HBase Practice At Xiaomi

huzheng@xiaomi.com

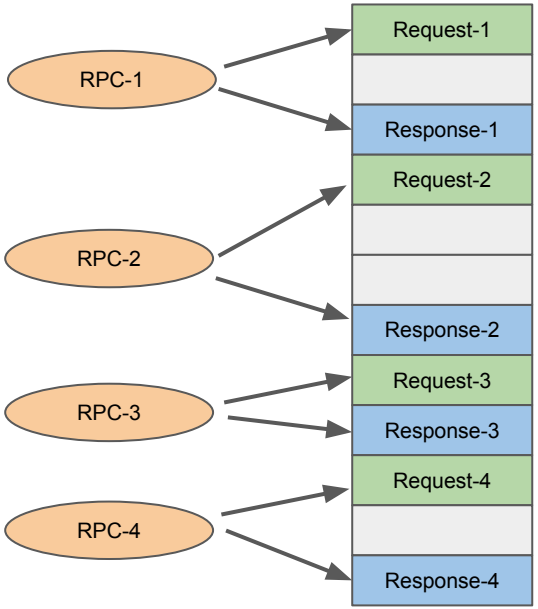
About This Talk

- Async HBase Client
  - Why Async HBase Client
  - Implementation
  - Performance
- How do we tuning G1GC for HBase
  - CMS vs G1
  - Tuning G1GC
  - G1GC in XiaoMi HBase Cluster

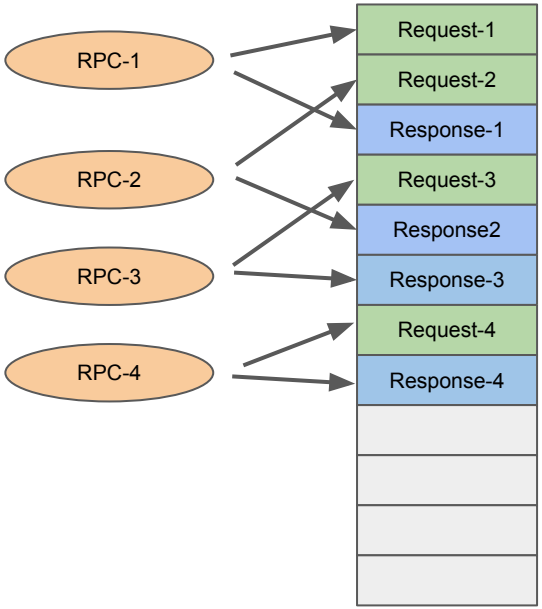
Part-1 Async HBase Client

Why Async HBase Client ?

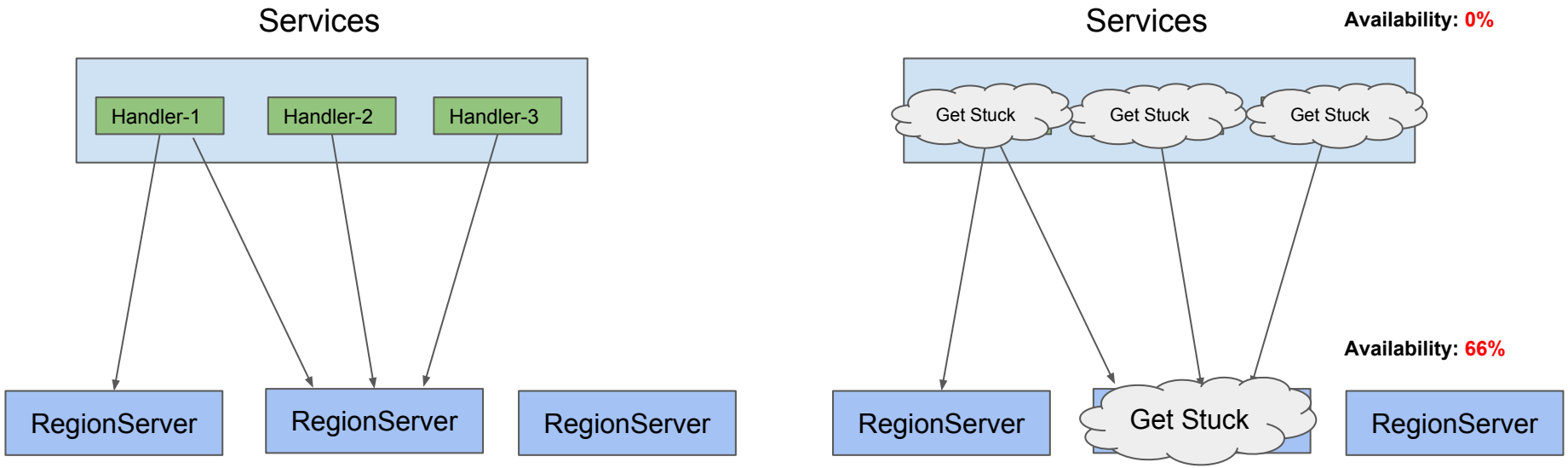
Blocking Client (Single Thread)



