Practical Caching

Jean-Daniel Cryans
DB Engineer, StumbleUpon
@jdcryans
January 19th, 2012, eBay

Overview

- Setup
- Serving from memory
- Trashing the cache
- Recommendations

Objectives

- Show how different configurations and datasets can affect random read performance in HBase 0.92 and Hadoop 1.0 (and variants).
- Demonstrate how to do those performance tests in a way that can be translated to your own use case.

Setup: Testbed

- 1 master, 14 slaves
  - Intel Xeon E5520
  - 24GB
  - 4x1TB, 7.2k RPM
  - “Old prod machines”
- Software
  - CDH3u3’s Hadoop, last snapshot from 2011
  - Tip of HBase’s 0.92 branch + modified HBASE-4440

Setup: Basic Configurations

- *mapred.tasktracker.map.tasks.maximum*=35
- *dfs.block.local-path-access.user*=hadoop

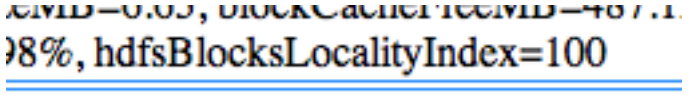
- *hbase.regionserver.handler.count*=100
- *hbase.client.scanner.caching*=100
- *hbase.online.schema.update.enable*=true

- */proc/sys/net/ipv4/tcp_tw_recycle*=1
  - see HBASE-2492

SERVING FROM MEMORY

Data Import

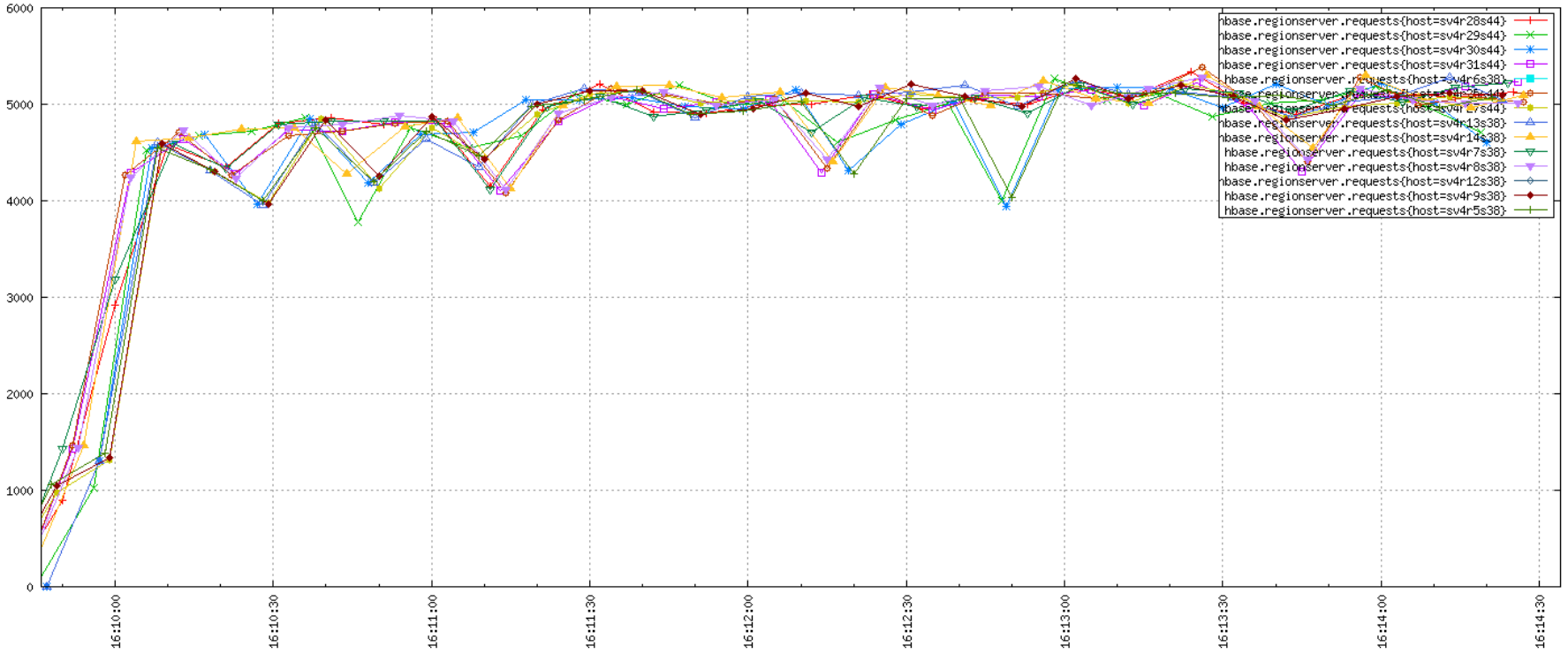
- "*PerformanceEvaluation --presplit=70 sequentialWrite 50*"
- Force flushed then major compacted to get 100% data locality



