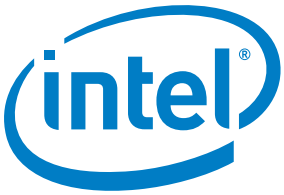

Optimize Hadoop Cluster Performance with Various Storage Media

This study covers HBase write performance on different storage media, leveraging the hierarchy storage management support in HDFS to store different categories of HBase data using the Yahoo! Cloud Serving Benchmark (YCSB) as the test workload.

Contents

Methodology **2**

Cluster Setup **3**

Stack Enhancement and Parameter Tuning **5**

Experimentation **7**

Issues and Improvements **17**

Conclusions **20**

About the Authors **21**

Acknowledgements **21**

Notice **22**

As more and more fast storage types (SSD, NVMe SSD, etc.) emerge, a methodology is necessary for better throughput and latency when using big data. However, these fast storage types are still expensive and are capacity limited. This study provides a guide for cluster setup with different storage media.

In general, this study considers the following questions:

1. What is the maximum performance a user can achieve by using fast storage?
2. Where are the bottlenecks?
3. How can the best balance be achieved between performance and cost?
4. How can the performance of a cluster with different storage combinations be predicted?

This study covers the HBase write performance on different storage media, leveraging the hierarchy storage management support in HDFS to store different categories of HBase data on different media.

Three different types of storage were evaluated:

- HDD: the most popular storage in current use.
- SATA SSD: a faster storage which is gaining popularity.
- RAMDISK: used to emulate the extremely high performance PCIe NVMe-based SSDs and upcoming faster storage (e.g. Intel 3D XPoint® based SSD). Due to hardware unavailability, RAMDISK was used to perform this emulation. The results hold for PCIe SSD and other fast storage types.

Note: RAMDISK is logical device emulated with memory. Files stored into RAMDISK are be cached in memory.

Methodology

The write performance in HBase was tested with a tiered storage in HDFS and compared to performance when storing different HBase data into different storages. Yahoo! Cloud Serving Benchmark (YCSB), a widely-used opensource framework for evaluating the performance of data serving systems, was used as the test workload.

Eleven test cases were evaluated in this study. The table was split into 210 regions in 1 TB dataset cases to avoid region split at runtime; the tables were presplit into 18 regions in 50 GB dataset cases.

The format of the case names is <dataset size>_<storage type>.

Table 1. Test Cases

Case Name	Storage	Dataset	Comment
50G_RAM	RAMDISK	50GB	Stored files in RAMDISK. Data size limited to 50GB due to the capacity limitation of RAMDISK.
50G_SSD	8 SSDs	50GB	Stored files in SATA SSD. Compared performance by 50GB data with 1st case.
50G_HDD	8 HDDs	50GB	Stored files in HDD.
1T_SSD	8 SSDs	1TB	Stored files in SATA SSD. Compared performance by 1TB data with cases in tiered storage.
1T_HDD	8 HDDs	1TB	Stored all files in HDD. Used 1TB.
1T_RAM_SSD	RAMDISK 8 SSDs	1TB	Stored files in tiered storage (i.e. different storage mixed together in one test); WAL was stored in RAMDISK; other files were stored in SSD.
1T_RAM_HDD	RAMDISK 8 HDDs	1TB	Stored files in tiered storage. WAL was stored in RAMDISK; other files were stored in HDD.
1T_SSD_HDD	4 SSDs 4 HDDs	1TB	Stored files in tiered storage. WAL was stored in SSD; some smaller files (less than 1.5GB) were stored in SSD and other files (including archived files) were stored in HDD.
1T_SSD_All_H DD	4 SSDs 4 HDDs	1TB	Stored files in tiered storage. WAL and flushed H Files were stored in SSD; all other files (including archived and compacted files) were stored in HDD.
1T_RAM_SSD_HDD	RAMDISK 4 SSDs 4 HDDs	1TB	Store files in tiered storage. WAL was stored in RAMDISK; some smaller files (less than 1.5GB) were stored in SSD and all other files (including archived files) were stored in HDD.
1T_RAM_SSD_All_HDD	RAMDISK 4 SSDs 4 HDDs	1TB	Store files in tiered storage, WAL was stored in RAMDISK and flushed HFiles were stored in SSD. All other files (including archived and compacted) were stored in HDD.

Note: In all 1TB test cases, the HBase table was presplit into 210 regions to avoid the region split at runtime.

The metrics in Table 2 are collected during the test for performance evaluation.

Table 2. Metrics collected during the tests	
Metrics	**Comment**
Storage media level	IOPS, throughput (sequential/random R/W)
OS level	CPU usage, network IO, disk IO, memory usage

Cluster Setup

In all, five nodes were used in the testing. Figure 1 shows the topology of these nodes; the services on each node are listed in **Error! Reference source not found.**.

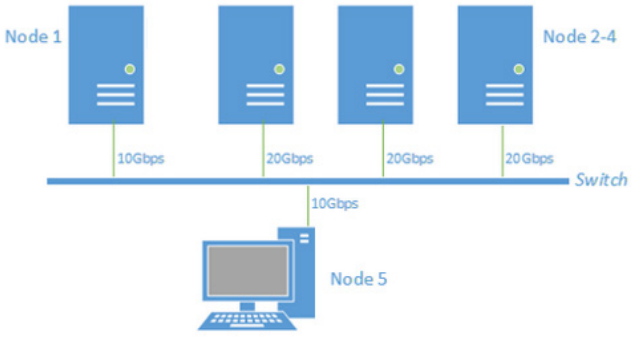

Figure 1. Cluster Topology

For HDFS, one node serves as NameNode and three nodes serve as DataNodes.

1. For HBase, one HMaster is co-located with NameNode. Three RegionServers are co-located with DataNodes.
2. All of the nodes in the cluster are connected to the full duplex 10Gbps DELL N4032 Switch.
3. Network bandwidth for clientNameNode, clientDataNode and NameNodeDataNode is 10Gbps. Network bandwidth between DataNodes is 20Gbps (20Gbps is achieved by network bonding).

Table 3. Services run on each node						
Node	**NameNode**	**DataNode**	**HMaster**	**RegionServer**	**ZooKeeper**	**YCSB Client**
1	√		√			
2		√		√	√	
3		√		√	√	
4		√		√	√	
5						√

Hardware

Hardware configurations of the nodes in the cluster are listed in the following tables.

Table 4. Hardware for DataNode/RegionServer	
Item	**Model/Comment**
CPU	Intel® Xeon® CPU E52695 v3 @ 2.3GHz, dual sockets
Memory	Micron 16GB DDR32133MHz, 384GB in total
NIC	Intel 10Gigabit X540AT2
SSD	Intel S3500 800G
HDD	Seagate Constellation™ ES ST2000NM0011 2T 7200RPM
RAMDISK	300GB

Note: OS is stored on an independent SSD (Intel® SSD DC S3500 240GB) for both DataNodes and NameNode. The number of SSD or HDD (OS SSD not included) in DataNode varies for different testing cases. See Methodology for details.

Table 5. Hardware for NameNode/HBase MASTER	
Item	**Model/Comment**
CPU	Intel® Xeon® CPU E52697 v2 @ 2.70GHz, dual sockets
Memory	Micron 16GB DDR32133MHz, 260GB in total
NIC	Intel 10Gigabit X540AT2
SSD	Intel S3500 800G

Table 6. Processor configuration	
Item	**Value**
Intel HyperThreading Tech	On
Tech On Intel Virtualization	Disabled
Intel Turbo Boost Technology	Enabled
Energy Efficient Turbo	Enabled

Software

Note: As mentioned in Methodology, HDFS and HBase have been enhanced to support this test.

