最优化Spark应用的性能

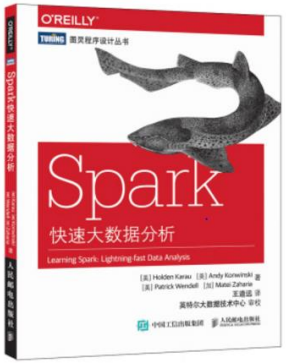
-- 使用高性价比的层次化方案加速大数据处理

Yucai, Yu (yucai.yu@intel.com)

BDT/STO/SSG April, 2016

About me/us

- Me: Spark contributor, previous on virtualization, storage, OS etc.
- Intel Spark team, working on Spark upstream development and x86 optimization, including: core, Spark SQL, Spark R, GraphX, machine learning etc.
- Top 3 contribution in 2015, 3 committers.
- Two publication:



Agenda

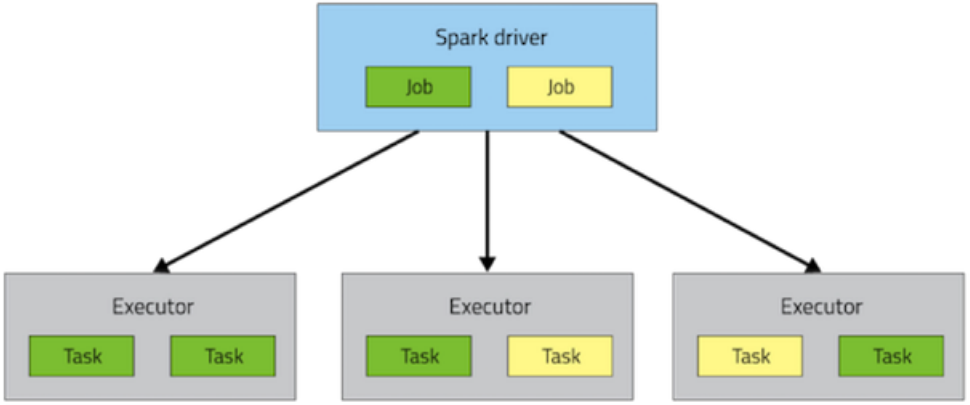
- General software tuning
- Bring up 3x performance with NVMe SSD
  - NVMe SSD Overview
  - Use NVMe SSD to accelerate computing
  - Why SSD is important to Spark

General software tuning

- Resource allocation
- Serialization
- Partition
- IO
- MISC

Resource Allocation - CPU

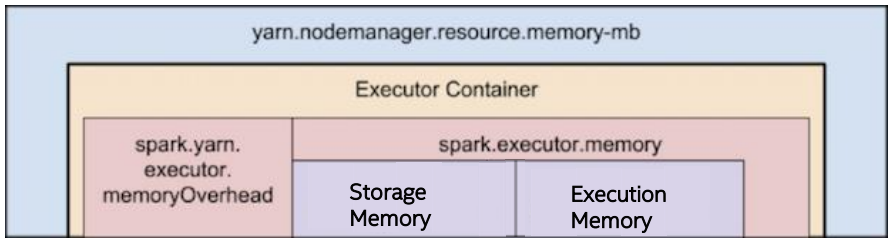
- **spark.executor.cores** – recommend **5** cores per executor*
  - Less core number (like single-core per executor) introduces JVM overhead. e.g., multiple broadcast copies
  - More cores number may hard to apply big resource
  - To achieve full write throughput onto HDFS
- **Number of executors per node** – cores per node / 5 * (1 ~ 0.9)



* http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/

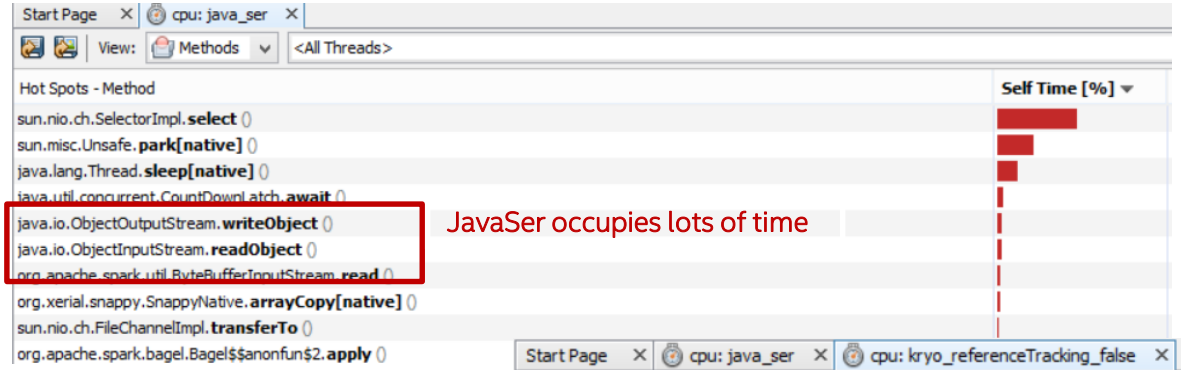
Resource Allocation - Memory

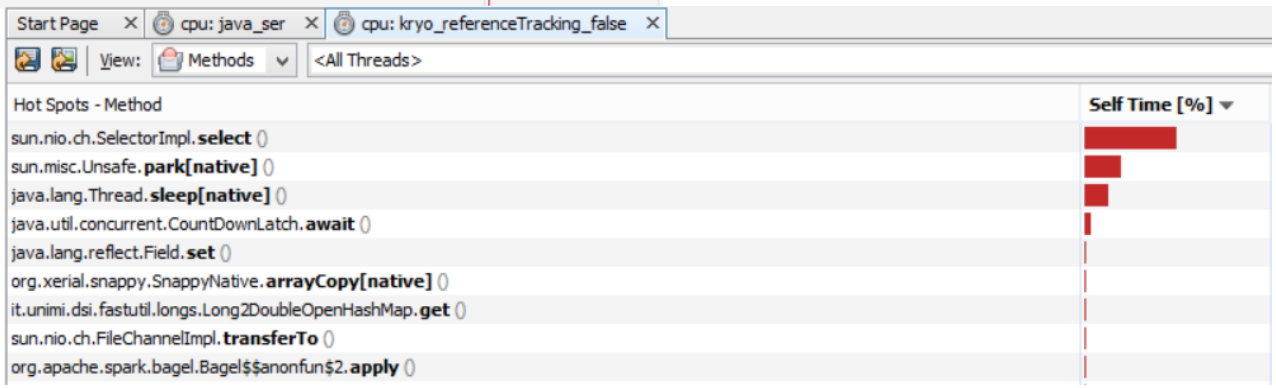
- spark.executor.memory – memory size per executor
  - Leave 10-15% total memory for OS cache: dcache, page cache etc.
  - memory per node * (85-90)% / executors per node
  - 2-5 GB per core: 2-5GB * spark.executor.cores
- spark.yarn.executor.memoryOverhead – indicate for offheap memory size, increasing that to avoid killing by yarn NM
  - Sometimes default value is too small as max(384, .07 * spark.executor.memory), netty may use these heavily
  - yarn.nodemanager.resource.memory-mb = spark.yarn.executor.memoryOverhead + spark.executor.memory



Software Tuning - Serialization

- **spark.serializer** – Class to use for serializing objects
  - using **kryo** when speed is necessary, it brings **15-20%** improvement

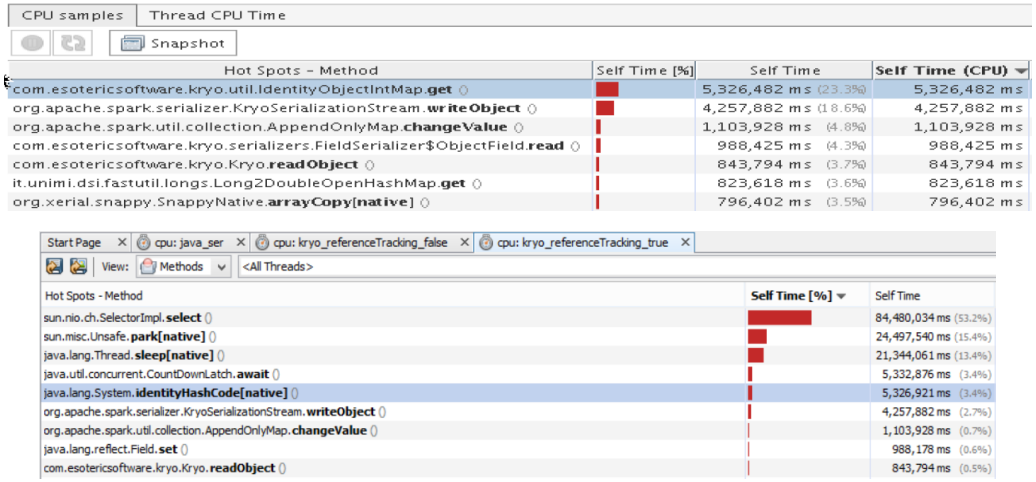




Software Tuning - Serialization

- **spark.kryo.referenceTracking** – Disabling to avoid java performance bug.
  - Kryo serializer has a mechanism for reducing repeated information in records by keeping a cache of previously seen objects. That cache can be surprisingly expensive because JDK's identityHashCode has bad performance issue.