- **51.28GB** as reported by hadoop's *dus*, 52,428,800 rows
- "*PerformanceEvaluation scan 50*" in order to load all the OS caches.

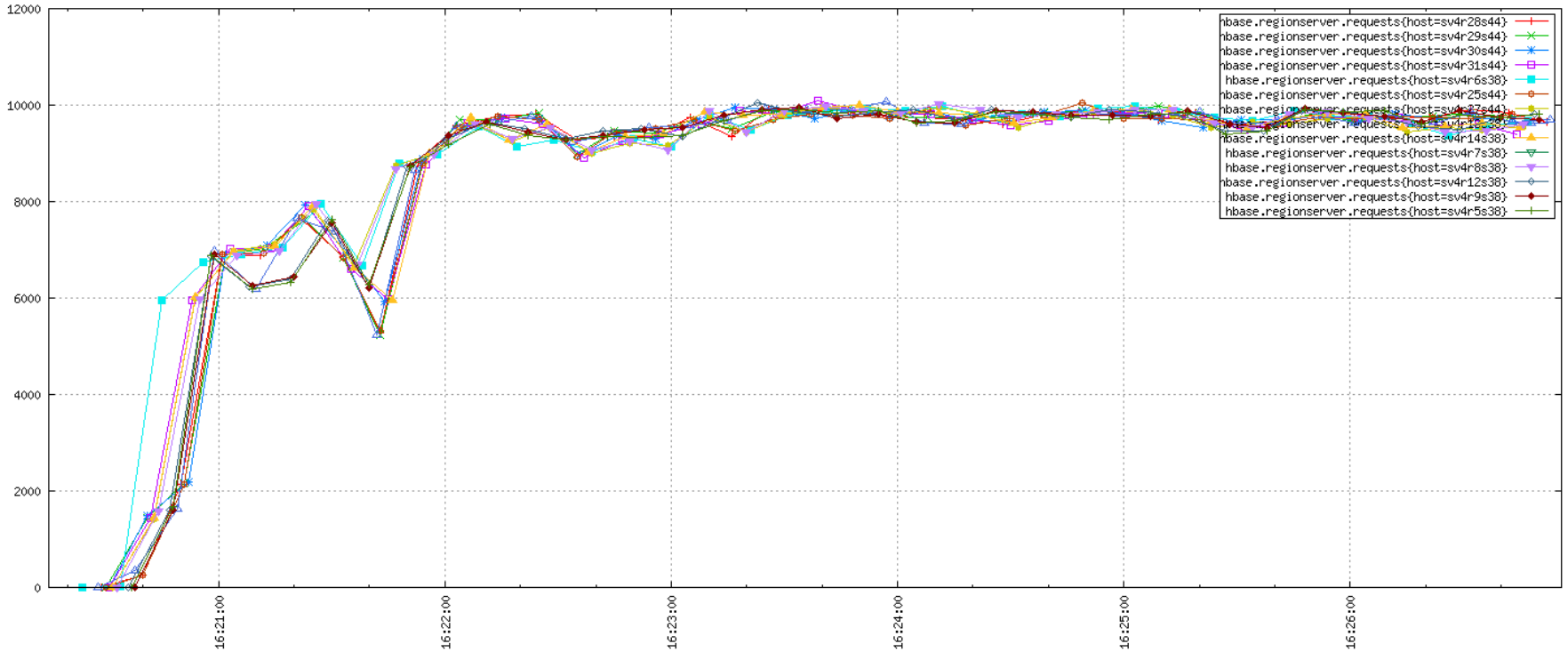
Default Heap, no Block Cache

- blockCacheSizeMB=4.61 on average
- Peaks at **70k/sec**
- *BLOCKCACHE* => '**false**'



