode health status. This type of heartbeat interval is configurable.

In addition, Hadoop's Task Tracker also sends a heartbeat signal to the Job Tracker every 3 s. Heartbeats carry task completion messages and result in scheduling delays of up to 3 s between two consecutive tasks (per wave). Unlike the previous one, this type of heartbeat interval is not configurable. We have modified Hadoop source code to reduce the Task Tracker heartbeat interval.

The frequency of the two heartbeat types may impact Hadoop map phase performance. We conducted DFSIO benchmarks with different heartbeat intervals in Fig. 11. The result shows that frequent heartbeat marginally enhances HDFS performance since DFSIO benchmark schedules tasks only at initial stage of benchmark running.

Figure 12 shows Terasort benchmark execution time with different heartbeat intervals. In contrast to the result in Fig. 11, frequent heartbeats reduce Terasort running time considerably. In the map phase, 17 waves (48 map tasks per wave) run in a Datanode. We expected map phase execution times to decrease by up to 54 s on average assuming slow heartbeat delays would be 0–3 s on average between two waves. However, the time we earned in map phase is surprisingly high (up to 74 s in the HDD + SSD configuration without the JVM reuse option, exceeding our expectation). This implies, with HDDs, there is an unrecognized heartbeat interval side effect in map phase. The shorter heartbeat interval reduces map phase run times by 38.0 and 32.7 % for HDD + SSD and SSD + SSD, respectively.

Despite our experimental result showing that frequent heartbeat improves Hadoop performance, a frequent heartbeat is not always beneficial. In large-scale cluster, frequent heartbeat signals may overwhelm the Namenode's processing ability and degrade overall performance. More investigation is required to understand the relationship between heartbeat intervals and Hadoop performance and to find an optimal heartbeat interval.

4.2 Java virtual machine reuse

Hadoop initiates Java Virtual Machines (JVMs) for each task and, by default, destroys JVMs at task completion. Unlike the shuffle and reduce phases, which typically have a single wave of tasks, in map phases, many waves force repetitive JVM construction and destruction. This leads to severe map phase performance degradation.

The JVM reuse option enables Datanodes to reduce map phase delay by reusing JVMs. Figure 12 compares execution time of Terasort with/without the JVM reuse option enabled.

The JVM reuse option is useful when a large number of map task waves are required. This is an unforeseen delay in original Hadoop design that assumed large-scale clusters requiring a few waves. However, small-size or medium-size Hadoop clusters experience a large number of waves. The result shows that 22.3 and 16.5 % of map phase execution time in HDD + SSD and SSD + SSD configurations, respectively, is wasted for unnecessary JVM construction and destruction.

It is an interesting observation that using both frequent heartbeat intervals and enabling the JVM reuse option makes Terasort faster than using a single approach. For example, total Terasort execution time in the HDD + SSD configuration decreased by 27.4 and 15.2 % just using 1-s heartbeat or just the JVM reuse option, respectively, while it does by 32.2 % using both options.