    http://bugs.java.com/view_bug.do?bug_id=6378256
  - People see the same issue also: https://www.factual.com/blog/profiling-hadoop-jobs-with-riemann



JDK-6378256 : Performance problem with System.identityHashCode in client compiler *

Details

Type: Enhancement
Status: In Progress
Project Name: JDK
Component: hotspot
Sub-Component: compiler
Priority: P3
Resolution: Unresolved
Affected Versions: 6,8,9
Targeted Versions: 9

Submit Date: 2006-01-28
Updated Date: 2015-10-30
Resolved Date:
OS: linux
CPU: x86

Related Reports

Relates: JDK-6182955 - Enum.hashCode() could be seven times faster

Sub Tasks

Description

```
As noticed in bug JDK-6182955, there appears to be a performance problem
when calling System.identityHashCode.  See that bug for a test case.
```

* It is a JDK bug actually, not related to complier

Software Tuning – Partition

- Tasks number are decided by RDD's partition number.
- How to choose proper partition number?
  - If there are fewer partition than available cores, the tasks won't be taking advantage of all CPU.
  - Fewer partition, bigger data size, it means that more memory pressure especially in join, cogroup, *ByKey etc.
  - If the number is too large, more tasks, more iterative, more time.
  - Too large also puts more pressure in disk. When shuffle read, it leads to more small segment to fetch, especially worse in HDDs.
  - Set a big number to make application run success, decrease it gradually to reach best performance point, pay attention to the GC.
  - Sometimes, changing partition number to avoid data incline, checking this info from WebUI.

Software Tuning – IO

- Storage Level
  - MEMORY_ONLY get the best performance most of time
  - MEMORY_ONLY_SER reduces the memory consumption by serialize objects but need use CPU
  - MEMORY_AND_DISK, DISK_ONLY: if data is large, you need use those two options
- Compression
  - spark.rdd.compress, spark.shuffle.compress, spark.shuffle.spill.compress: trade off between CPU and disk
  - spark.io.compression.codec: lz4, lzf, and snappy

Software Tuning - MISC

- **spark.dynamicAllocation.enabled –** To scale up or down executor number on-demand
  - Must turn on **spark.shuffle.service.enabled**, otherwise some shuffle data will be lost
  - To free or occupy executor resources timely, e.g., spark-shell, SQL
- **GC – GC easily leads to straggler issue**
  - Measuring the impact of GC
    - WebUI (4040/18080), stage page -> “GC Time”
  - Ease GC pressure
    - Reduce parallelism number
    - Increase partition number
    - Lower spark.memory.storageFraction
    - Increase memory

Agenda

- General software tuning
- Bring up 3x performance with NVMe SSD
  - NVMe SSD Overview
  - Use PCIe SSD to accelerate computing
  - Why SSD is important to Spark

Agenda

- General software tuning
- Bring up 3x performance with NVMe SSD
  - NVMe SSD Overview
  - Use NVMe SSD to accelerate computing
  - Why SSD is important to Spark

NVMe SSD Overview

- 突破性的性能

面向 PCIe* 的英特尔® 固态盘数据中心家族具备多达 2.8GB/秒和 460K IOPS（每秒输入/输出操作）的数据读取性能，以及多达 2.0GB/秒和 175K IOPS 的数据写入性能。

面向 PCIe* 的英特尔® 固态盘数据中心家族具备极其卓越的性能，只需一块硬盘便能够取代七块通过 HBA 聚合在一起的 SATA 固态盘（约 500K IOPS），其中尤以英特尔® 固态盘数据中心 P3700 的性能最为出色 (460K IOPS)。对于性能为 200 IOPS 的传统机械硬盘而言，需要组合使用 2,300 块 15K 机械硬盘，才能媲美一块面向 PCIe* 的英特尔® 固态盘数据中心家族。

- 插卡式（AIC）

- 2.5 英寸 U.2 (SFF8369 支持热插拔)

Agenda

- General software tuning
- Bring up 3x performance with NVMe SSD
  - NVMe SSD Overview
  - Use NVMe SSD to accelerate computing
  - Why SSD is important to Spark

Motivation

- 使用NVMe SSD和服务器上已有的机械硬盘搭建一个层次化外存系统：用SSD来优先缓存需要放到外存上的数据，这主要包括Shuffle相关的数据以及从内存中evict出来的RDD等，当SSD的容量不够时，再使用机械硬盘的空间。
- 期望SSD的高带宽和高IOPS可以为Spark 应用带来性能上的提升。

Motivation

- 使用NVMe SSD和服务器上已有的机械硬盘搭建一个层次化外存系统：用SSD来优先缓存需要放到外存上的数据，这主要包括Shuffle相关的数据以及从内存中evict出来的RDD等，当SSD的容量不够时，再使用机械硬盘的空间。
- 期望SSD的高带宽和高IOPS可以为Spark 应用带来性能上的提升。
- 业界也早已有层次化存储的探索，比如Tachyon（现在已更名为Alluxio），但它只能缓存RDD型的数据，不能缓存Shuffle 的数据。

Implementation

[SPARK-12196][Core] Store/retrieve blocks in different speed storage devices by hierarchy way #10225

- Spark和外存交互的接口是文件。也就是说当一个内存数据块需要放到外存去的时候，Spark Core就是在外存上生成一个文件，然后把这个数据块写到这个文件里面去。所以，我们只要修改Spark Core的文件分配算法，让它优先从SSD上分配，就很简单地就实现外存的层次化存储。
- Yarn dynamical allocation is supported also.

Usage

1. Set the priority and threshold in spark-default.xml.

```
spark.storage.hierarchyStore     ssd 30GB
```

2. Configure “ssd” location: just put the keyword like "ssd“ in local dir. For example, in yarn-site.xml:

```
<property>
  <name>yarn.nodemanager.local-dirs</name>
  <value>file:///mnt/DP_disk1/yucai/yarn/local,file:///mnt/DP_disk2/yucai/yarn/local,
         file:///mnt/DP_disk3/yucai/yarn/local,file:///mnt/DP_disk4/yucai/yarn/local,
         file:///mnt/DP_disk5/yucai/yarn/local,file:///mnt/DP_disk6/yucai/yarn/local,
         file:///mnt/DP_disk7/yucai/yarn/local,file:///mnt/DP_disk8/yucai/yarn/local,
         file:///mnt/DP_disk9/yucai/yarn/local,file:///mnt/DP_disk10/yucai/yarn/local,
         file:///mnt/DP_disk11/yucai/yarn/local, file:///mnt/nvme0n1/yucai/yarn.local.ssd,
  </value>
</property>
```

Real world Spark adoptions

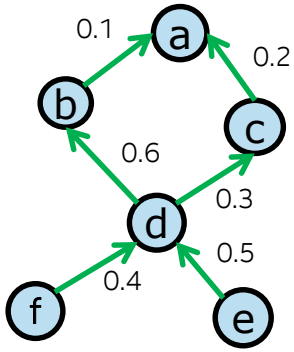
Workload	Category	Description	Rationale	Customer
NWeight	Machine Learning	To compute associations between two vertices that are *n-hop* away	Iterative graph-parallel algorithm, implemented with Bagel, GraphX.	Youku
SparkSQL	SQL	Real analysis queries from Baidu NuoMi.	Left Outer Join, Group BY, UNION etc.	Baidu

NWeight Introduction

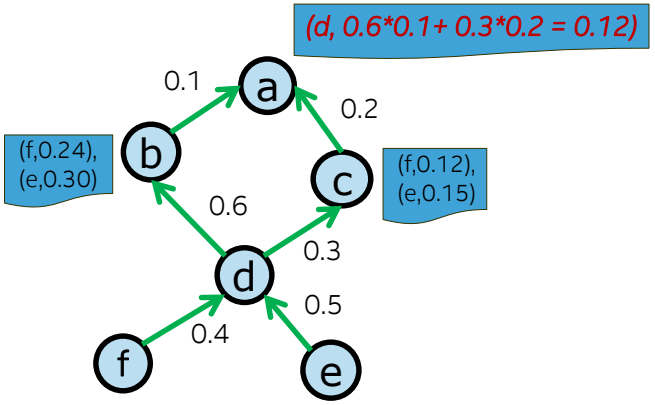
To compute associations between two vertices that are n-hop away.