Non-Blocking Client(Single Thread)



Fault Amplification When Using Blocking Client



All of handlers are blocked if a region server blocked when using blocking client

Why Async HBase Client ?

- Region Server / Master STW GC
- Slow RPC to HDFS
- Region Server Crash
- High Load
- Network Failure

BTW: HBase may also suffer from fault amplification when accessing HDFS, so AsyncDFSClient ?

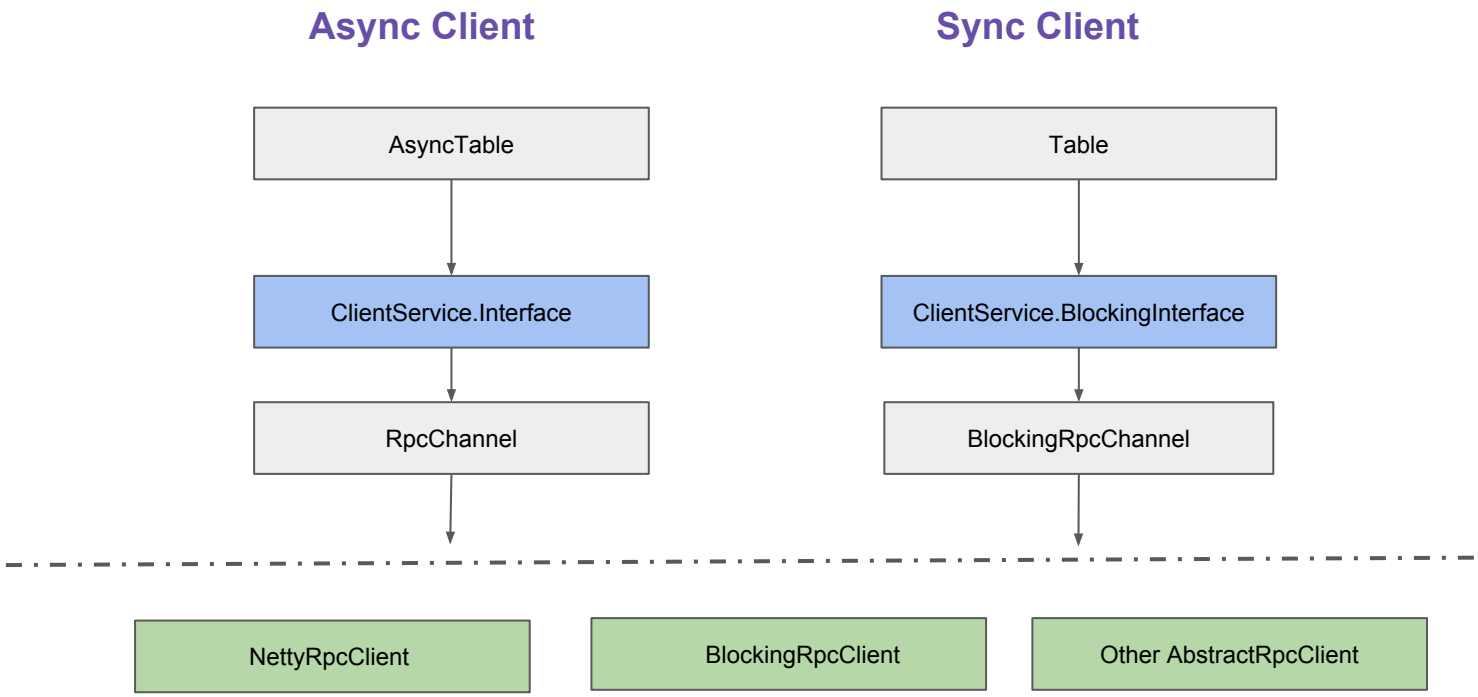
Async HBase Client *VS* OpenTSDB/asynchbase