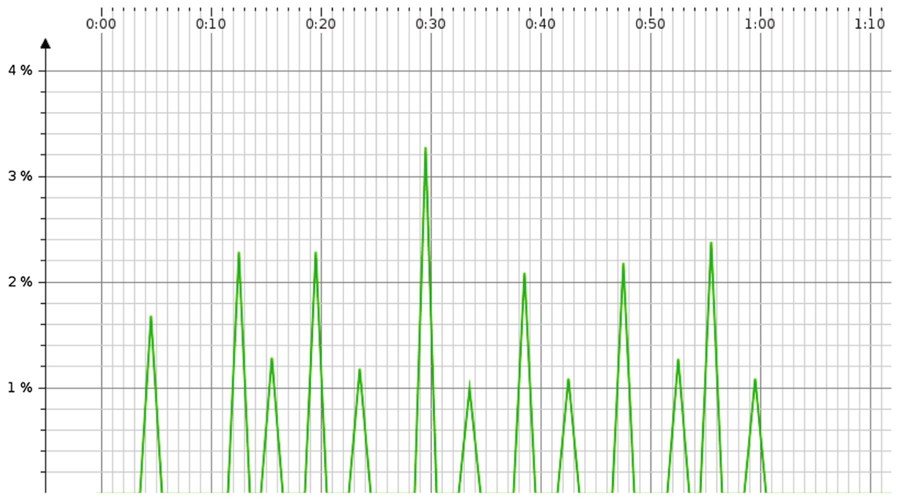


Fig. 13 Garbage collection overhead in Terasort (SSD + SSD configuration) using JVM reuse option. In this figure, the x-axis shows the elapsed time (min) and the y-axis shows how much time (%) was spent for garbage collection

The JVM reuse option is not always beneficial for map phase performance. When reusing a JVM, data the JVM previously used are garbage and should be cleaned out. The cleaning is done by JVM garbage collection which may be CPU and memory usage expensive.

Consequently, recycling a JVM many times requires heavy garbage collection processing, potentially delaying execution time. We use JProfiler [13] to track how many garbage collection processes JVM reuse requires. Figure 13 shows the garbage collection overhead during Terasort map phase when JVM reuse is enabled.

We see a number of garbage collections occurred. However, in our experiments, garbage collection performance degradation is less than that from JVM reuse.

4.3 Native file system access time option

We tried to optimize native file system (ext4) performance using the *noatime* option. The noatime option is recognized to be effective in improving traditional Hadoop's performance (with HDDs only) [14–17]. We expect this is also effective in SSD configurations. However, our experimental results show no significant improvement from the change. We found that the noatime option is only effective when small size files are massively overwritten in the native file system because benefits come from reduced metadata modification. Since the HDD + SSD and SSD + SSD configuration use SSDs in the shuffle phase, small and random access is not a Hadoop framework bottleneck. This minimizes the noatime optimization benefit in HDD + SSD and SSD + SSD configurations. This implies that other options to relieve random access speed in Hadoop would not favor HDD + SSD and SSD + SSD configurations because SSDs sufficiently addresses required random access.

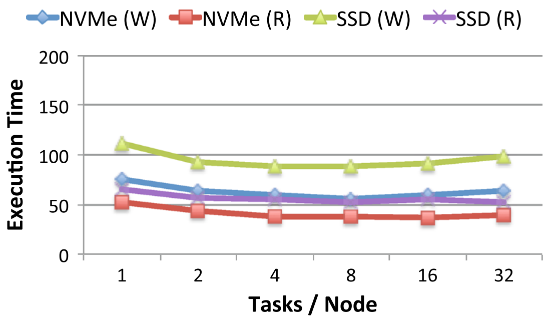


Fig. 14 DFSIO execution time of SSD (SAS) and NVMe SSD

This section has explored different options such as frequent heartbeat, JVM reuse, and noatime in ext4 filesystem to reduce execution time in map and/or reduce phase. Frequent heartbeat and JVM reuse options are effective and more when both of them are used, while noatime option has a marginal impact on the performance of MapReduce framework.

5 NVM express SSD

We investigated the effectiveness of using high-end SSDs connected to a PCIe bus that exposed an NVM express (NVMe) interface. The NVMe [18] interface is a variation of PCI express interface supporting high-performance PCI express-based SSDs. Specifically, we used Samsung XS1715 SSDs (1.6TB) [19] in our experiment. In this section, a single 10 Gbit Ethernet is used instead of 20 Gbit Ethernet composed of two bonded 10 Gbit Ethernet interfaces.

The NVMe SSD has 3.0 GB/s sequential read bandwidth and 1.5 GB/s sequential write bandwidth which is several times faster than an SAS SSD can provide. With NVMe SSDs, we expected the Terasort execution time to considerably decrease since Terasort map/reduce phase performance highly depends on storage media sequential I/O performance.