e.g., friend –to-friend, or similarities between videos for recommendation

Initial directed graph

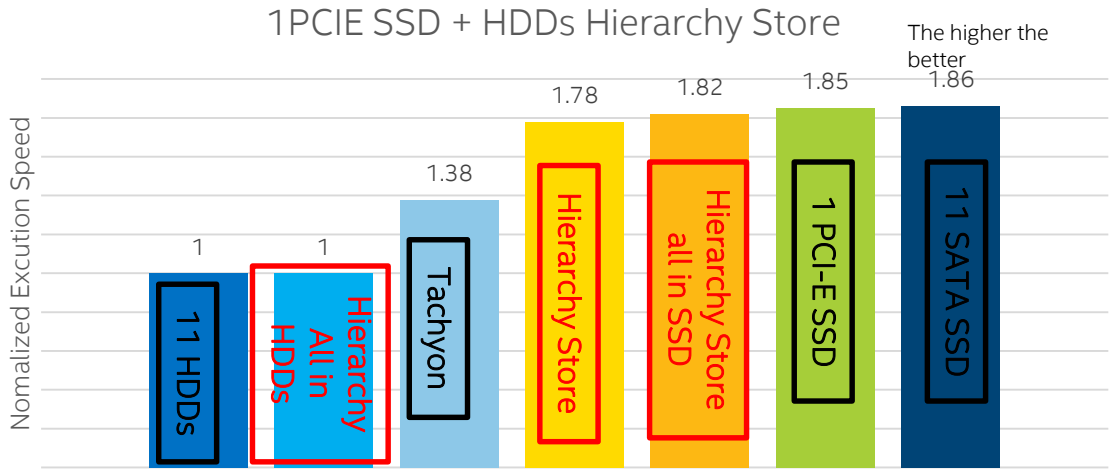


2-hop association



Hierarchy store performance report

- No extra overhead: best case the same with pure SSD (PCIe/SATA SSD), worst case the same with pure HDDs.
- Compared with 11 HDDs, *x1.82* improvement at least (CPU limitation).
- Compared with Tachyon, still shows *x1.3* performance advantage: cache both RDD and shuffle, no inter-process communication.



An Analysis SQL from Baidu NuoMi

Run 8 instance simultaneously

Stage Id	Tasks	Input	Shuffle Read	Shuffle Write
22	42685	61.1 KB		4.8 KB
20	26/26	3.1 GB		1982.7 MB
19	62/62	30.3 MB		15.2 MB
16	42685	61.1 KB		4.8 KB
14	26/26	3.1 GB		1982.7 MB
12	497/497	56.9 GB		36.0 GB
11	100/100	178.2 MB		77.6 MB
8	42685	61.1 KB		4.8 KB
6	26/26	3.1 GB		1982.7 MB
5	273/273	203.6 MB		145.7 MB

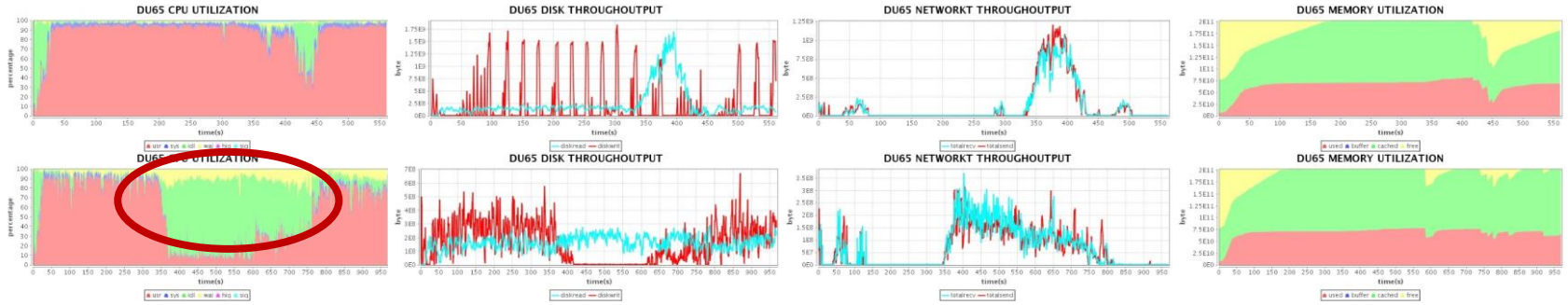
Stage Id	Tasks	Input	Shuffle Read	Shuffle Write
24	320/320		15.8 MB	
23	960/960		88.2 MB	15.8 MB
18	320/320		32.5 MB	45.1 MB
21	320/320		1997.9 MB	3.5 MB
17	320/320		16.9 MB	32.5 MB
15	320/320		2.1 GB	16.9 MB
13	320/320		36.0 GB	206.3 MB
10	320/320		26.7 MB	39.9 MB
9	320/320		16.6 MB	26.7 MB
7	320/320		2.1 GB	16.6 MB

SSD v.s. HDDs: x1.7 end-to-end improvement

x1.7 end-to-end improvement (7 mins v.s. 12 mins)
x5 shuffle improvement (1 min v.s. 5 mins)

x6 Disk BW

x4 network BW



Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	svctm	%util
nvme0n1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
sda	0.00	0.00	0.00	9.00	0.00	36.00	8.00	0.00	0.00	0.00	0.00
sdc	2.00	0.00	398.00	0.00	37268.00	0.00	187.28	7.09	15.63	2.46	97.90
sdb	19.00	347.00	389.00	8.00	35236.00	1420.00	184.66	10.37	25.28	2.45	97.30
sdd	19.00	0.00	326.00	23.00	29592.00	11776.00	237.07	153.35	137.34	2.87	100.00
sde	0.00	0.00	317.00	0.00	30344.00	0.00	191.44	2.87	9.02	2.54	80.40
sdf	11.00	0.00	267.00	1.00	25332.00	4.00	189.07	12.98	50.60	3.73	100.00
sdg	18.00	332.00	384.00	0.00	34684.00	0.00	180.65	25.56	47.58	2.60	100.00
sdh	4.00	183.00	334.00	5.00	33392.00	752.00	201.44	4.86	14.35	2.69	91.30

Significant IO bottleneck

Agenda

- General software tuning
- Bring up 3x performance with NVMe SSD
  - NVMe SSD Overview
  - Use PCIe SSD to accelerate computing
  - Why SSD is important to Spark

Deep dive into a real customer case

NWeight

x2-3 improvement!!!

						11 HDDs	PCIe SSD
Stage Id	Description	Input	Output	Shuffle Read	Shuffle Write	Duration	Duration
23	saveAsTextFile at BagelNWeight.scala:102+details	50.1 GB	27.6 GB			27 s	20 s
17	foreach at Bagel.scala:256+details	732.0 GB		490.4 GB		*23 min*	*7.5 min*
16	flatMap at Bagel.scala:96+details	732.0 GB			490.4 GB	15 min	13 min
11	foreach at Bagel.scala:256+details	590.2 GB		379.5 GB		*25 min*	11 min
10	flatMap at Bagel.scala:96+details	590.2 GB			379.6 GB	12 min	10 min
6	foreach at Bagel.scala:256+details	56.1 GB		19.1 GB		4.9 min	3.7 min
5	flatMap at Bagel.scala:96+details	56.1 GB			19.1 GB	1.5 min	1.5 min
2	foreach at Bagel.scala:256+details			15.3 GB		38 s	39 s
1	parallelize at BagelNWeight.scala:97+details					38 s	38 s
0	flatMap at BagelNWeight.scala:72+details	22.6 GB			15.3 GB	46 s	46 s

5 Main IO pattern

	Map Stage	Reduce Stage
RDD	rdd_read_in_map	rdd_read_in_reduce rdd_write_in_reduce
Shuffle	shuffle_write_in_map	shuffle_read_in_reduce

How to do IO characterization?

- We use blktrace* to monitor each IO to disk. Such as:

```
8,32    5     13561     43.740003038 13615  D   W 52090704 + 560 [java]
8,32    5     13567     43.741721414 13615  D   R 13637888 + 256 [java]
8,32    5     13569     43.741778543 13615  C   R 13637888 + 256 [0]
8,32    5     13570     43.743476793 13615  C   W 52089584 + 560 [0]
8,32    5     13571     43.743519464 13615  D   W 52091264 + 560 [java]
8,32    5     13577     43.743901725 13615  D   R 13638144 + 256 [java]
8,32    5     13579     43.745785309 13615  C   W 52090144 + 560 [0]
8,32    5     13585     43.746306216 13615  D   R 13638400 + 256 [java]
8,32    5     13587     43.748340765     0  C   W 52090704 + 560 [0]
8,32   17      5938     43.761506738 13449  D   R 50251520 + 8 [java]
```

Start to write 560 sectors from address 52090704

Start to read 256 sectors from address 13637888
Finish the previous read command (13637888 + 256)

Finish the previous write command (52090704 + 560)

- We parse those raw info, generating 4 kinds of charts: IO size histogram, latency histogram, seek distance histogram and LBA timeline, from which we can understand the IO is sequential or random.