	Async HBase Client	OpenTSDB/asynchbase(1.8)
HBase Version	>=2.0.0	Both 0.9x and 1.x.x
Table API	Every API In Blocking API	Parts of Blocking API
HBase Admin	Supported	Not Supported
Implementation	Included In HBase Project	Independent Project Based On PB protocol
Coprocessor	Supported	Not Supported

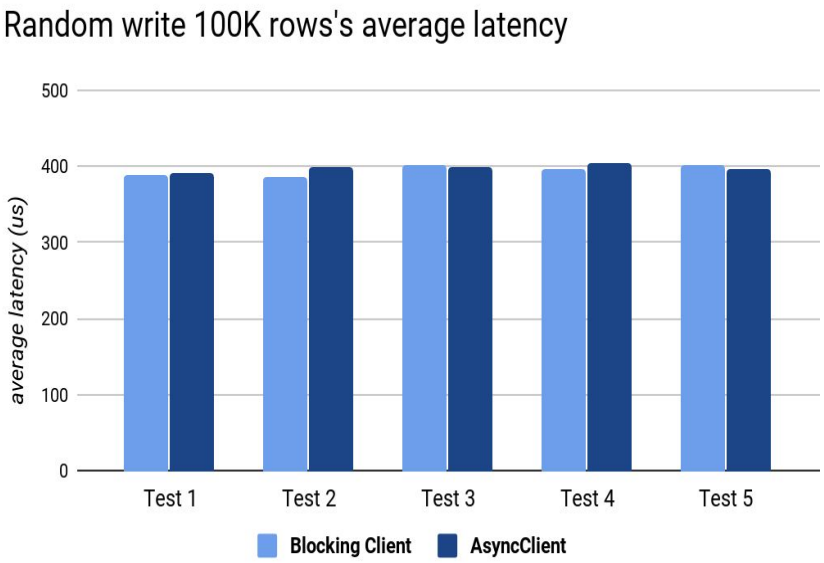
Async HBase Client Example

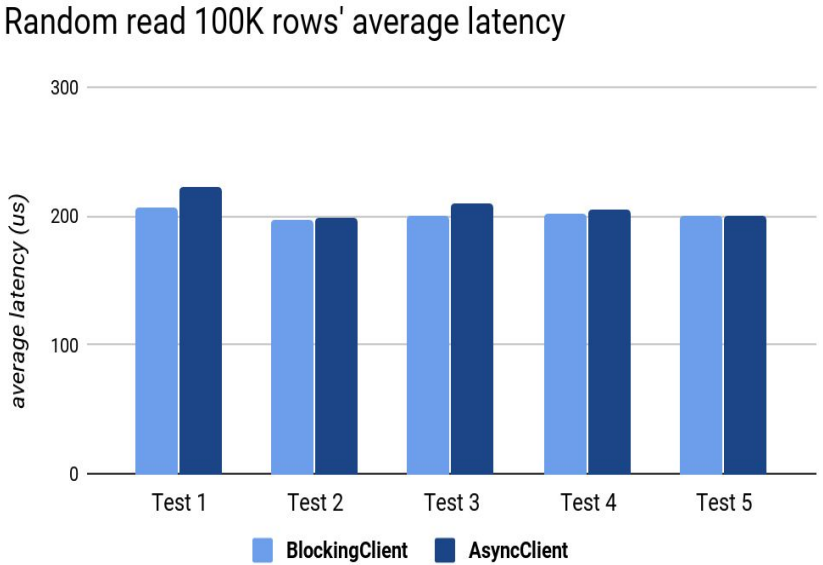
```
CompletableFuture<Result> asyncResult = new CompletableFuture<>();
ConnectionFactory.createAsyncConnection(conf)
  .whenComplete((asyncConn, error) -> {
        if (error != null) {
          asyncResult.completeExceptionally(error);
          return;
        }