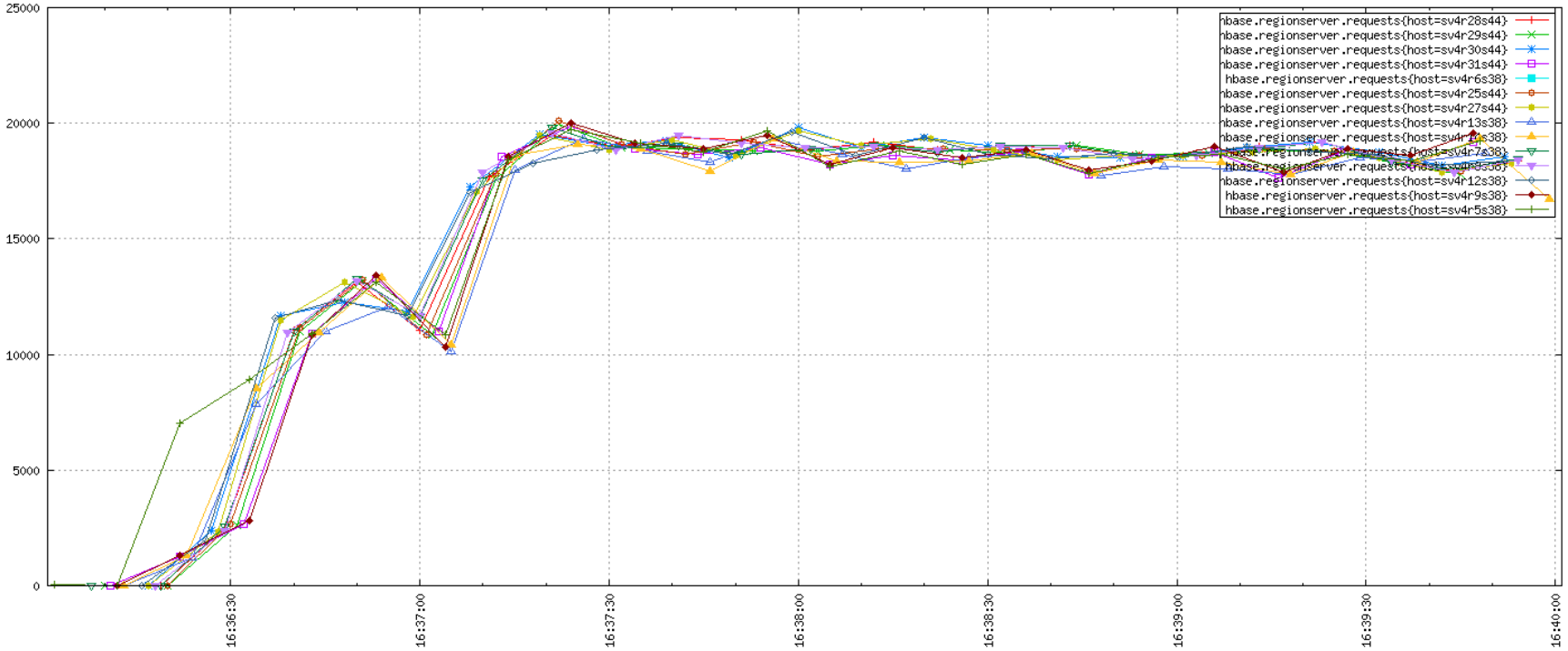
Read Short Circuit

- *dfs.client.read.shortcircuit*=**true** in hbase-site
- Peaks at **140k/sec**



100% Block Cached

- Heap bumped to **8GB**
- *hfile.block.cache.size*=**0.60**
- Peaks at **265k/sec**



Different Block Size

- *BLOCKSIZE* => '4096’
- 4KB
  - Default: Peaks at **70k/sec** (Stable)
  - Short circuit: Peaks at **210k/sec** (50% faster)
  - 100% BC: Peaks at **290k/sec** (10% faster)
  - The difference is less checksumming (which is done every 512 bytes).