Table 7. Software stack version and configuration	
Software	**Details**
OS	CentOS release 6.5
Kernel	2.6.32431.el6.x86_64
Hadoop	2.6.0
HBase	1.0.0
Zookeeper	3.4.5
YCSB	0.3.0
JDK	jdk1.8.0_60
JVM Heap	NameNode: 32GB DataNode 4GB HMaster: 4GB RegionServer: 64GB
GC	G1GC

Benchmarks

For a benchmark, YCSB 0.3.0 was used; one YCSB client was used in the tests.

This is the workload configuration for 1T dataset:

```
# cat ../workloads/1T_workload

fieldcount=5

fieldlength=200

recordcount=1000000000

operationcount=0

workload=com.yahoo.ycsb.workl
oads.CareWorkload

readallfields=true
```

The following command was used to start the YCSB client:

```
./ycsb load hbase10P
../workloads/1T_workload
threads 200 p
columnfamily=family p
clientbuffering=true s >
1T_workload.dat
```

Stack Enhancement and Parameter Tuning

Stack Enhancement

To perform the study, the following set of enhancements in the software stack was used:

- HDFS:
  - Support a new storage RAMDISK.
  - Add file level mover support: users could move blocks per file without scanning all metadata in NameNode.
- HBase:
  - WAL, flushed HFiles, HFiles generated in compactions, and archived HFiles can be stored in different storage.
  - When renaming HFiles across storage, the blocks of that file were moved to the target storage asynchronously.

HDFS/HBase Tuning

This step is to find the best configurations for HDFS and HBase.

Known Key Performance Factors in HBase

The key performance factors in HBase are:

1. **WAL:** write ahead log to guarantee the non-volatility and consistency of data. Each record inserted into HBase must be written to WAL which can slow down user operations due to its latency sensitivity.

2. **Memstore and Flush:** records inserted into HBase were cached in memstore; when a threshold was reached the memstore was flushed to a store file. Slow flush can lead to high GC (Garbage Collection) pause, and make memory usage reach the thresholds in regions and region server, which can block user operations.

3. **Compaction and Number of Store Files:** compacts small store files to a larger one which can reduce the number of store files and accelerate the reading; it can generate heavy I/O and consume the disk bandwidth in runtime. Less compaction can accelerate the writing but generates too many store files, which slow down the reading. When there are too many store files, the memstore flush can be slowed down which can lead to a large memstore and further slow the user operations.

Based on this understanding, Table 8 lists the tuned parameters used.

Table 8. Parameters

HDFS configuration	
Property	**Value**
dfs.datanode.handler.count	64
dfs.namenode.handler.count	100
HBase configuration	
Property	**Value**
hbase.regionserver.thread.compaction.small	3 for nonSSD test cases. 8 for all SSD related test cases.
hbase.hstore.flusher.count	5 for nonSSD test cases. 15 for all SSD related test cases.
hbase.wal.regiongrouping.numgroups	4
hbase.wal.provider	Multiwall
hbase.hstore.blockingStoreFiles	15
hbase.regionserver.handler.count	200
hbase.hregion.memstore.chunkpool.maxsize	1
hbase.hregion.memstore.chunkpool.initialsize	0.5

Experimentation

Performance for Single Storage Type

First, the HBase write performance was studied on each single storage type (i.e. no storage type mix). This test shows the maximum performance each storage type can achieve and provides a guide to following hierarchy storage analysis.

The single storage type performance was studied with two data sets: 50GB and 1TB data insertion. The 1TB is a reasonable data size in practice. The 50GB is used to evaluate the performance uplimit as HBase performance is typically higher when the data size is small. The 50GB size was chosen to avoid data out of space in test. Further, due to RAMDISK limited capacity, a small data size was used when storing all data in RAMDISK.

50GB Dataset in a Single Storage

The throughput and latency by YCSB for 50GB dataset are listed in Figure 2 and Figure 3. As expected, storing 50GB data in RAMDISK had the best performance whereas storing data in HDD had the worst.

Note: Performance data for SSD and HDD may be better than its real capability as OS can buffer/cache data in memory.

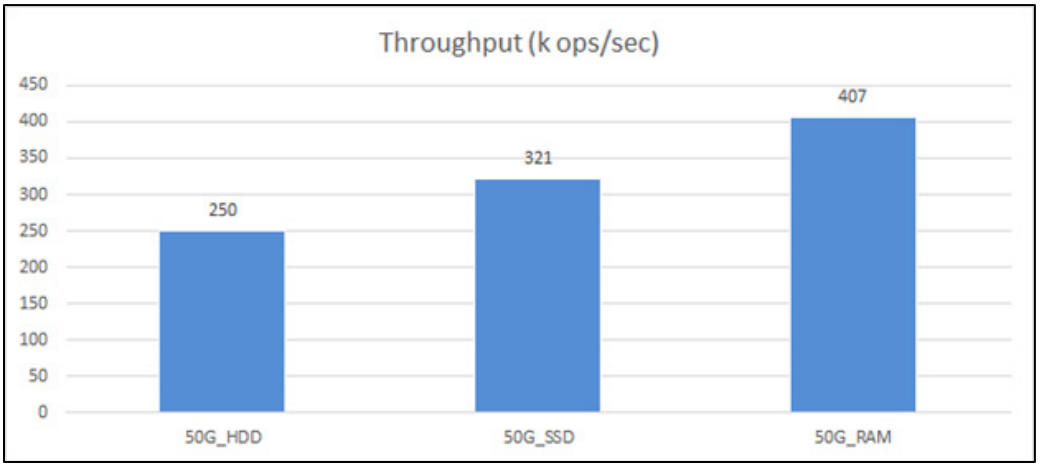

Figure 2. YCSB throughput of a single storage with 50GB dataset

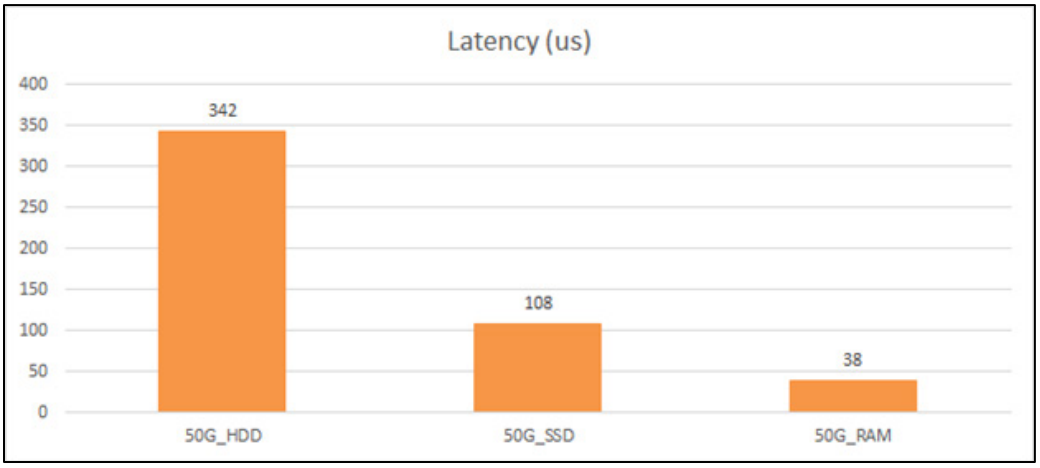

Figure 3. YCSB latency of a single storage with 50GB dataset

Note: Performance data for SSD and HDD may be better than its real capability as OS can buffer/cache data in memory.