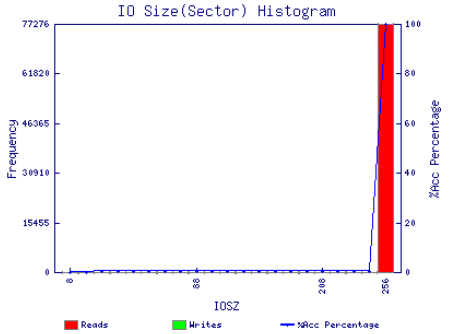
* blktrace is a kernel block layer IO tracing mechanism which provides detailed information about disk request queue operations up to user space.

RDD Read in Map: sequential

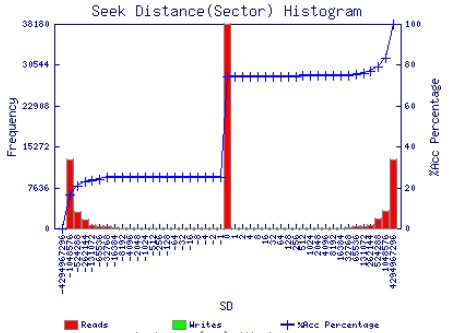
Red is Read
Green is Write

Sequential data distribution

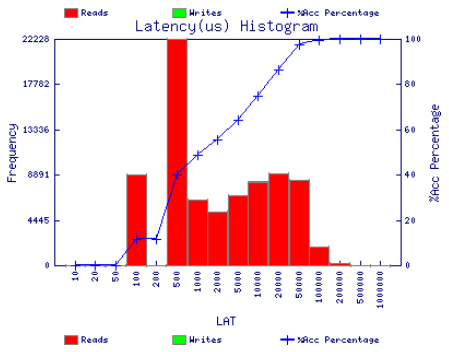
Classic hard disk seek time is 8-9ms, spindle rate is 7200rps, it means 1 sector random access needs 13ms at least.