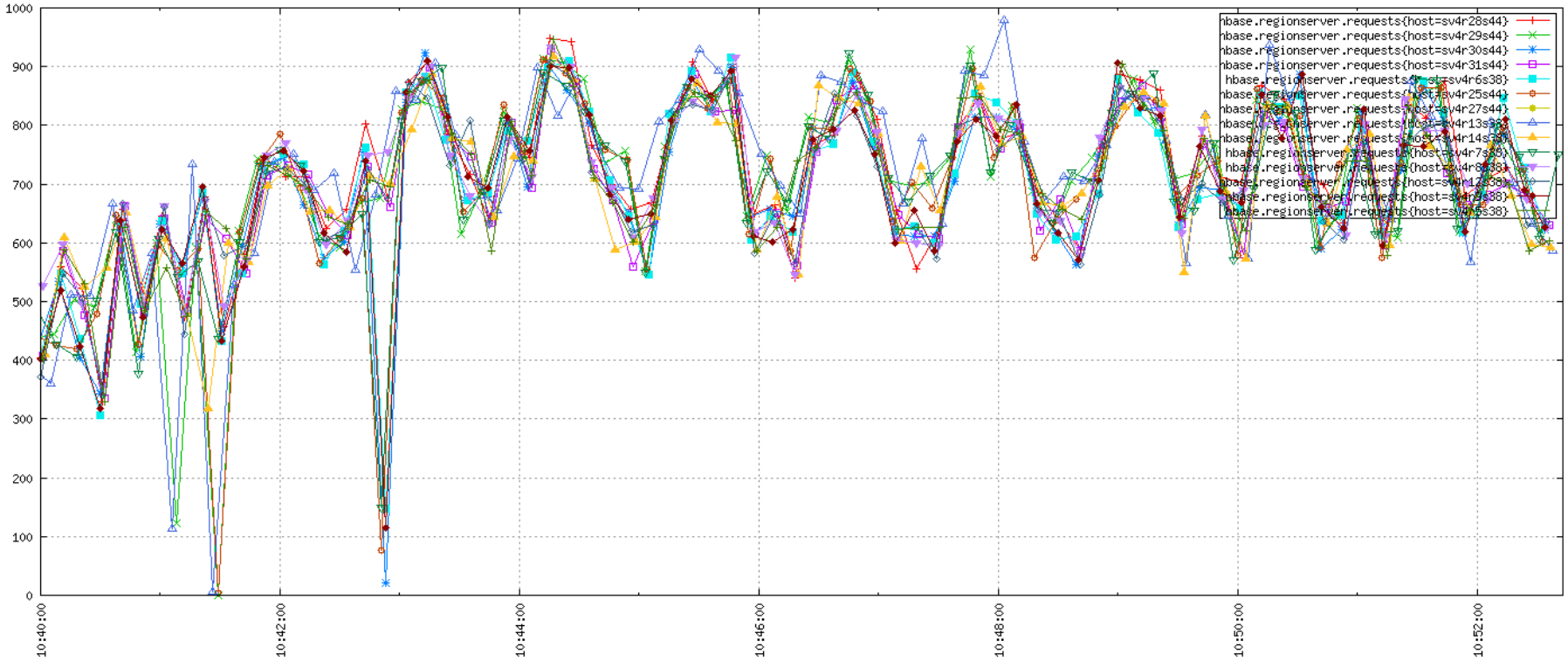
TRASHING THE CACHE

Data Import

- "*PerformanceEvaluation --presplit=70 sequentialWrite 300*"
- Force flushed then major compacted to get 100% data locality
- **312GB** as reported by hadoop's *dus*, 314,572,800 rows