We first conducted a DFSIO benchmark using an NVMe SSD. Figure 14 compares DFSIO execution time using a SAS SSD (denoted SSD) and an NVMe SSD (denoted NVMe). NVMe SSD reduces DFSIO read/write execution time by 33.9/26.5 % on average (by up to 36.4/33.5 %) as long as SSD. Despite NVMe SSDs being several times faster than SAS SSDs, this improvement was below expectation.

We determined that when using an NVMe SSD, network bandwidth throttles maximum storage device throughput. As described in Sect. 2.2.1, (full duplex) network bandwidth should be more than 0.67 times storage bandwidth. Based on our analysis, more than a 10 Gbit network bandwidth is required to fulfill an NVMe SSD's high-performance potential. In other words, 10 Gbit full duplex Ethernet network bandwidth supports less than 1880 MB/s storage throughput which looks sufficient for an NVMe SSD. Figure 15 shows the CPU, network, and I/O usage profile during the DFSIO write benchmark where the number of tasks per node is 32. Note that 10 Gbit Ethernet turns out to sustain throughput less than 1600 MB/s instead of 2500 MB/s in full duplex

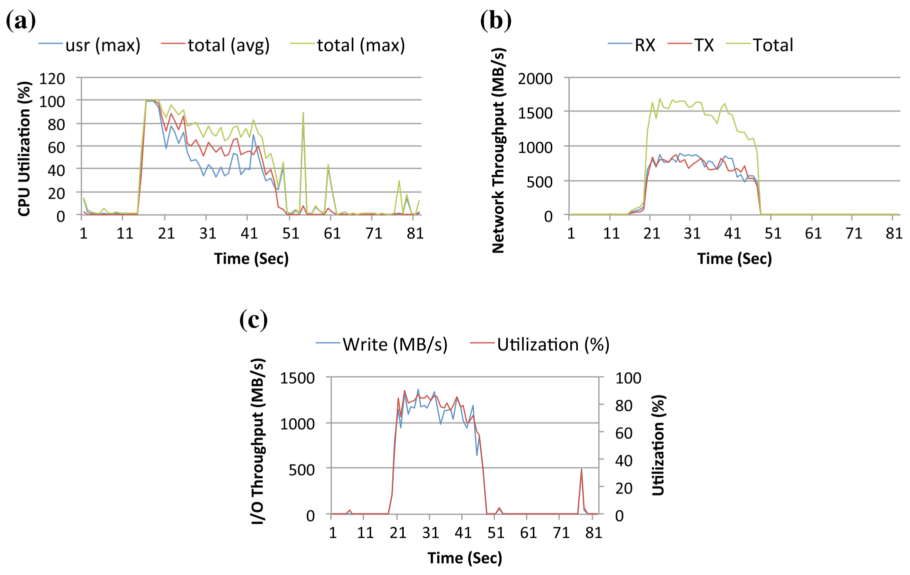
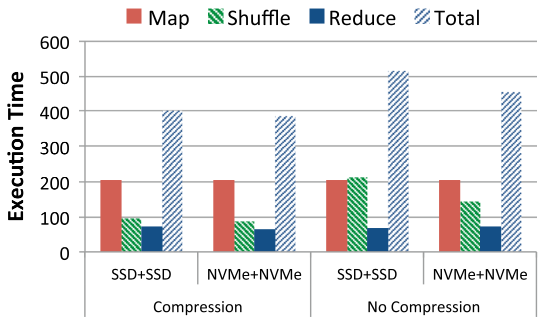


Fig. 15 DFSIO write, 32 tasks per node. **a** CPU usage profile, **b** network usage profile, **c** I/O usage profile

Fig. 16 Terasort (100 GB data) execution times for different configurations with 16 GB DRAM and 10 Gbit Ethernet. Map output compression option is enabled/disabled for each configuration



mode. Unlike the analysis result above, it is a cluster bottleneck and the network constrains storage bandwidth to less than 1.2 GB/s. In the figure, it is clearly shown that the 10 Gbit Ethernet's bandwidth is fully utilized while I/O and CPU bandwidths are not, and overall system performance is determined by the network bandwidth.