        AsyncTable table = asyncConn.getTable(TABLE_NAME, pool);
        table.get(new Get(ROW_KEY)).whenComplete((result, throwable) -> {
          if (throwable != null) {
            asyncResult.completeExceptionally(throwable);
            return;
          }
          asyncResult.complete(result);
        });
      });
```

Async HBase Client Architecture



Performance Test



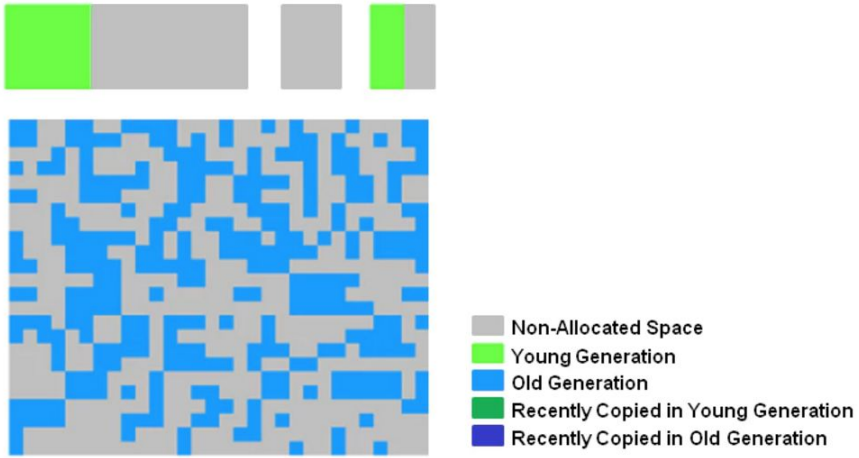


- Test Async RPC by CompletableFuture.get() (Based on XiaoMi HBase0.98)
- Proof that latency of async client is at least the same as blocking hbase client.

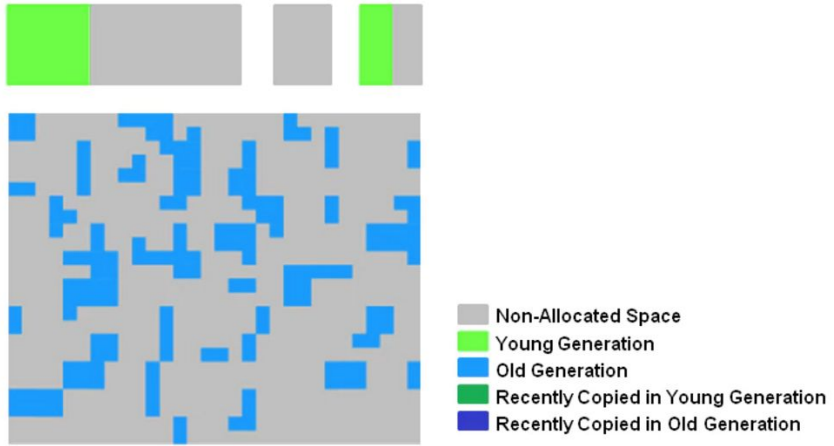
Part-2 HBase + G1GC Tuning

CMS *VS* G1

OldGen GC (Before)