Default Heap, no Block Cache

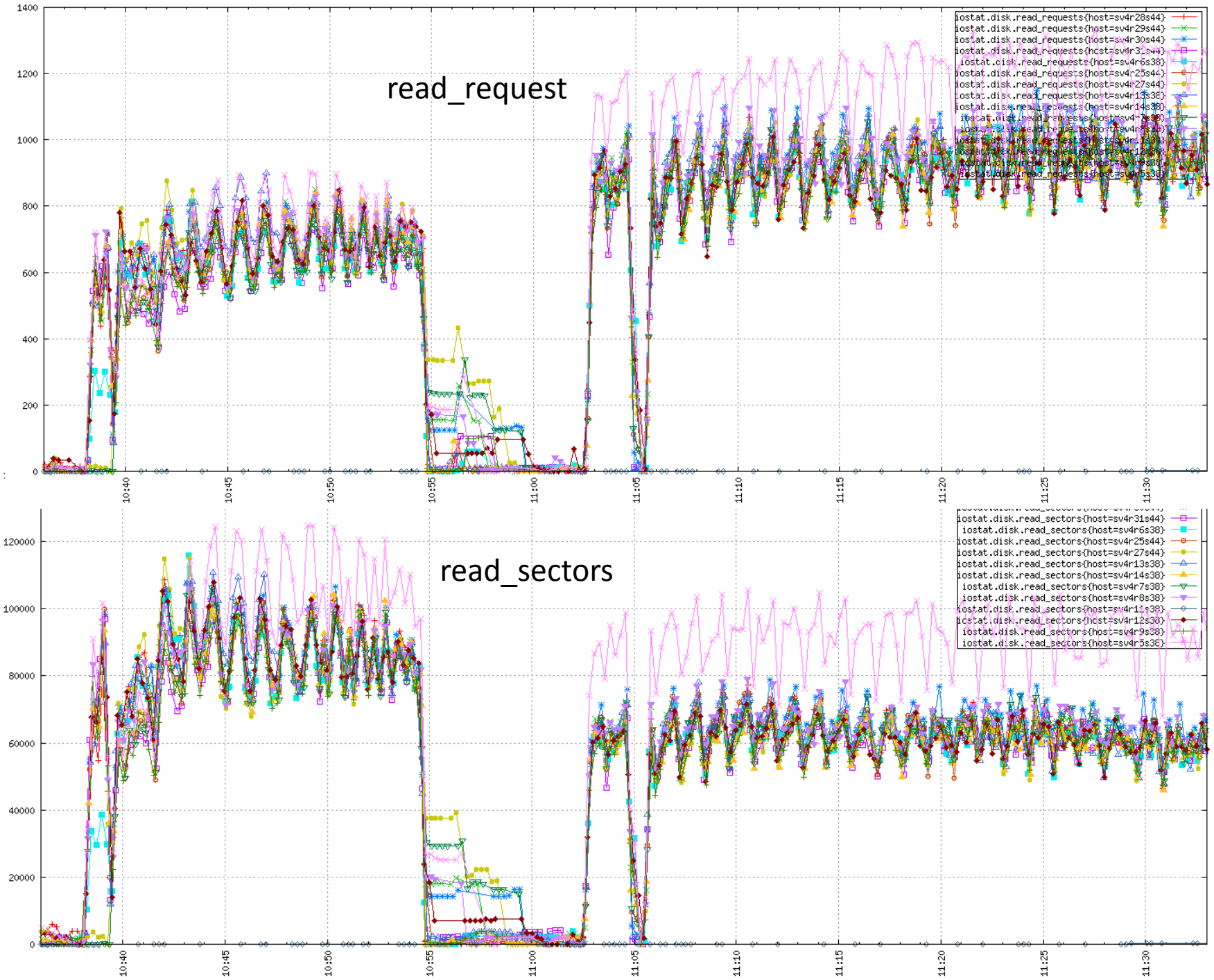
- blockCacheSizeMB=20MB on average
- Peaks at **13k/sec**



Read Short Circuit

- *dfs.client.read.shortcircuit*=**true**
- Peaks at **6k/sec**

What ???