Big IO size

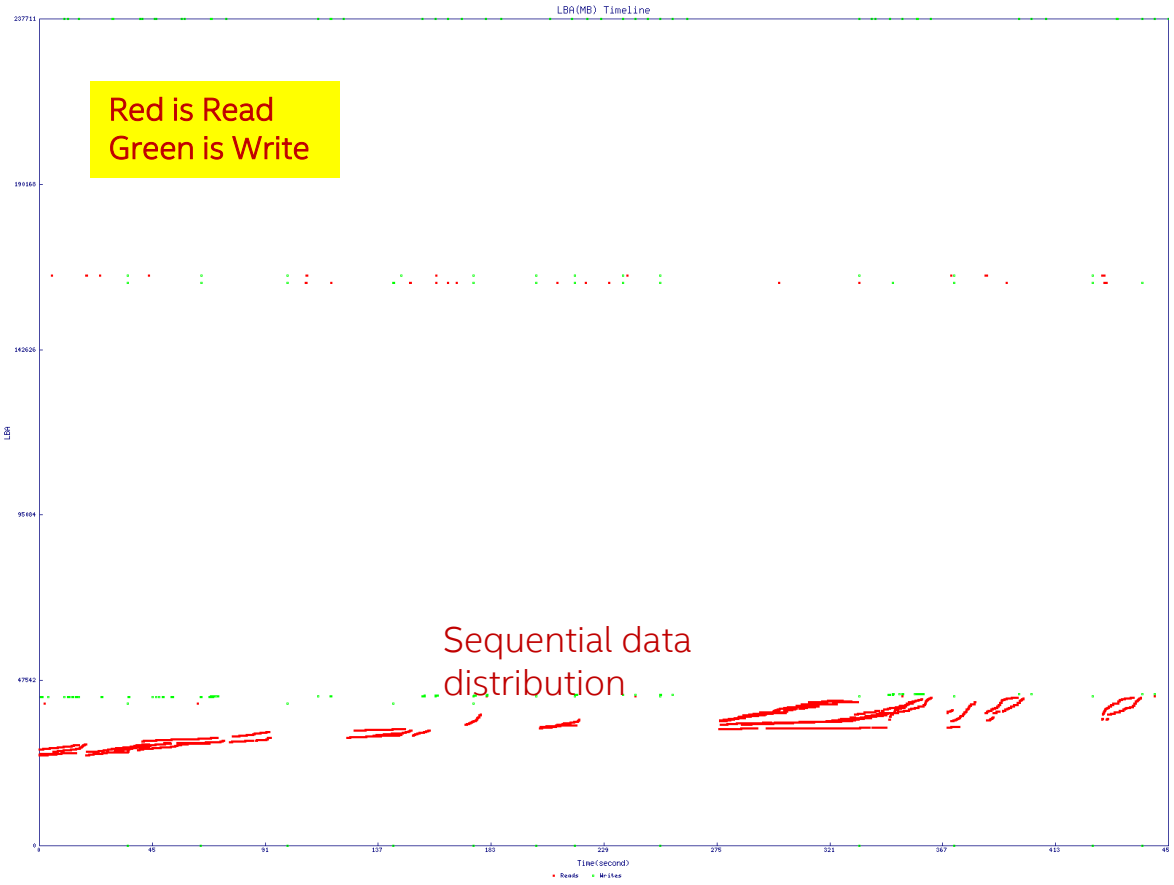


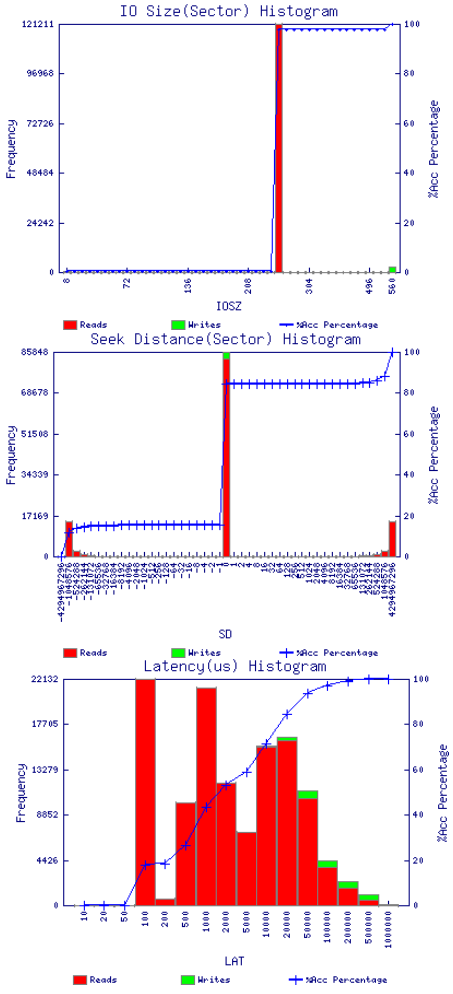
Much 0 SD



Low latency

RDD Read in Reduce: sequential



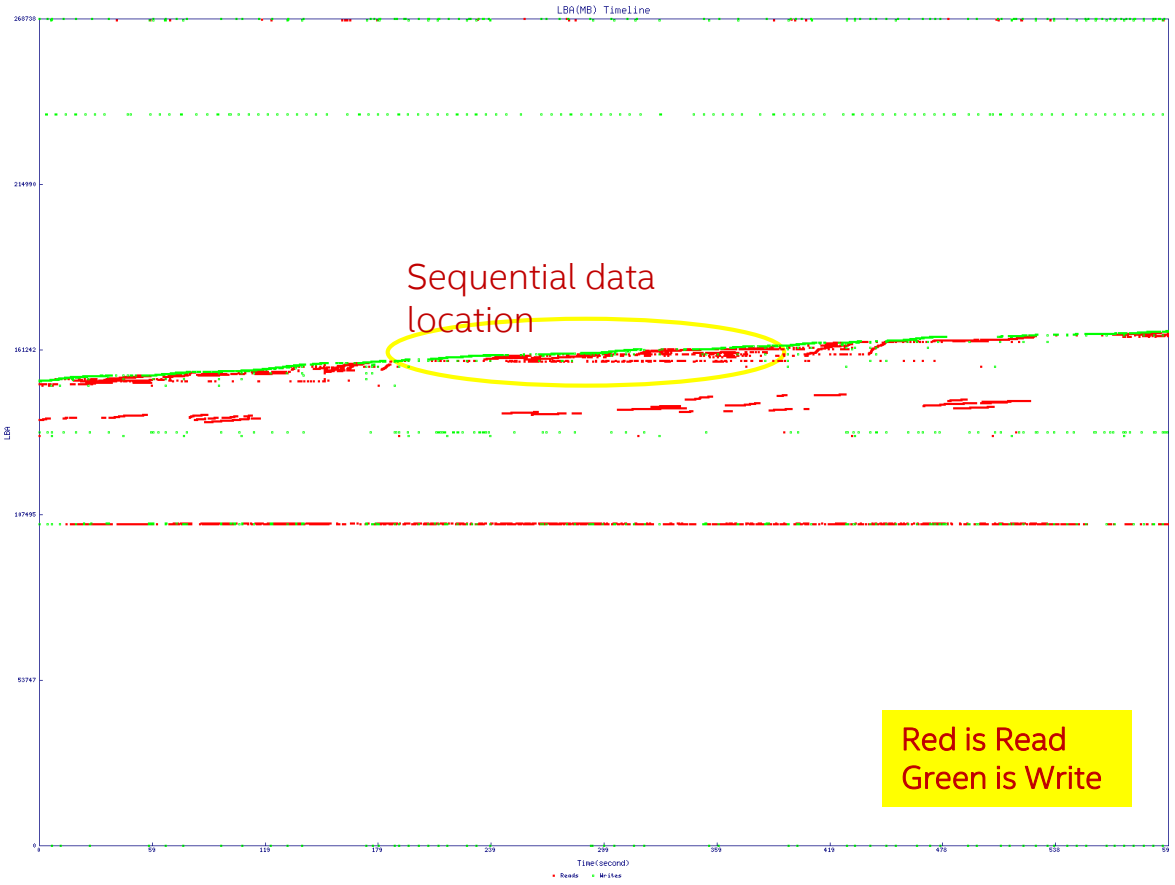


Big IO size

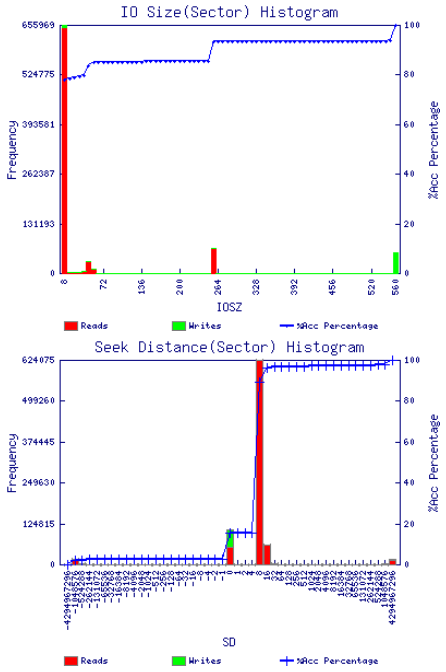
Much 0 SD

Low latency

RDD Write in Reduce: sequential write but with frequent 4K read

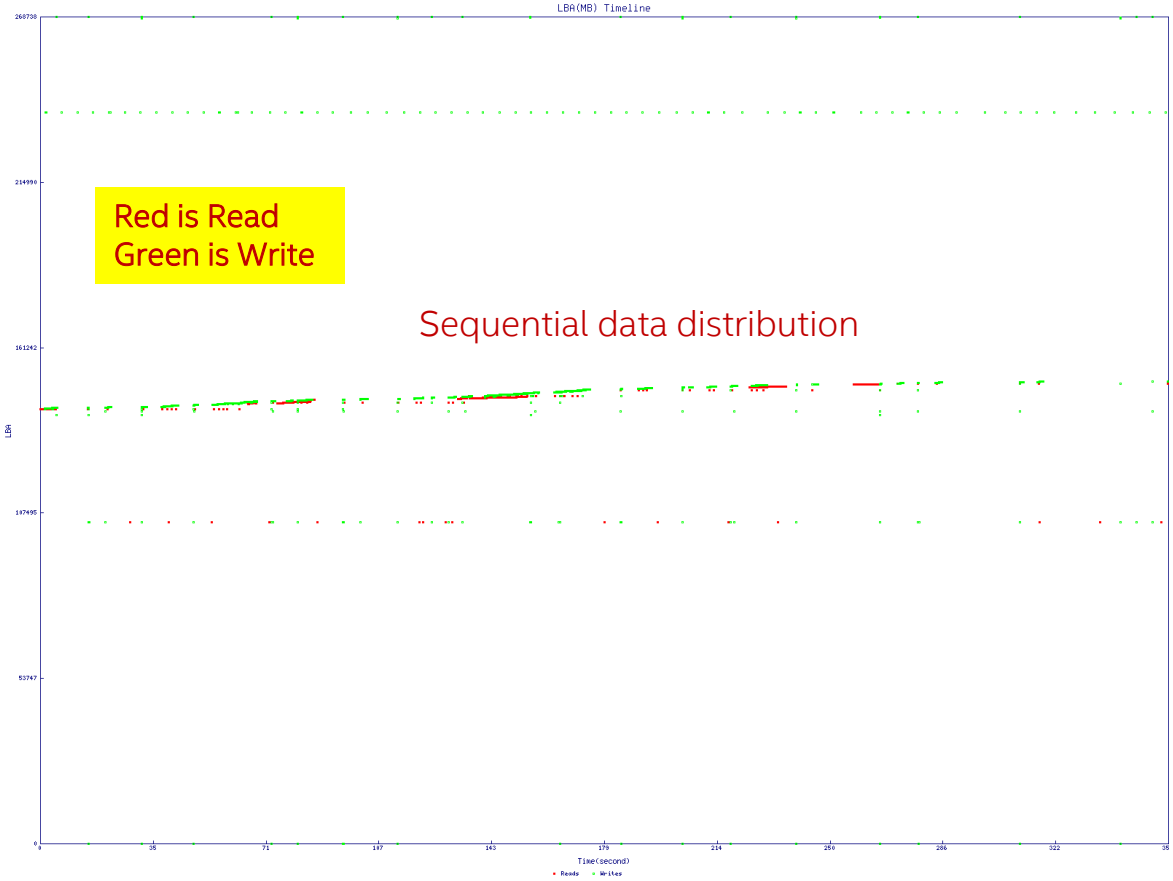


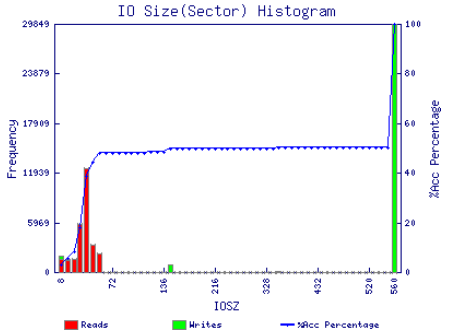
Those 4K read is probably because of spilling in cogroup, maybe a spark issue



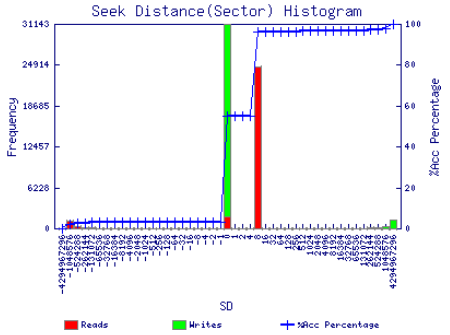
Write IO size is big but with many small 4K read IO