For throughput, RAMDISK and SSD were higher than HDD (163% and 128% of HDD throughput, respectively). But the average latencies were dramatically lower than HDD (only 11% and 32% of HDD latency). This is expected because HDD has long latency on seek operation. The latency improvements of using RAMDISK, SSD over HDD were bigger than throughput improvement. This was caused by the huge access latency gap of different storage. The latency measured from accessing 4KB raw data from RAM, SSD and HDD were respectively 89ns, 80µs and 6.7ms (RAM and SSD were about 75000x and 84x faster than HDD).

Now consider the data of hardware (disk throughput, network throughput, and CPU utilization in each DataNode) collected during the tests.

Disc Throughput

Table 9. Disk utilization of test cases

	Throughput theoretically (MB/s)	Throughput measured (MB/s)	Utilization
50G_HDD	127 x 8 = 1016	870	86%
50G_SSD	447 x 8 = 3576	1300	36%
50G_RAM	>11194	1800	<16%

Note: The data listed in the table for 50G_HDD and 50G_SSD is the total throughput of 8 disks.

The disk throughput was decided by factors such as access model, block size, and I/O depth. The theoretical disk throughput was measured by using performance-friendly factors; in real cases they may not be that friendly and would limit the data to lower values. In fact, the disk utilization usually increased to 100% for HDD.

Network Throughput

Each DataNode was connected by a 20Gbps (2560MB/s) full duplex network, which means both receive and transmit speed could reach 2560MB/s simultaneously. In the tests, the receive and transmit throughput are almost identical, so only the receive throughput data is listed in **Error! Reference source**

not found.

Table 10. Network utilization of test cases			
	Throughput theoretically (MB/s)	Throughput measured (MB/s)	Utilization
50G_HDD	2560	760	30%
50G_SSD	2560	1100	43%
50G_RAM	2560	1400	55%

CPU Utilization

Clearly, the performance on HDD (50G_HDD) was bottlenecked on disk throughput. However, for SSD and RAMDISK, neither disk, network, or CPU were the bottleneck. The bottlenecks must be in the software (e.g. HBase compaction, memstore in regions, etc.) or in the hardware (except disk, network, and CPU).

Table 11. CPU utilization of test cases	
50G_HDD	36%
50G_SSD	55%
50G_RAM	60%

To further understand why the utilization of SSD was so low and throughput was not as high as expected (only 132% of HDD, considering SSD had much higher theoretical throughput: 447 MB/s vs. 127MB/s per disk), an additional test was performed to write 50GB data on four SSDs per node instead of eight SSDs per node.

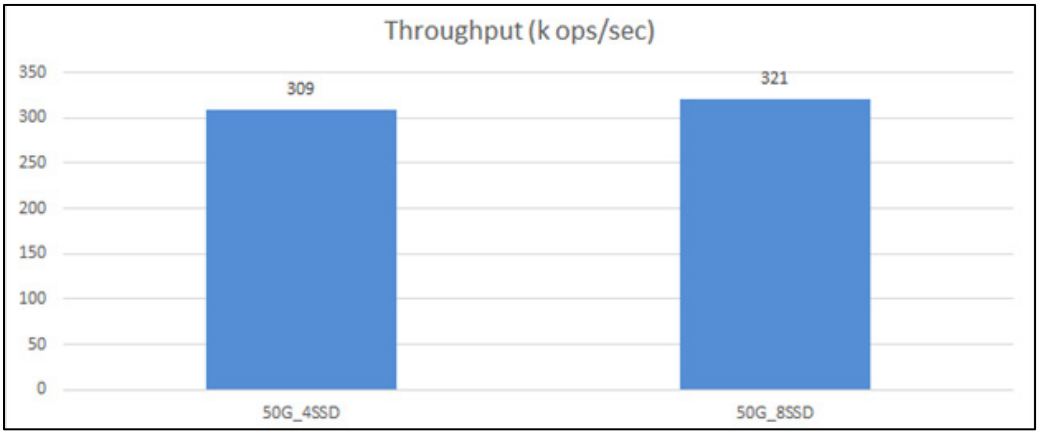

Figure 4. YCSB throughput of a single storage type with 50 GB dataset stored in different number of SSDs

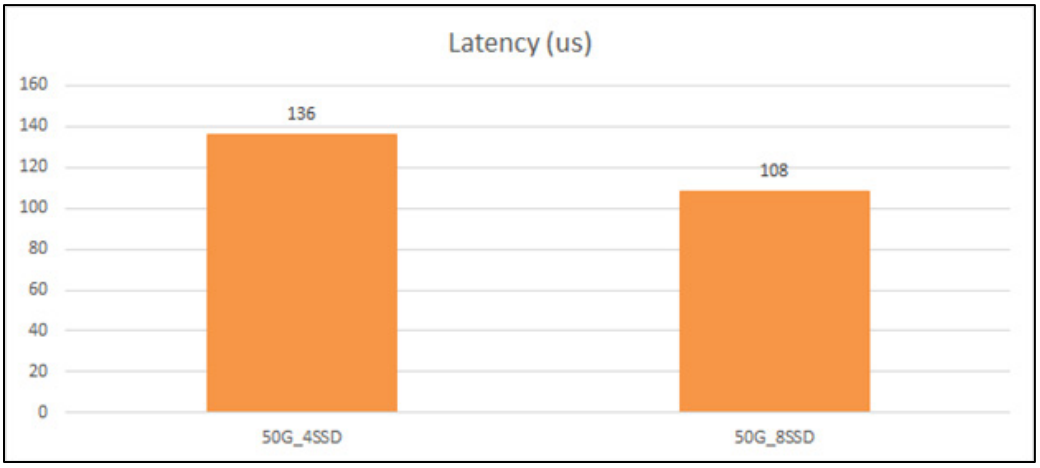

Figure 5. YCSB latency of a single storage type with 50 GB dataset stored in different number of SSDs

While the number of SSDs doubled, the throughput and latency of eight SSDs per node case (50G_8SSD) improved to a much lesser extent (104% throughput and 79% latency) compared to four SSDs per node case (50G_4SSD). This means that the ability of SSDs were far from full use. In further examination it was found to be caused by current mainstream server hardware design.

The mainstream server had two SATA controllers which could support up to 12 Gbps bandwidth. This means that the total disk bandwidth was limited to about 1.5GB/s, or the bandwidth of approximately 3.4 SSDs. For details, see Disk Bandwidth Limitation. SATA controllers have become the bottleneck for eight SSDs per node. This explained why there was almost no improvement on YCSB throughput for eight SSDs per node. In the 50G_8SSD test, the disk was the bottleneck.

1TB Dataset in a Single Storage

The performance for 1TB dataset in HDD and SSD is shown in Figure 6 and Figure 7. Due to the limitation of memory capability, 1TB dataset in RAMDISK was not tested.