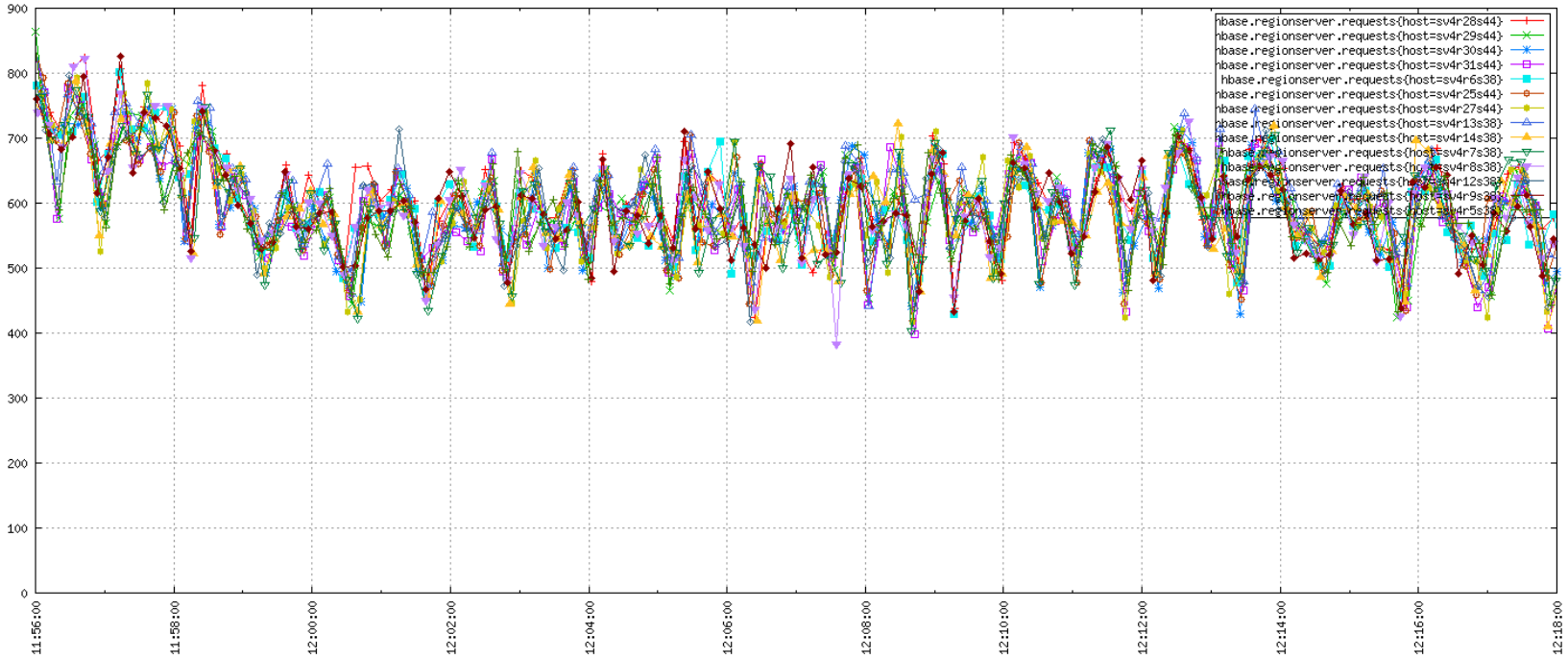

read_request
1400
1200
1000
800
600
400
200
0
10:40
10:45
10:50
10:55
11:00
11:05
11:10
11:15
11:20
11:25
11:30
iostat.disk.read_requests{host=sv4r28s44}
iostat.disk.read_requests{host=sv4r29s44}
iostat.disk.read_requests{host=sv4r30s44}
iostat.disk.read_requests{host=sv4r31s44}
iostat.disk.read_requests{host=sv4r6s38}
iostat.disk.read_requests{host=sv4r25s44}
iostat.disk.read_requests{host=sv4r27s44}
iostat.disk.read_requests{host=sv4r13s38}
iostat.disk.read_requests{host=sv4r14s38}
iostat.disk.read_requests{host=sv4r7s38}
read_sectors
120000
100000
80000
60000
40000
20000
0
10:40
10:45
10:50
10:55
11:00
11:05
11:10
11:15
11:20
11:25
11:30
iostat.disk.read_sectors{host=sv4r31s44}
iostat.disk.read_sectors{host=sv4r6s38}
iostat.disk.read_sectors{host=sv4r25s44}
iostat.disk.read_sectors{host=sv4r27s44}
iostat.disk.read_sectors{host=sv4r13s38}
iostat.disk.read_sectors{host=sv4r14s38}
iostat.disk.read_sectors{host=sv4r7s38}
iostat.disk.read_sectors{host=sv4r8s38}
iostat.disk.read_sectors{host=sv4r11s38}
iostat.disk.read_sectors{host=sv4r12s38}
iostat.disk.read_sectors{host=sv4r9s38}
iostat.disk.read_sectors{host=sv4r5s38}