Shuffle Write in Map: sequential



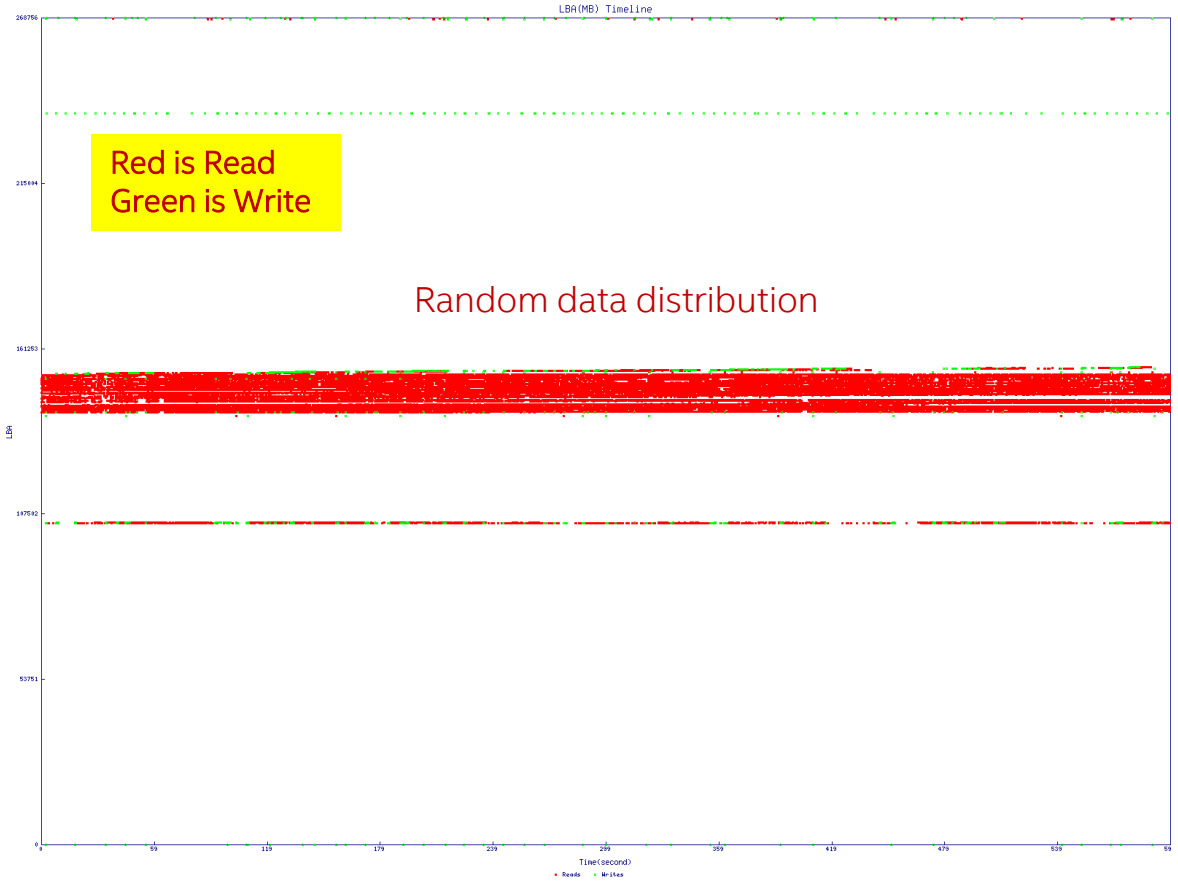


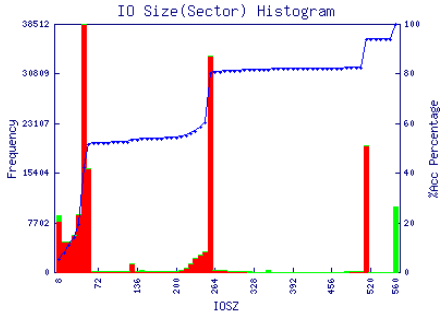
Big IO size



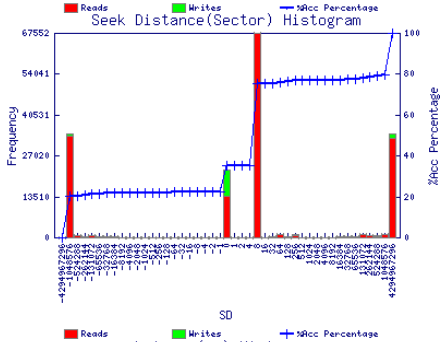
Much 0 SD

Shuffle Read in Reduce: random

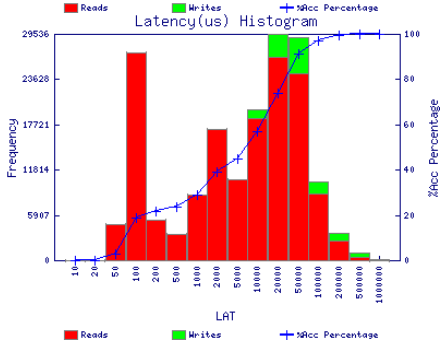




Small IO size



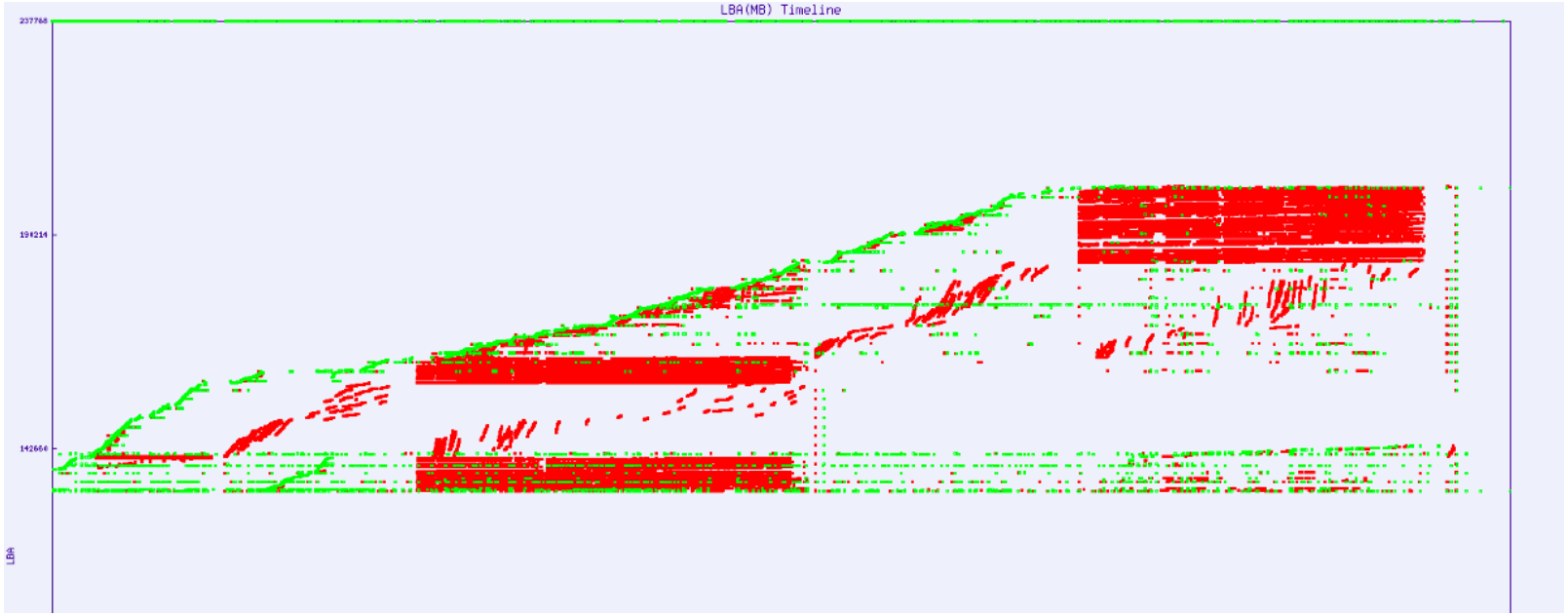
Few 0 SD



High latency

Conclusion

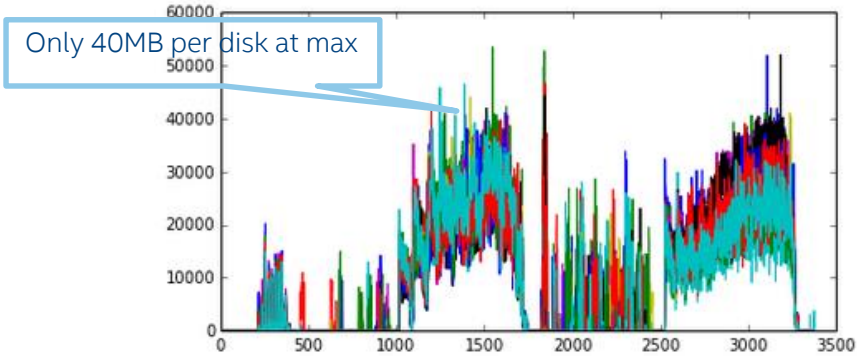
- RDD read/write, shuffle write are sequential.
- Shuffle read is very random.



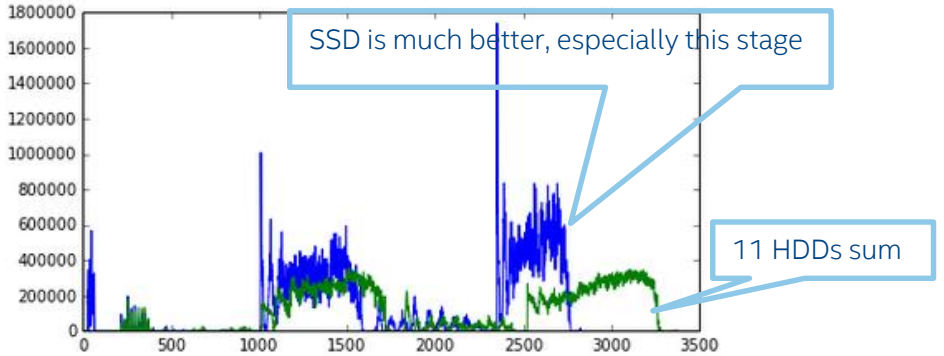
How bad when “shuffle read” with HDDs

x2 improvement for “shuffle read in reduce”
x3 improvement in real shuffle
x2 improvement in E2E testing