Figure 16 compares Terasort running times for the SSD + SSD and NVMe + NVMe configurations. Unfortunately, we could not see much improvement from NVMe SSDs. Unlike the DFSIO benchmark, we found this is due to the overloaded CPU in map phase in those configurations.

This result is not surprising. As explained in Sect. 3, the SSD + SSD configuration performance bottleneck is not the SSD but the CPU. In Figs. 8i and 9i, we see that the SSD is utilized less than 50 %. Because the CPU is a map phase bottleneck for both the SSD + SSD and the NVMe + NVMe configurations, the map phase performance for them is exactly equal.

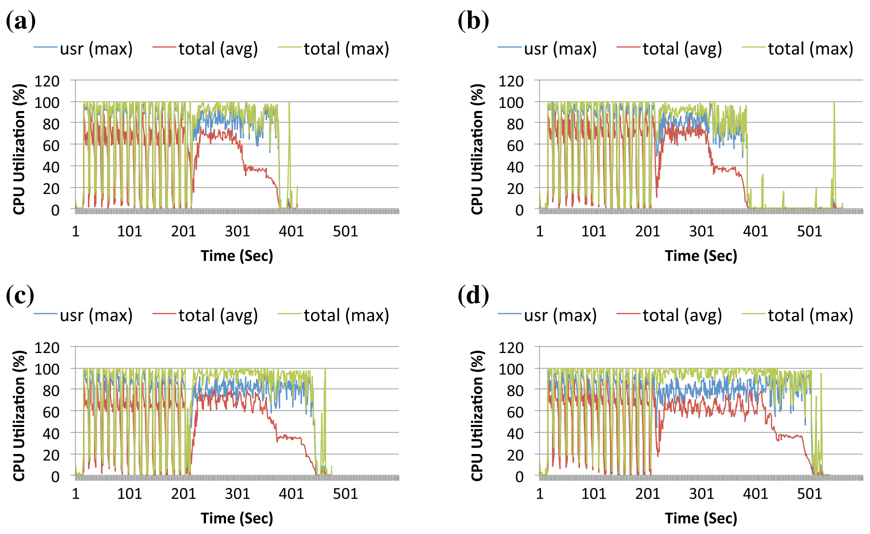


Fig. 17 Datanode CPU utilization during Terasort. **a** NVMe + NVMe, compress map output; **b** SSD + SSD, compress map output; **c** NVMe + NVMe, no compression; **d** SSD + SSD, no compression

Figure 17 illustrates the CPU utilization of the SSD + SSD and the NVMe + NVMe configurations. CPUs are mostly utilized for map phase user processing: 0–200 s in time (x-axis) with many peaks. To relieve the heavy user process CPU workloads, we disabled the compression option and ran Terasort again. However, deselecting the compression option does not reduce CPU utilization. It turns out that, without compression, CPUs write about five times more map output data. This equals the CPU utilization saving from disabling compression. Moreover, the larger amount of map output data significantly increases shuffle and reduce phase execution times.

Note that Terasort is a MapReduce application baseline because it does nothing but sorts key value pairs. Typical MapReduce applications may suffer more from CPU power shortages than from storage I/O throttling.

According to our experiment, it seems not effective to use faster SSDs in MapReduce frameworks. However, we cannot conclude that high-performance storage devices are always inefficient. The result shows just a typical case when high-performance storage devices are not a MapReduce workflow performance bottleneck. To enjoy maximum high-performance storage benefits, we can increase network bandwidth for HDFS I/O performance using Eq. (1) in Sect. 2.2.1. Moreover, since CPUs can be a MapReduce cluster bottleneck, more careful consideration when selecting configuration resources is required when designing Hadoop clusters employing high-end SSDs for MapReduce frameworks. Hadoop benefits can be optimized when storage performance simultaneously matches CPU capability and network bandwidth. In addition, the high-performance storage can benefits the mixed workloads where multiple jobs share the storage as observed in multi-thread experiments.