Read Short Circuit

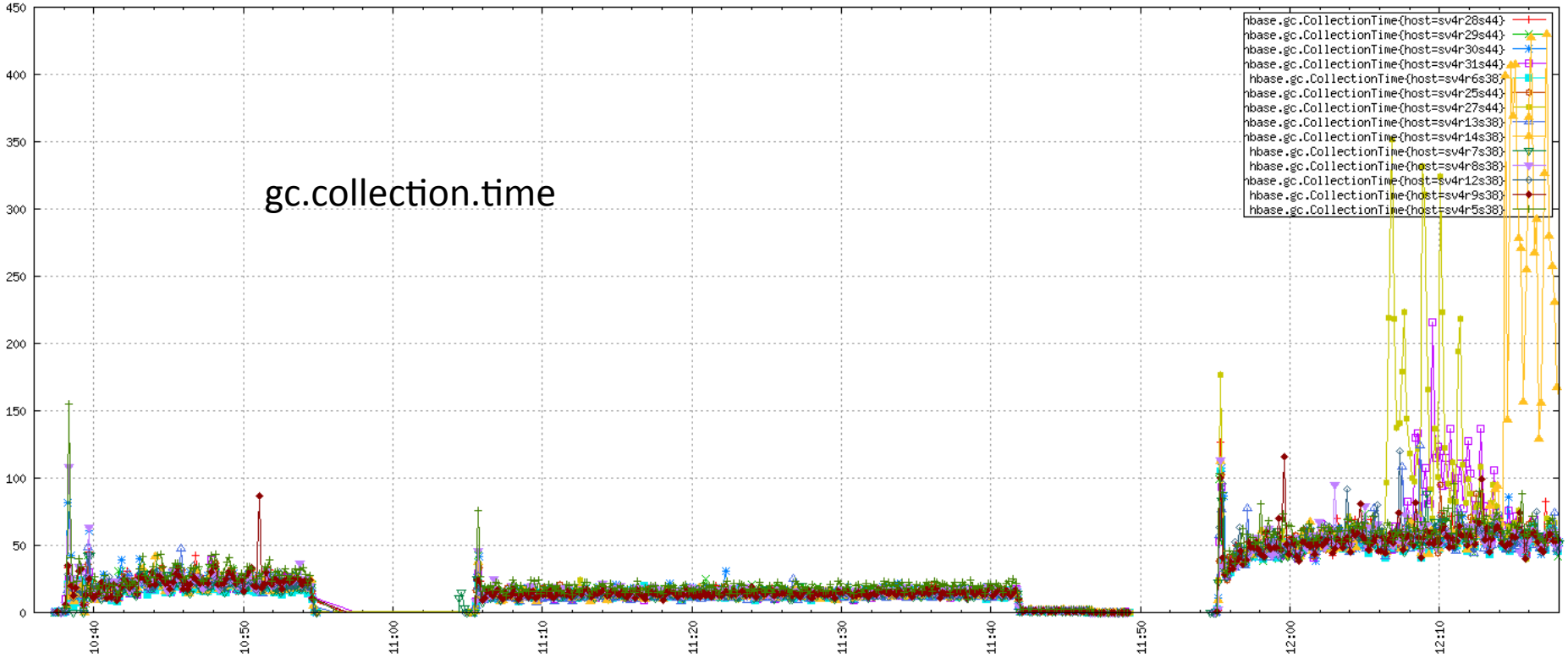
- More requests, but
- Smaller ones, so
- Throughput is destroyed
- Probably a performance bug, to be continued…

Block Caching

- Heap bumped to **13GB**
- *hfile.block.cache.size*=**0.70**
- Peaks early at **11k/sec**, “stable” at **9k/sec**



GC Comparison



default, no block caching

read short circuit

block caching

Recommendations

- It’s important to understand how much data you need to read, and how it fits into HBase.
- Read short circuit should always be enabled, unless you’re constantly trashing the cache.
- A smaller block size uses more memory but it’s faster when relying on the OS cache.

Looking Forward

- "*Off Heap Caching*" introduced in **HBASE-4027** gives a good alternative to using the OS cache while not suffering from GC.
- There's a lot more testing to do for reads, like enabling compression, writing concurrently, using bloom filters, etc