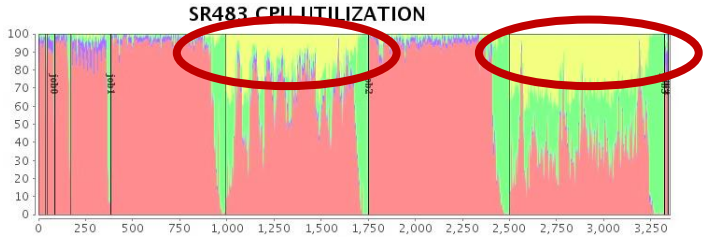
Per disk BW when shuffle read from 11 HDD



BW when shuffle read from 1 NVMe SSD

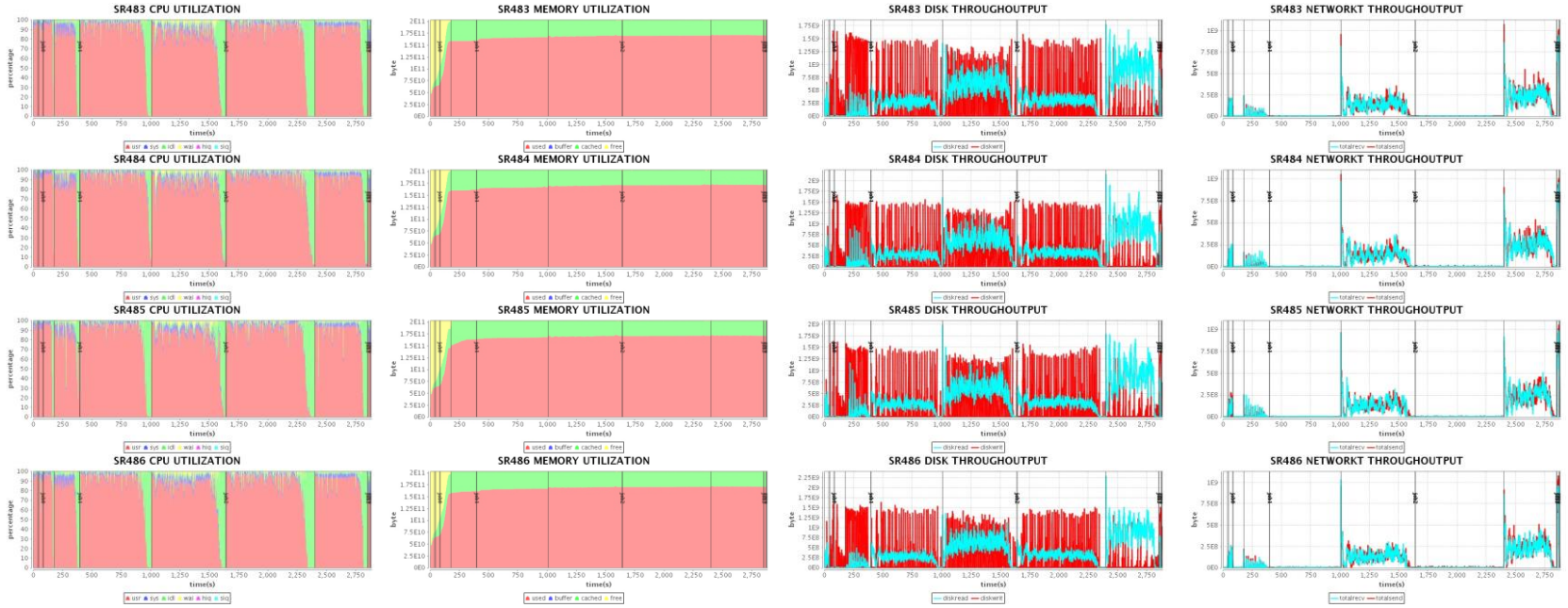


Shuffle read from HDD leads to High IO Wait



Description	Shuffle Read	Shuffle Write	SSD-RDD + HDD-Shuffle	1 SSD
saveAsTextFile at BagelNWeight.scala			20 s	20 s
foreach at Bagel.scala	490.3 GB		14 min	7.5 min
flatMap at Bagel.scala		490.4 GB	12 min	13 min
foreach at Bagel.scala	379.5 GB		13 min	11 min
flatMap at Bagel.scala		379.6 GB	10 min	10 min
foreach at Bagel.scala	19.1 GB		3.5 min	3.7 min
flatMap at Bagel.scala		19.1 GB	1.5 min	1.5 min
foreach at Bagel.scala	15.3 GB		38 s	39 s
parallelize at BagelNWeight.scala			38 s	38 s
flatMap at BagelNWeight.scala		15.3 GB	46 s	46 s

NVMe SSD HS kills IO bottleneck

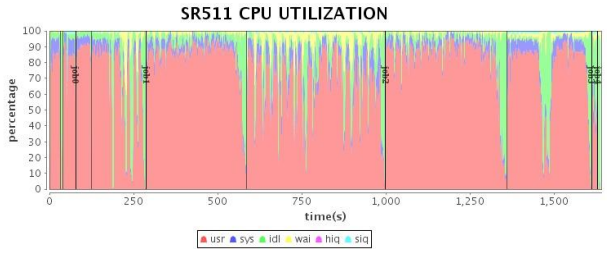


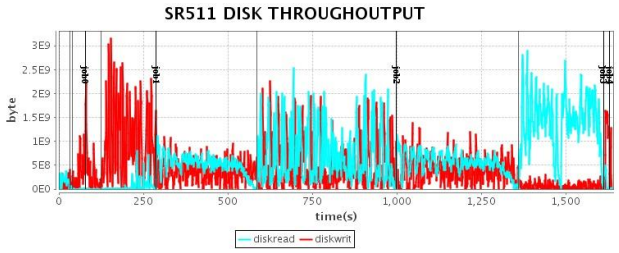
If CPU is not so bottleneck?

x3-5 improvement for shuffle
x3 improvement in E2E testing

NWeight

						11 HDDs	PCIe SSD	HSW
Stage Id	Description	Input	Output	Shuffle Read	Shuffle Write	Duration	Duration	Duration
23	saveAsTextFile at BagelNWeight.scala:102+details	50.1 GB	27.6 GB			27 s	20 s	26 s
17	foreach at Bagel.scala:256+details	732.0 GB		490.4 GB		*23 min*	*7.5 min*	*4.6 min*
16	flatMap at Bagel.scala:96+details	732.0 GB			490.4 GB	15 min	13 min	6.3 min
11	foreach at Bagel.scala:256+details	590.2 GB		379.5 GB		*25 min*	*11 min*	*7.1 min*
10	flatMap at Bagel.scala:96+details	590.2 GB			379.6 GB	12 min	10 min	5.3 min
6	foreach at Bagel.scala:256+details	56.1 GB		19.1 GB		4.9 min	3.7 min	2.8 min
5	flatMap at Bagel.scala:96+details	56.1 GB			19.1 GB	1.5 min	1.5 min	47 s
2	foreach at Bagel.scala:256+details			15.3 GB		38 s	39 s	36 s
1	parallelize at BagelNWeight.scala:97+details					38 s	38 s	35 s
0	flatMap at BagelNWeight.scala:72+details	22.6 GB			15.3 GB	46 s	46 s	43 s





Hierarchy store summary

- IO是Spark应用的一个常见瓶颈，而Shuffle Read是造成这个问题的首要原因。由于其与生俱来的随机性，单纯的加入机械硬盘往往事倍功半。
- 使用NVMe SSD做缓存，彻底解决Spark的IO瓶颈，带来了显著的性能提升。
- 由于只加入一块NVMe SSD，成本也得好很好的控制。


Lex
上海 浦东新区
扫一扫上面的二维码图案，加我微信

Geekbang 极客邦科技 InfoQ

THANKS!