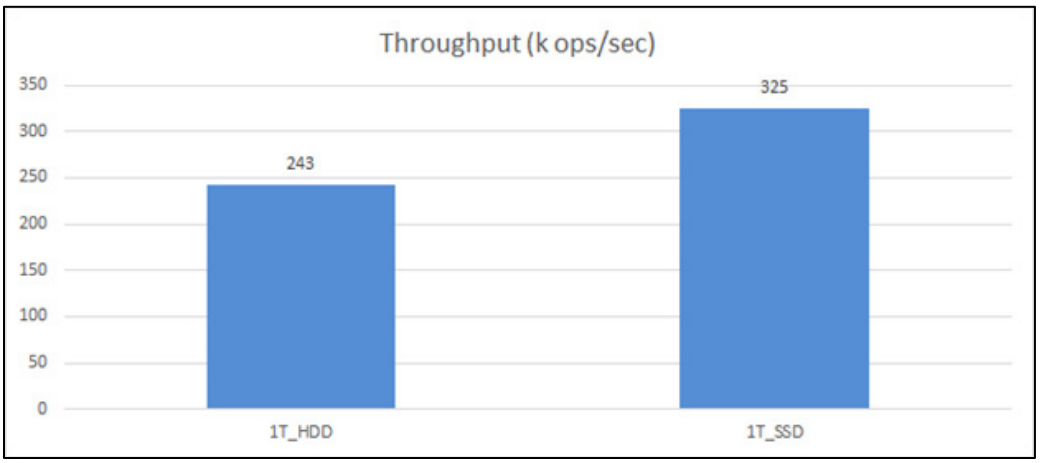

Figure 6. YCSB throughput of a single storage type with 1TB dataset

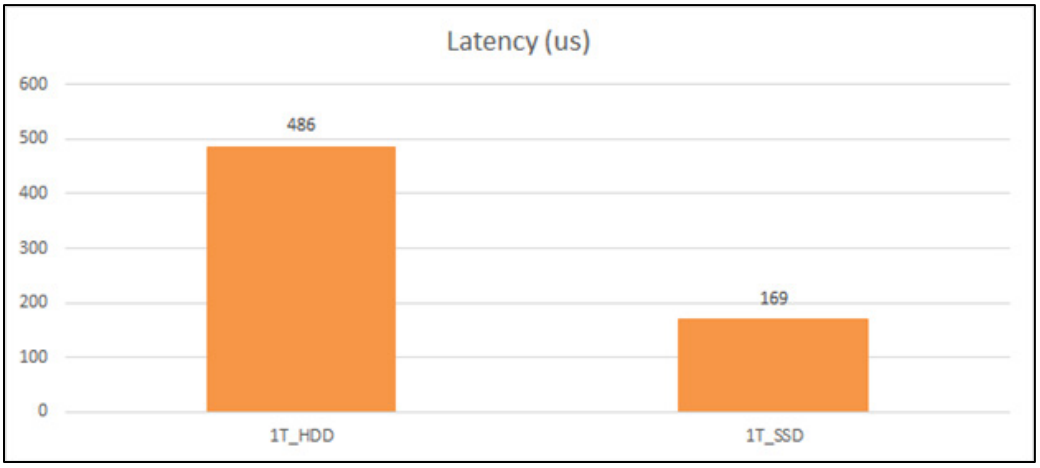

Figure 7. YCSB latency of a single storage type with 1TB dataset

The throughput and latency on SSD were both better than HDD (134% throughput and 35% latency).

This is consistent with the 50GB data test.

The benefits gained for throughput by using SSD were different between 50GB and 1TB (from 128% to 134%): SSD gained more benefits in the 1TB test. This was because much more I/O intensive events such as compactions occurred in the 1TB dataset test than in the 50GB test. This showed the superiority of SSD in huge data scenarios. Figure 8 shows the changes of the network throughput during the tests.

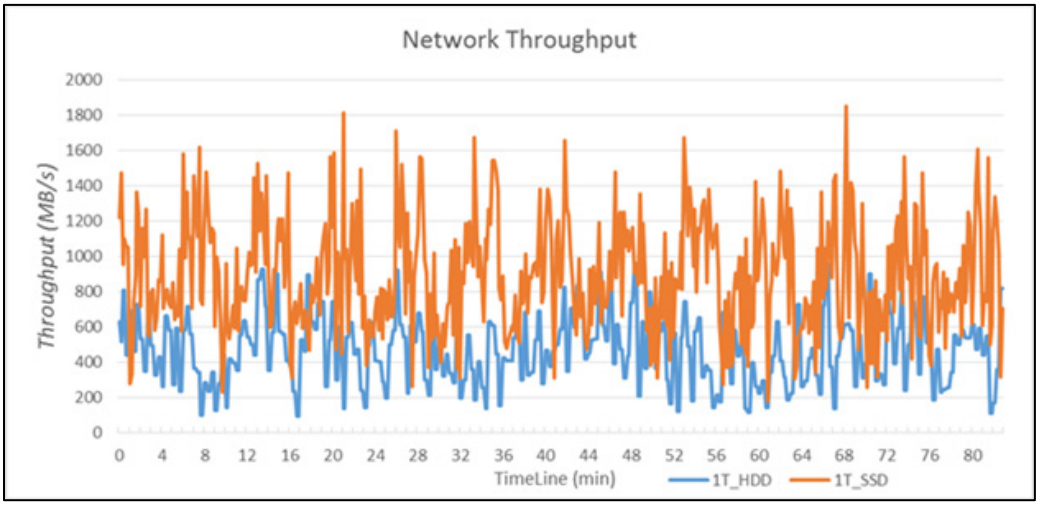

Figure 8. Network throughput measured for case 1T_HDD and 1T_SSD

In the 1T_HDD test case, the network throughput was lower than 10Gbps, and in the 1T_SSD test case the network throughput could be much larger than 10Gbps. This means if a 10Gbps switch was used in the 1T_SSD test case, the network should have been the bottleneck.

Figure 9 shows that the bottleneck for these two cases is disk bandwidth.

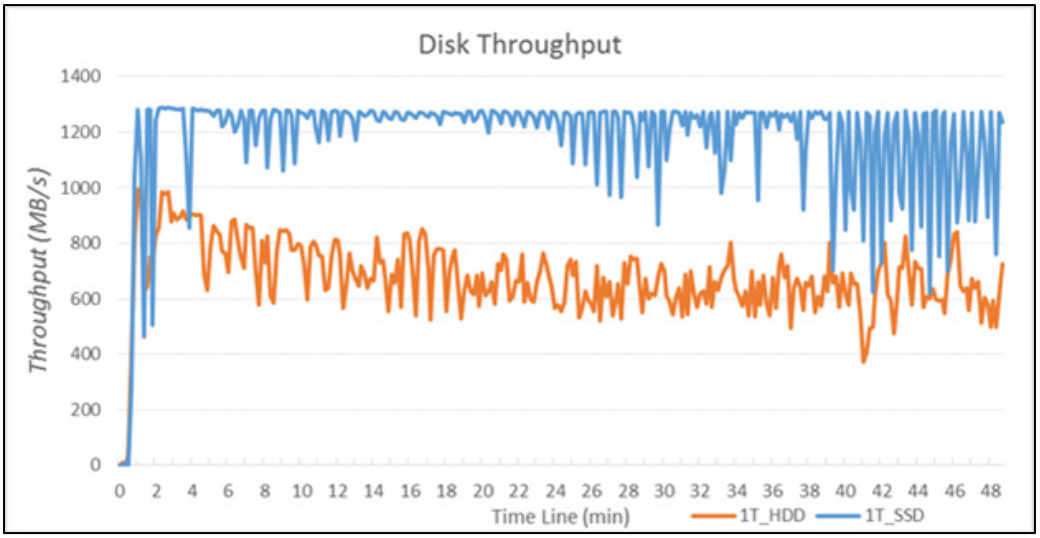

Figure 9. Disk throughput measured for case 1T_HDD and 1T_SSD

- In 1T_HDD, at the beginning of the test the throughput is almost 1000 MB/s, but after a while the throughput drops down due to memstore limitation of regions caused by slow flush.
- In 1T_SSD case, the throughput seems to be limited by a ceiling of around 1300 MB/s, nearly the same with the bandwidth limitation of SATA controllers. To further improve the throughput, more SATA controllers are needed (e.g. using HBA card) instead of more SSDs are needed.

During the 1T_SSD test case, the operation latencies on eight SSDs per node were very different as shown in Figure 9. Figure 10 shows only the latency of two disks: sdb represents disks with a high latency and sdf represents disks with a low latency.