In summary, the NVMe SSD is expected to boost the MapReduce performance further. However, we have shown that considerable amount of network and CPU resources should be provided to support the maximal throughput of the NVMe SSD.

6 Related work

Related work [20,21] explores performance improvements achieved by combining SSDs and increasing network bandwidth. It correctly identifies network bandwidth as a potential HDFS bottleneck. However, our investigations include an HDFS throughput and network bandwidth quantitative analysis.

A recent work [22] investigates a cost-efficient Hadoop MapReduce Framework. The study compares a workstation cluster to a single high-performance server, while we compare HDD storage systems to SSD storage systems and to SSD/HDD hybrid storage systems. They simply examine execution time, while our study explores resource utilizations with extensively different configurations and identifies which factors delay overall system performance.

Another work [6] reveals multiple concurrent read/write requests can cause severe HDD performance degradation, significantly impacting Hadoop performance. It mitigates this problem by modifying I/O scheduling to HDFS storage devices. Our benchmarks also demonstrate that HDDs suffer from multiple concurrent I/Os and suggest using SSDs in places where concurrent I/Os concentrate. While other work focuses on micro-analyzing HDFS read/write primitives, we investigate end-to-end, system-wide performance.

Recent work [23] studied efficient employment of flash memory for a specific application (Facebook messages) in Hbase + HDFS. Similar to this paper, they also reveal that using flash tier is more cost-effective configuration than using RAM or HDDs in the application. In this paper, a different application (Hadoop MapReduce framework) from the related work is studied. While the related work built layered architecture of DRAM, flash memory and HDDs, this paper uses an SSD as intermediate storage which is simple to implement. In addition, many Hadoop configuration issues such as frequent heartbeats and JVM reuse options are discussed in this paper.

It is widely investigated in industry to enhance the performance of Hadoop with SSDs. A recent work [24] shows that using SSD for intermediate data is the cost-effective configuration. This paper originally studies the issue earlier and in detail with resource profiling analysis.

The work [25] built a resource profile suite for Hadoop and HDFS, and measured the performance of Hadoop with SSDs, whereas we exploit Linux mpstat and sysstat tools with our own automation scripts and focus more on finding cost-effective configuration with SSDs.

JVM and its garbage collection impact on Hadoop Terasort is explored in [26]. While the work studied a vanilla Hadoop, we focused on different Hadoop configurations with HDDs and SSDs.

Map phase compression is not always beneficial. One study [27] argued that disabling compression may be better for MapReduce framework energy efficiency.

7 Conclusion

This paper demonstrated that building balanced Hadoop systems via proper system configurations is critical to maximize potential SSD benefits. Sufficient network bandwidth and increased I/O parallelism to utilize maximum SSD performance are critical to HDFS and Hadoop MapReduce Framework performance. Specifically, SSDs help when they are used as temporary storage for intermediate data which consists of heavy random and concurrent I/O. We observed that using SSDs for temporary storage and HDDs for permanent HDFS storage is the most cost-effective configuration today. Judiciously mixing SSDs and HDDs within heterogeneous HDFS designs enables enterprises to leverage and capture SSD strengths and benefits.

We found that enterprise level SSDs (NVMe SSDs) cannot maximize their potential performance until other resources such as CPU and network bandwidth are sufficiently provided. It is advantageous to revisit Hadoop MapReduce system design periodically with consideration of balance of resource utilizations.