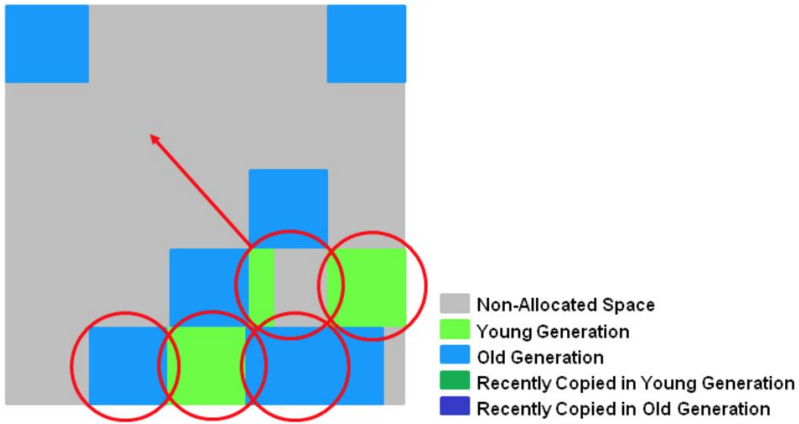
OldGen GC (After)



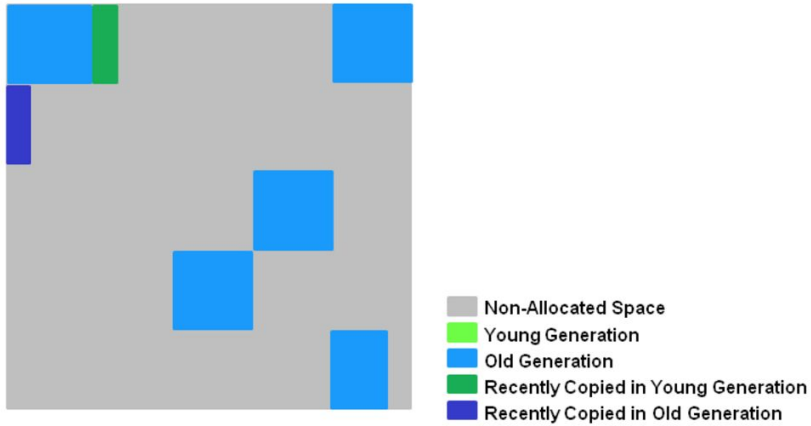
CMS Old Gen GC

CMS *VS* G1

Mixed GC (Before)



Mixed GC (After)