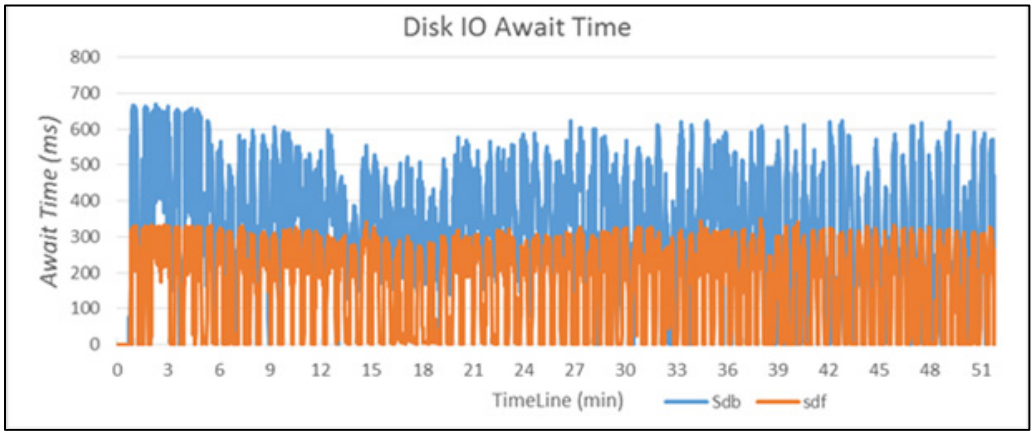

Figure 10. I/O await time measured for different disks

Four of the disks had a better latency than the others. This was caused by the hardware design issue. For details, see Disk I/O Bandwidth and Latency Varies for Ports. The disks with higher latency might have taken the same workload as the disks with lower latency in the existing VolumeChoosingPolicy; this would slow down the performance. It is recommended to implement a Latencyaware VolumeChoosingPolicy in HDFS.

Performance Estimation for RAMDISK with 1TB Dataset

The performance of RAMDISK with 1T dataset could not be measured due to RAMDISK's limited capacity. Instead, its performance was evaluated by analyzing the results of test cases HDD and SSD. The performance between 1TB and 50GB dataset were fairly close in HDD and SSD. The throughput difference between 50GB and 1TB dataset for HDD was |250034 |×100% .89%

$$\frac{242801}{250034} - 1 \mid x\ 100\% = 2.89\%$$

while the SSD value is:

$$\frac{325148}{320616} - 1 \mid x\ 100\% = 1.41\%$$

If an average of the above values is used as the throughput decrease in RAMDISK between 50GB and 1TB dataset, it is around 2.15% ((2.89%+1.41%)/2=2.15%). Thus the throughput for RAMDISK with 1T dataset should be:

06577×(1 4 + 2.15%) = 415318 (ops/sec)

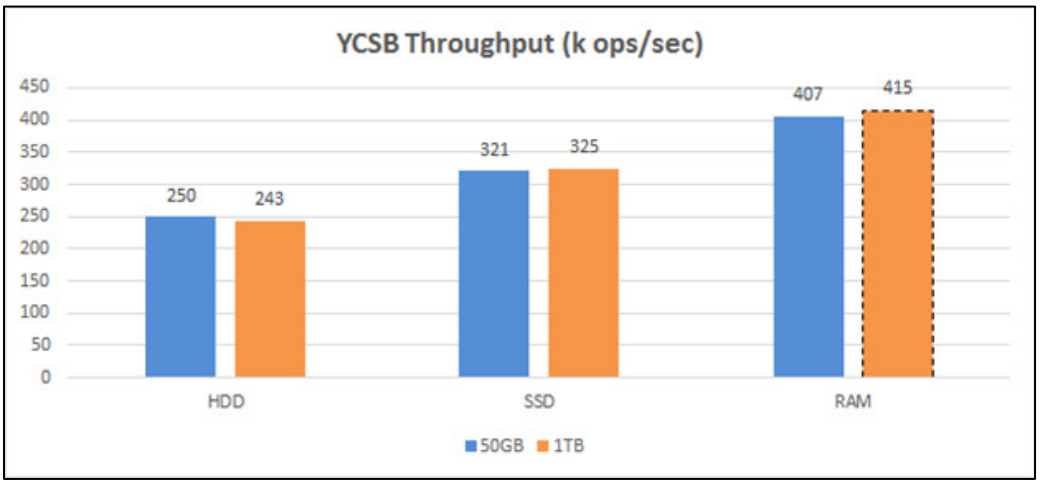

Figure 11. YCSB throughput estimation for RAMDISK with 1TB dataset

Note: The throughput does not drop much in 1TB dataset cases compared to 50 GB dataset cases because they do not use the same number of presplit regions. The table is presplit to 18 regions in 50 GB dataset cases, and it is presplit to 210 regions in the 1 TB dataset.

Performance for Tiered Storage

In this section, the HBase write performance is studied on tiered storage (i.e. different storage mixed together in one test). This would show what performance can be achieved by mixing fast and slow storage together, and help to conclude the best balance of storage between performance and cost. Figure 12 and Figure 13 show the performance for tiered storage. You can find the description of each case in Table 1.

Most of the cases that introduced fast storage had better throughput and latency. With no surprise, 1T_RAM_SSD had the best performance among them. The real surprise was that the throughput of 1T_RAM_HDD was worse than 1T_HDD (11%), the throughput of 1T_RAM_SSD_All_HDD was worse than 1T_SSD_All_HDD (2%) after introducing RAMDISK, and the throughput of 1T_SSD was worse than 1T_SSD_HDD (2%).