Usage

1. Set the priority and threshold in spark-default.xml.

```
spark.storage.hierarchyStore     ssd 30GB
```

2. Configure “ssd” location: just put the keyword like "ssd“ in local dir. For example, in yarn-site.xml:.

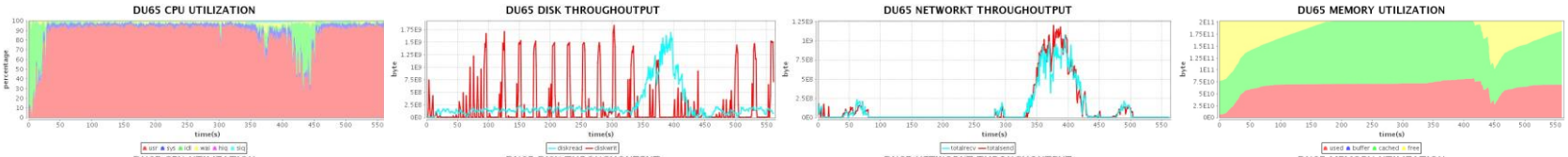
```
<property>
  <name>yarn.nodemanager.local-dirs</name>
  <value>file:///mnt/DP_disk1/yucai/yarn/local,file:///mnt/DP_disk2/yucai/yarn/local,
         file:///mnt/DP_disk3/yucai/yarn/local,file:///mnt/DP_disk4/yucai/yarn/local,
         file:///mnt/DP_disk5/yucai/yarn/local,file:///mnt/DP_disk6/yucai/yarn/local,
         file:///mnt/DP_disk7/yucai/yarn/local,file:///mnt/DP_disk8/yucai/yarn/local,
         file:///mnt/DP_disk9/yucai/yarn/local,file:///mnt/DP_disk10/yucai/yarn/local,
         file:///mnt/DP_disk11/yucai/yarn/local, file:///mnt/nvme0n1/yucai/yarn.local.ssd,
  </value>
</property>
```

SSD v.s. Memory: similar performance, lower price

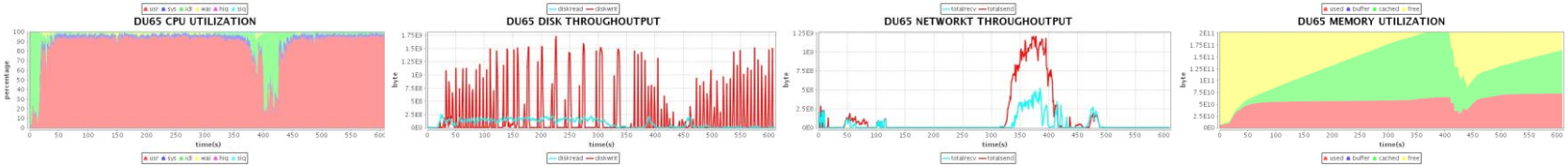
similar end to end performance (<5%)

Same network BW，
similar shuffle time

128GB Mem



192GB Mem



During shuffle, most in memory already, few disk access.

SUT #A

		IVB
Master	CPU	Intel(R) Xeon(R) CPU E5-2680 @ 2.70GHz (16 cores)
	Memory	64G
	Disk	2 SSD
	Network	1 Gigabit Ethernet
Slaves	Nodes	4
	CPU	Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz (2 CPUs, 10 cores, 40 threads)
	Memory	192G DDR3 1600MHz
	Disk	11 HDDs/11 SSDs/1 PCI-E SSD(P3600)
	Network	10 Gigabit Ethernet

	IVB E5-2680
OS	Red Hat 6.2
Kernel	3.16.7-upstream
Spark	Spark 1.4.1
Hadoop/HDFS	Hadoop-2.5.0-cdh5.3.2
JDK	Sun hotspot JDK 1.8.0 (64bits)
Scala	scala-2.10.4

SUT #B

		HSW
Master	CPU	Intel(R) Xeon(R) CPU X5570 @ 2.93GHz (16 cores)
	Memory	48G
	Disk	2 SSD
	Network	1 Gigabit Ethernet
Slaves	Nodes	4
	CPU	Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz (2 CPUs, 18 cores, 72 threads)
	Memory	256G DDR4 2133MHz
	Disk	11 SSD
	Network	10 Gigabit Ethernet

	HSW E5-2699
OS	Ubuntu 14.04.2 LTS
Kernel	3.16.0-30-generic.x86_64
Spark	Spark 1.4.1
Hadoop/HDFS	Hadoop-2.5.0-cdh5.3.2
JDK	Sun hotspot JDK 1.8.0 (64bits)
Scala	scala-2.10.4

SUT #C

Hardware Specification		
Slaves	Nodes	4
	CPU	Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz (40 cores)
	Memory	192GB/128GB DDR4 2133MHz
	Disk	7 HDDs + 1 PCIE-SSD (P3700)
	Network	10 Gigabit Ethernet
Master	CPU	Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz (32 cores)
	Memory	64G
	Disk	2 SSD
	Network	10 Gigabit Ethernet

HasWell E5-2650 Software Specification	
OS	CentOS release 6.3 (Final)
Kernel	Baidu 2.6.32_1-18-0-0
Spark	Spark 1.6.0 with SSD hierarch support
Hadoop/HDFS	Hadoop-2.5.0-cdh5.3.2
JDK	Sun hotspot JDK 1.8.0 (64bits)
Scala	scala-2.10.4

Analysis SQL from Baidu NuoMi

Test Configuration

executors number: 32

executor memory: 18G

executor-cores: 5

spark-defaults.conf:

spark.serializer	org.apache.spark.serializer.KryoSerializer
spark.kryo.referenceTracking	false

Test Configuration

executors number: 32

executor memory: 10-14G

executor-cores: 5

spark-defaults.conf:

spark.yarn.executor.memoryOverhead	4096
spark.sql.shuffle.partitions	320
spark.storage.hierarchyStore	ssd 20GB

HDD (Seagate ST9500620NS) SPEC

General

Device Type	Hard drive - internal
Capacity	500 GB
Form Factor	2.5"
Interface	Serial ATA-600
Buffer Size	64 MB
Bytes per Sector	512
Features	Native Command Queuing (NCQ), Perpendicular Magnetic Recording (PMR), PowerChoice , S.M.A.R.T.
Width	2.8 in
Depth	4 in
Height	0.6 in
Weight	6.5 oz
First Seen On Google Shopping	December 2010

HDD (Seagate ST9500620NS) SPEC

Performance	
Drive Transfer Rate	600 MBps (external)
Seek Time	8.5 ms (average)
Track-to-Track Seek Time	0.7 ms
Average Latency	4.16 ms
Spindle Speed	7200 rpm

Reliability	
MTBF	1,400,000 hours
Non-Recoverable Errors	1 per 10^15

PCIe SSD(P3600) SPEC

Table 6: Sequential Read and Write Bandwidth

Specification	Unit	Intel SSD DC P3600 Series				
		400GB	800GB	1.2TB	1.6TB	2TB
Sequential Read (up to)[1]	MB/s	2,100	2,600	2,600	2,600	2,600
Sequential Write (up to)[1]	MB/s	550	1000	1250	1600	1700

NOTE: Performance measured using IOMeter* with 128 KB (131,072 bytes) of transfer size with Queue Depth 128. Power mode set at 25W.

PCIe SSD(P3600) SPEC

Table 4: Random Read/Write Input/Output Operations Per Second (IOPS)

Specification[1]	Unit	Intel SSD DC P3600 Series				
		400GB	800GB	1.2TB	1.6TB	2TB
Random 4KB 70/30 Read/Write (up to)[2]	IOPS	80,000	110,000	130,000	160,000	160,000
Random 8KB 70/30 Read/Write (up to)[2]	IOPS	45,000	55,000	65,000	75,000	80,000
Random 4KB Read (up to)	IOPS	320,000	430,000	450,000	450,000	450,000
Random 4KB Write (up to)	IOPS	30,000	50,000	50,000	56,000	56,000
Random 8KB Read (up to)	IOPS	180,000	250,000	260,000	270,000	275,000
Random 8KB Write (up to)	IOPS	19,000	26,000	27,000	33,000	33,000

NOTES:

1. Performance measured using IOMeter* on Intel provided Windows Server 2012 R2 driver with Queue Depth 32 and number of workers equal to 4. Measurements are performed on a full Logical Block Address (LBA) span of the drive. Power mode set at 25W.
2. 4KB = 4,096 bytes
3. 8KB = 8,192 bytes

PCIE-SSD(P3700) SPEC

- Performance[1,2]
  - Seq R/W: Up to 2800/2000MB/s[3]
  - IOPS Rnd 4KB[4] 70/30 R/W: Up to 265K
  - IOPS Rnd 4KB[4] R/W: Up to 460/175K
  - Seq Latency (typ) R/W: 20/20µs

Table 6: Sequential Read and Write Bandwidth

Specification	Unit	Intel SSD DC P3700 Series			
		400GB	800GB	1.6TB	2TB
Sequential Read (up to)[1]	MB/s	2,700	2,800	2,800	2,800
Sequential Write (up to)[1]	MB/s	1,080	1,900	1,900	2,000

PCIE-SSD(P3700) SPEC

Table 4: Random Read/Write Input/Output Operations Per Second (IOPS)

Specification[1]	Unit	Intel SSD DC P3700 Series			
		400GB	800GB	1.6TB	2TB
Random 4KB 70/30 Read/Write (up to)[2]	IOPS	150,000	200,000	240,000	265,000
Random 8KB 70/30 Read/Write (up to)[3]	IOPS	75,000	100,000	140,000	150,000
Random 4KB Read (up to)[2]	IOPS	450,000	460,000	450,000	450,000
Random 4KB Write (up to)	IOPS	75,000	90,000	150,000	175,000
Random 8KB Read (up to)[3]	IOPS	275,000	285,000	290,000	295,000
Random 8KB Write (up to)	IOPS	32,000	45,000	75,000	90,000


intel
Software