References

1. Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: OSDI'04: 6th symposium on operating system design and implementation, San Francisco
2. Ghemawat S, Gobioff H, Leung S (2003) The Google file system. In: SOSP'03: 19th ACM symposium on operating systems principles
3. Apache Hadoop Project. http://hadoop.apache.org. Accessed 21 May 2015
4. Shvachko K, Kuang H, Radia S, Chansler R (2010) The Hadoop distributed file system. In: MSST'10: 26th IEEE symposium on massive storage systems and technologies
5. Dell. Solid state drive vs. hard disk drive price and performance study. [White paper]
6. Shafer J, Rixner S, Cox A (2010) The Hadoop distributed filesystem: balancing portability and performance. In: ISPASS'10: IEEE international symposium on performance analysis of systems and software
7. Moon S, Lee J, Kee Y (2014) Introducing SSDs to Hadoop MapReduce Framework. In: IEEE Cloud' 14: 7th IEEE international conference on cloud computing
8. Flexible IO Tester. Available in http://git.kernel.dk/?p=fio.git;a=summary. Accessed 21 May 2015
9. DFSIO program. Available in Hadoop source distribution: src/test/org/apache/hadoop/fs/TestDFSIO. Accessed 21 May 2015
10. Linux iostat manual page. http://sebastien.godard.pagesperso-orange.fr/man_iostat.html. Accessed 21 May 2015
11. Terasort program. http://hadoop.apache.org/docs/r0.23.6/api/org/apache/hadoop/examples/terasort/package-summary.html. Accessed 21 May 2015
12. Twitter's Hadoop-LZO. http://github.com/twitter/hadoop-lzo. Accessed 21 May 2015
13. JProfiler, ej-technologies GmbH. https://www.ej-technologies.com/products/jprofiler/overview.html. Accessed 21 May 2015
14. Cloud Computing, Intel Inc., Optimizing Hadoop deployments. [White paper]
15. Intel Xeon Processor-Based Servers, Big data analytics, Intel Inc., Optimizing Hadoop Deployments. [White paper]
16. Hortonworks Inc., Best practices: Linux file systems for HDFS. http://hortonworks.com/kb/linux-file-systems-for-hdfs. Accessed 21 May 2015
17. White Tom (2012) Hadoop: the definitive guide. O'Reilly Media Inc, USA
18. NVM Express Interface. http://www.nvmexpress.org. Accessed 21 May 2015
19. Samsung Enterprise Class SSD Datasheet. http://www.samsung.com/global/business/semiconductor/file/product/XS1715_ProdOverview_2014_1.pdf
20. Sur S, Wang H, Huang J, Ouyang X, Panda D (2010) Can high-performance interconnects benefit Hadoop distributed file system. In: MASVDC'10: workshop on micro architectural support for virtualization, data center computing, and clouds in conjunction with MICRO'10
21. Islam N, Rahman M, Jose J, Rajachandrasekar R, Wang H, Subramoni H, Murthy C, Panda D (2012) High performance RDMA-based design of HDFS over InfiniBand. In: SC '12: the international conference on high performance computing, networking, storage and analysis

22. Appuswamy R, Gkantsidis C, Narayanan D, Hodson O, Rowstron A (2013) Scale-up vs scale-out for Hadoop: time to rethink? In: ACM symposium on cloud computing, 2 October 2013
23. Harter T, Borthakur D, Dong S, Aiyer A, Tang L, Arpaci-Dusseau A, Arpaci-Dusseau R (2014) Analysis of HDFS under HBase: a Facebook messages case study. In: FAST'14: 12th USENIX conference on file and storage technologies
24. SanDisk, Increasing Hadoop performance with SanDisk solid state drives (SSDs). [White paper]
25. Dai J, Huang J, Huang S, Huang B, Liu Y (2011) HiTune: dataflow-based performance analysis for big data cloud. In: Usenix ATC'11: USENIX annual technical conference
26. Joshi S, Liaskovitis V (2012) Java garbage collection characteristics and tuning guidelines for Apache Hadoop TeraSort workload. [White paper]
27. Chen Y, Ganapathi AS, Katz RH (2010) To compress or not to compress—compute vs. IO tradeoffs for MapReduce energy efficiency. Technical Report No. UCB/EECS-2010-36, University of California at Berkeley