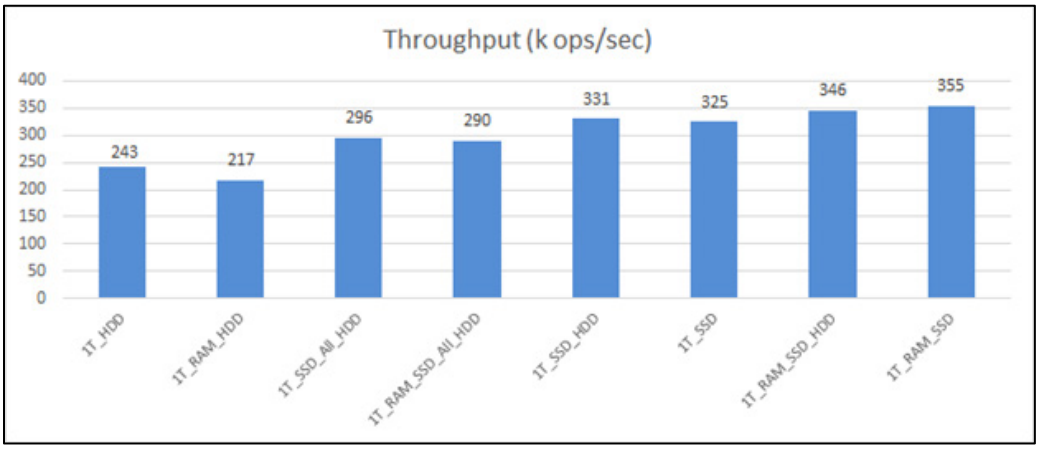

Figure 12. YCSB throughput data for tiered storage

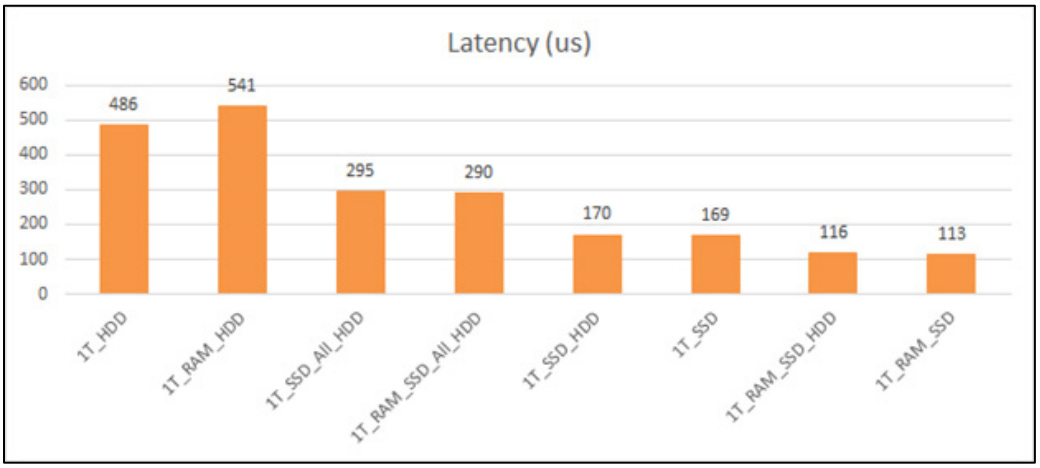

Figure 13. YCSB latency data for tiered storage

The amount of data written to different storage types was analyzed by collecting information from one DataNode. As shown in Figure 14, generally, more data was written to disks for test cases with higher throughput. Fast storage could accelerate the flush and compaction, which could lead to more flushes and compactions. Thus, more data was written to disks. In some RAMDISKrelated cases, only WAL could be written to RAMDISK, and there were 1216 GB WALs written to one DataNode.

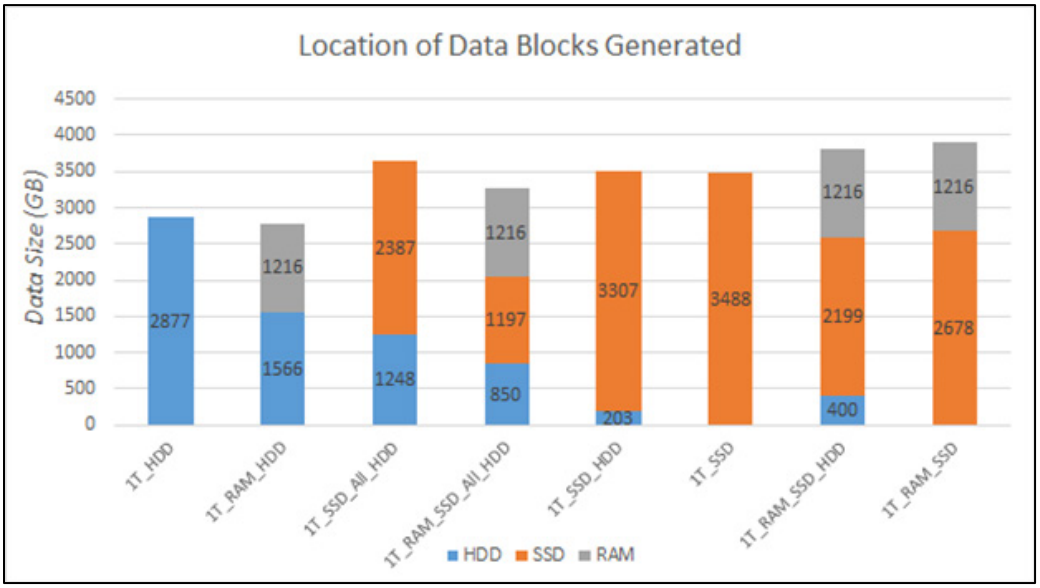

Figure 14. Distribution of data blocks on each storage of HDFS in one DataNode

For tests without SSD (1T_HDD and 1T_RAM_HDD), the number of flush and compaction actions was intentionally limited by using fewer flushers and compactors. This was due to limited IOPs capability of HDD, which could lead to fewer flush and compactions. Too many concurrent reads and writes could hurt HDD performance which eventually slowed down the performance.

Many BLOCKED DataNode threads could be blocked up to tens of seconds in 1T_RAM_HDD. This occurred in other cases but it was more frequently observed in 1T_RAM_HDD test cases. This was because each DataNode holds one big lock when creating and finalizing HDFS blocks; these methods might have taken tens of seconds sometimes (see Longtime BLOCKED threads in DataNode), the more these methods were used (in HBase they were used in flusher, compactor, and WAL), the more often the BLOCKED occurred. Writing WAL in HBase needed to create/finalize blocks which could be blocked, and consequently users' inputs are blocked. Multiple WAL with a large number of groups or WAL per region might also encounter this problem, especially in HDD.

Considering the written data distribution, performance results in Figure 12 and Figure 13 should be reviewed:

1. Mixing SSD and HDD can greatly improve the performance (136% throughput and 35% latency) compared to pure HDD. But fully replacing HDD with SSD does not show an improvement (98% throughput and 99% latency) over mixing SSD/HDD. This is because the hardware design cannot evenly split the I/O bandwidth to all eight disks, and 94% data are written in SSD while only 6% data are written to HDD in SSD/HDD mixing case. This strongly hints a mix use of SSD/HDD can achieve the best balance between performance and cost. More information is in Disk Bandwidth Limitation and Disk I/O Bandwidth and Latency Varies for Ports .
2. Including RAMDISK in SSD/HDD tiered storage has different results with 1T_RAM_SSD_All_HDD and 1T_RAM_SSD_HDD. The case 1T_RAM_SSD_HDD shows a result when there are only a few data written to HDD, which improves the performance over SSD/HDD mixing cases. The results of 1T_RAM_SSD_All_HDD when there was a large number of data written to HDD was worse than SSD/HDD mixing cases. This means if the data is distributed appropriately to SSD and HDD in HBase, a good performance can be gained when mixing RAMDISK/SSD/HDD.

3. The RAMDISK/SSD tiered storage is the winner of both throughput and latency (109% throughput and 67% latency of pure SSD case). If cost is not an issue and maximum performance is desired, RAMDISK/SSD should be chosen.

The throughput decreased by 11% by comparing 1T_RAM_HDD to 1T_HDD. This was initially because 1T_RAM_HDD used RAMDISK which consumed part of the RAM, and resulted in the OS buffer having less memory to cache the data.

Further, with 1T_RAM_HDD, the YCSB client can push data at very high speed, cells are accumulated very fast in memstore while the flush and compaction in HDD are slow, the RegionTooBusyException occurs more often (Figure 15 shows a much larger memstore in 1T_RAM_HDD than 1T_HDD), and a much longer GC pause was observed in 1T_RAM_HDD than 1T_HDD (up to 20 seconds in a minute).

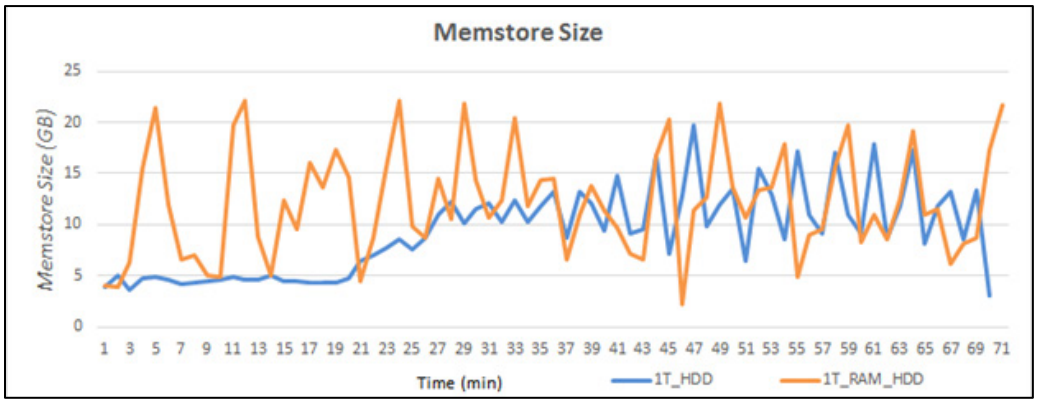

Figure 15. Memstore size in 1T_RAM_HDD and 1T_HDD

Finally, as the number of flushers and compactors was increased, the performance decreased because of the reasons mentioned in Longtime BLOCKED threads in DataNode. The performance reduction in 1T_RAM_SSD_All_HDD than 1T_SSD_All_HDD (2%) is due to the same reasons mentioned above.