G1 Old Gen GC

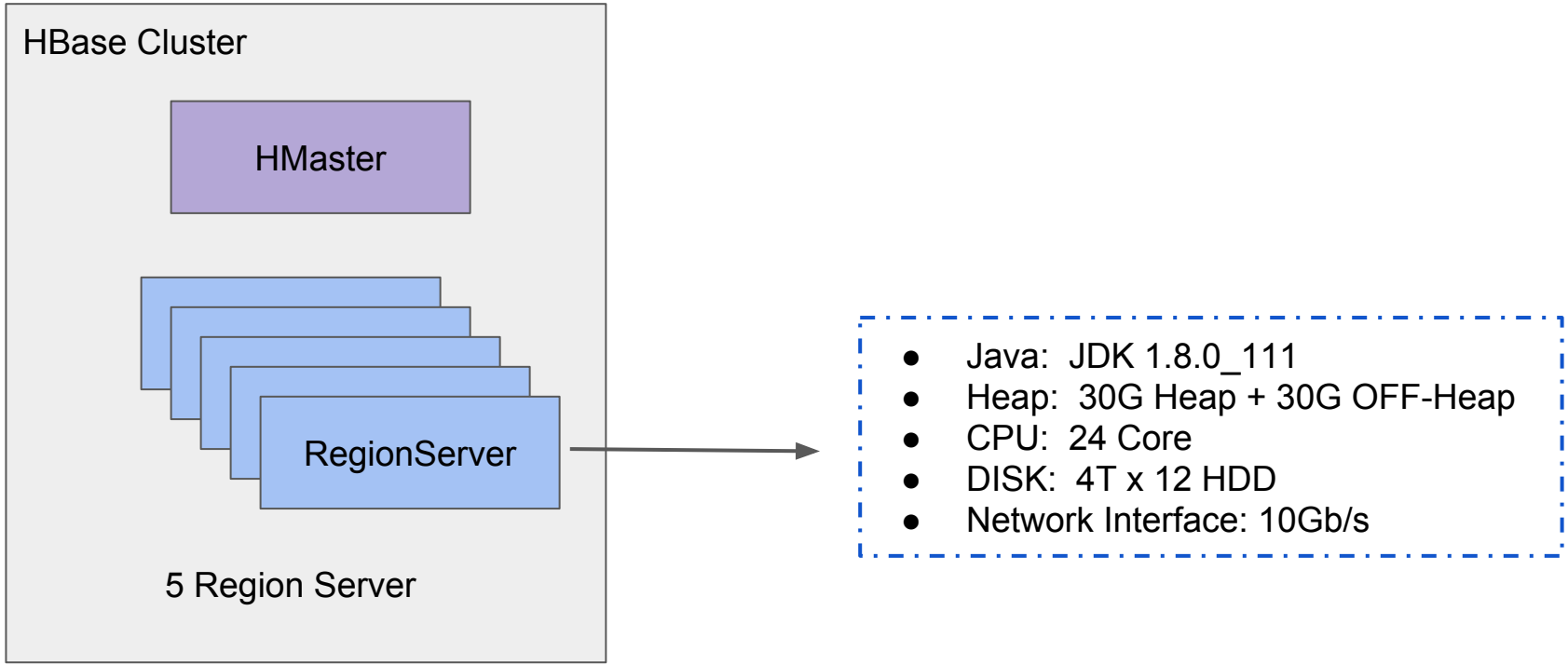
CMS *VS* G1

- STW Full GC
  - CMS can only compact fragments when full GC, so theoretically you can not avoid full GC.
  - G1 will compact fragments incrementally by multiple mixed GC. so it provides the ability to avoid full GC.
- Heap Size
  - G1 is more suitable for huge heap than CMS

Pressure Test

- Pre-load data
  - A new table with 400 regions
  - 100 millions rows whose value is 1000 bytes
- Pressure test for G1GC tuning
  - 40 write clients + 20 read clients
  - 1 hour for each JVM option changed
- HBase configuration
  - 0.3 <= global memstore limit <= 0.45
  - hfile.block.cache.size = 0.1
  - hbase.hregion.memstore.flush.size = 256 MB
  - hbase.bucketcache.ioengine = offheap

Test Environment



Initial JVM Options

-Xmx30g -Xms30g
-XX:MaxDirectMemorySize=30g
-XX:+UseG1GC
-XX:+UnlockExperimentalVMOptions
-XX:**MaxGCPauseMillis**=90
-XX:**G1NewSizePercent**=1
-XX:**InitiatingHeapOccupancyPercent**=30
-XX:+ParallelRefProcEnabled
-XX:ConcGCThreads=4
-XX:ParallelGCThreads=16
-XX:**MaxTenuringThreshold**=1
-XX:**G1HeapRegionSize**=32m
-XX:**G1MixedGCCountTarget**=32
-XX:**G1OldCSetRegionThresholdPercent**=5