The following are recommended:

1. Use reasonable configurations for flusher and compactor, especially in HDDrelated cases.
2. Do not use the storage with large performance gaps, (i.e. directly mixing RAMDISK and HDD).
3. In many cases, the long GC pause was about 10 seconds per minute. To solve long GC pause issues, an offheap memstore in HBase should be implemented.
4. Implement a finer grained lock mechanism in DataNode.

Issues and Improvements

This section describes the issues found during tests.

Software

Longtime BLOCKED threads in DataNode

In the 1T_RAM_HDD test, a substantial 0 throughput period was observed in the YCSB client. After delving into the thread stack, many threads of DataXceiver were found to be stuck in a BLOCKED state for a long time in DataNode. Although this was observed in other cases, it most often occurred in the 1T_RAM_HDD test.

In each DataNode, there was a single instance of FsDatasetImpl where there were many synchronized methods, DataXceiver threads used this instance to achieve synchronization when creating/finalizing blocks. A slow creating/finalizing operation in one DataXceiver thread could block other creating/finalizing operations in all other DataXceiver threads. Table 12 shows the time consumed by these operations in the 1T_RAM_HDD test.

Table 12. DataXceiver threads and time consumed

Synchronized methods	Max exec time (ms) in light load	Max wait time (ms) in light load	Max exec time (ms) in heavy load	Max wait time (ms) in heavy load
finalizeBlock	0.0493	0.1397	17652.9761	21249.0423
createRbw	0.0480	1.7422	21145.8033	56918.6570

Both execution time and wait time on the synchronized methods increased dramatically with the increment of the system load. The time to wait for locks was as much as tens of seconds. Slow operations usually came from the slow storage such as HDD; this could hurt the concurrent operations of creating/finalizing blocks in fast storage so that HDFS/HBase cannot make better use of tiered storage.

A finer grained lock mechanism in DataNode is needed to fix this issue and is being worked on (HDFS9668).

Load Imbalance in HDFS with Tiered Storage

In tiered storage cases, the utilization was not the same among volumes of the same storage type when using the policy RoundRobinVolumeChoosingPolicy. The root cause was that in RoundRobinVolumeChoosingPolicy, a shared counter was used to choose volumes for all storage types. It might be unfair when choosing volumes for a certain storage type, so the volumes in the tail of the configured data directories had a lower chance to be written.

The situation becomes even worse when there are different numbers of volumes of different storage types. A patch is currently available and a more permanent fix is being developed (HDFS9608).

Asynchronous File Movement across Storage When Renaming in HDFS

Currently data blocks in HDFS are stored in different types of storage media according to pre-specified storage policies when creating. After that data blocks will remain where they were until an external tool Mover in HDFS is used. Mover scans the whole namespace, and moves the data blocks that are not stored in the right storage media as the policy specifies.

In a tiered storage, when a file or directory is renamed from one storage to another, the blocks of that file or all files must be moved from under that directory to the right storage. This is not currently provided in HDFS.

Non-configurable Storage Type and Policy

Currently in HDFS both storage type and storage policy are predefined in source code. This makes it inconvenient to add implementations for new devices and policies. It is better to make them configurable.

No Optimization for Certain Storage Type

Currently there is no difference in the execution path for different storage types. As more and more high performance storage devices are adopted, the performance gap between storage types will become larger, and the optimization for certain types of storage will be needed.

For example, consider writing certain numbers of data into HDFS. To minimize the total time to write, the optimal way for HDD may be using compression to save disk I/O, while for RAMDISK writing directly is more suitable as it eliminates the overheads of compression. This scenario requires configurations per storage type, but it is not supported in the current implementation.

Hardware

Disk Bandwidth Limitation

In the section 50GB Dataset in a Single Storage, the performance difference between four SSDs and eight SSDs is very small. The root cause is the total bandwidth available for the eight SSDs is limited by upper level hardware controllers. Figure 16 illustrates the motherboard design. The eight disk slots connect to two different SATA controllers (Ports 0:3 SATA and Port 0:3 sSATA). As highlighted in the red rectangle, the maximum bandwidth available for the two controller in the server is 2*6 Gbps =1536 MB/s.

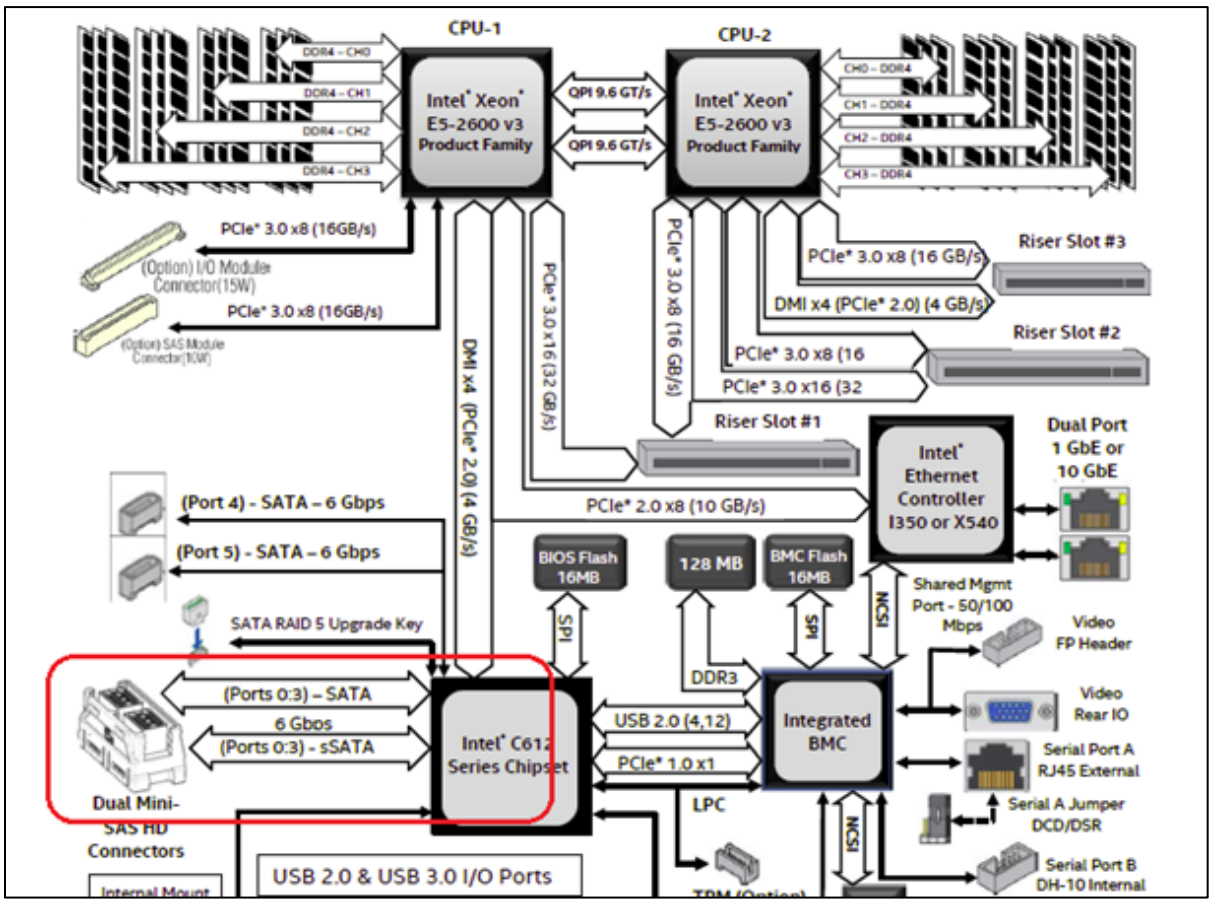

Figure 16. Hardware design of server motherboard

Maximum throughput for single disk is measured with the FIO tool (see Table 13).

Table 13. Maximum throughput of storage medias

	Read BW (MB/s)	Write BW (MB/s)
HDD	140	127
SSD	481	446
RAMDISK	>12059	>11194

Note: RAMDISK is memory essentially and it does not go through the same controller as SSDs and HDDs, so it does not have the 2*6Gbps limitation. Data of RAMDISK is listed in the table for convenience of comparison.

According to Table 13, the writing bandwidth of eight SSDs is 447 x 8 = 3576 MB/s. It exceeds the controllers' 1536 MB/s physical limitation, thus only 1536 MB/s are available for all eight SSDs. HDD is not affected by this limitation as their total bandwidth (127 x 8 = 1016 MB/s) is below the limitation. This fact greatly impacts the performance of the storage system. Try one of the following recommendations:

- Enhance hardware by using more HBA cards to eliminate the limitation of the design issue.
- Use SSD and HDD together with an appropriate ratio (for example four SSDs and four HDDs) to achieve a better balance between performance and cost.

Disk I/O Bandwidth and Latency Varies for Ports

As described in the section Disk Bandwidth Limitation, four of the eight disks connect to Ports 0:3 - SATA and the rest of the disks connect to Port 0:3 – sSATA; the total bandwidth of the two controllers is 12Gbps. The bandwidth is not evenly divided by the disk channels. With each test, each SSD (sda, sdb, sdc, and sdd connect to Port 0:3 sSATA, sde, sdf, sdg, and sdh connect to Ports 0:3 SATA) is written by an individual FIO process. It is expected that eight disks are written at the same speed, but according to the output of IOSTAT the bandwidth 1536 MB/s is not evenly divided by two controllers and eight disk channels. As shown in Figure 17, the four SSDs connected to Ports - 0:3 SATA obtain more I/O bandwidth (213.5MB/s*4) than the others (107MB/s*4).

When setting up a cluster, consider the controller limitation and storage bandwidth. Using four SSDs and four HDDs in a node is a reasonable choice, and it is better to install the four SSDs to Port 0:3 SATA.

Additionally, the disk with higher latency might take the same workload as the disks with lower latency in the existing VolumeChoosingPolicy. This slows down the performance; try implementing a latency-aware VolumeChoosingPolicy in HDFS.

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          0.18    0.00    0.21    1.38    0.00   98.23

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await  svctm  %util
sda               0.00     0.00    0.00  214.00     0.00   107.00  1024.00    63.92  298.35   4.67 100.00
sdc               0.00     0.00    0.00  214.00     0.00   107.00  1024.00    63.93  298.34   4.67 100.00
sdd               0.00     0.00    0.00  214.00     0.00   107.00  1024.00    63.91  298.24   4.67 100.00
sdb               0.00     0.00    0.00  214.00     0.00   107.00  1024.00    63.86  298.33   4.67  99.90
sde               0.00     0.00    0.00  424.00     0.00   212.00  1024.00    63.85  150.30   2.36 100.00
sdf               0.00     0.00    0.00  427.00     0.00   213.50  1024.00    63.84  149.41   2.34 100.00
sdg               0.00     0.00    0.00  427.00     0.00   213.50  1024.00    63.86  149.44   2.34 100.00
sdh               0.00     0.00    0.00  427.00     0.00   213.50  1024.00    63.85  149.45   2.34 100.00
sdi               0.00     0.00    0.00    1.00     0.00     0.00     8.00     0.00    1.00   0.00   0.00
dm-0              0.00     0.00    0.00    1.00     0.00     0.00     8.00     0.00    1.00   1.00   0.10
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
dm-2              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
```

Figure 17. Different write speed and await time of disks

Conclusions

There are many things to consider when choosing the hardware of a cluster. According to the test results, in the SSDrelated cases the network utility between DataNodes is larger than 10Gbps. If you are using a 10Gbps switch, the network will be the bottleneck and impact the performance. It is recommended to extend the network bandwidth by network bonding, or upgrade to a more powerful switch with a higher bandwidth. In cases 1T_HDD and 1T_RAM_HDD, the network utility is lower than 10 Gbps in most time, using a 10 Gbps switch to connect DataNodes is fine.

In all 1T dataset tests, 1T_RAM_SSD shows the best performance. An appropriate mix of different types of storage can improve the HBase write performance. First, write the latencysensitive and blocking data to faster storage, and write the data that are rarely compacted and accessed to slower storage. Second, avoid mixing types of storage with a large performance gap, such as with 1T_RAM_HDD.

The hardware design issue limits the total disk bandwidth which makes there is hardly superiority of eight SSDs than four SSDs. Either to enhance hardware by using HBA cards to eliminate the limitation of the design issue for eight SSDs or to mix the storage appropriately. According to the test results, in order to achieve a better balance between performance and cost, using four SSDs and four HDDs can achieve a good performance (102% throughput and 101% latency of eight SSDs) with a much lower price. The RAMDISK/SSD tiered storage is the winner of both throughput and latency among all the tests, so if cost is not an issue and maximum performance is needed, RAMDISK(extremely high speed block device, e.g. NVMe PCIE SSD)/SSD should be chosen.

You should not use a large number of flusher/compactor when most of data are written to HDD. The read and write shares the single channel per HDD; too many flushers and compactors at the same time can slow down the HDD performance.

During the tests, some things were found that can be improved in both HBase and HDFS. In HBase, the memstore is consumed quickly when the WALs are stored in fast storage; this can lead to regular long GC pauses. It is better to have an offheap memstore for HBase. In HDFS, each DataNode shares the same lock when creating/finalizing blocks. Any such slow operations in one DataXceiver can block any other operations of creating/finalizing blocks in other DataXceiver on the same DataNode no matter what storage they are using. the blocking access must be eliminated across storage, and a finer-grained lock mechanism to isolate the operations on different blocks is needed (HDFS9668). And it will be good to implement a latencyaware VolumeChoosingPolicy in HDFS to remove the slow volumes from the candidates. RoundRobinVolumeChoosingPolicy can lead to load imbalance in HDFS with tiered storage (HDFS9608).

In HDFS, renaming a file to a different storage does not move the blocks indeed; in these cases, the HDFS blocks must be moved asynchronously.

About the Authors

Jingcheng Du is a Software Engineer at Intel and an HBase contributor.
Wei Zhou is a Software Engineer at Intel and an HDFS contributor.

Acknowledgements

The authors would like to thank Weihua Jiang – who is the previous manager of the big data team in Intel – for leading this performance evaluation, and thank Anoop Sam John(Intel), Apekshit Sharma(Cloudera), Jonathan Hsieh(Cloudera), Michael Stack(Cloudera), Ramkrishna S. Vasudevan(Intel), Sean Busbey(Cloudera) and Uma Gangumalla(Intel) for the nice review and guidance.

Notice

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document. Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 18005484725, or go to: http://www.intel.com/design/literature.htm

334463-001US