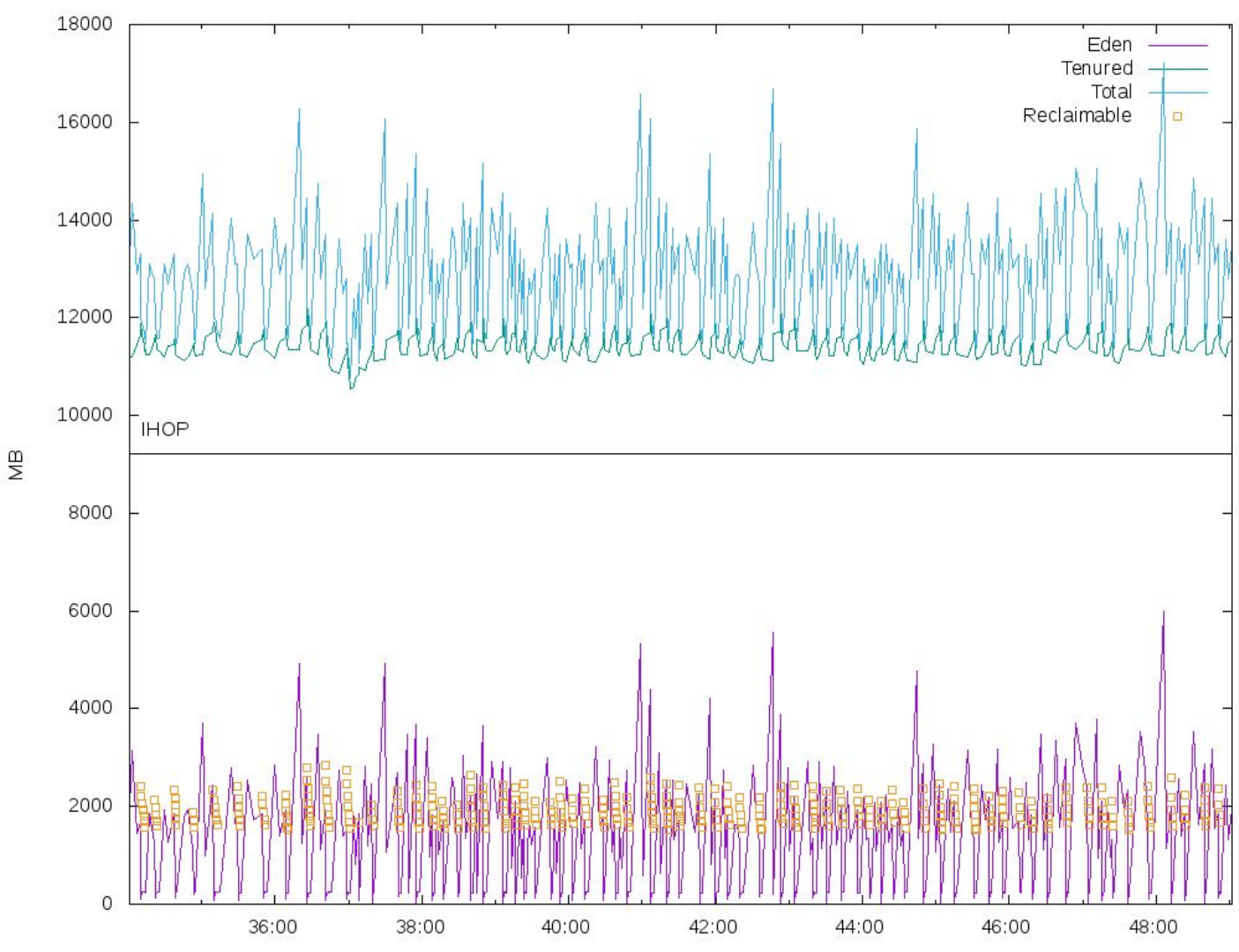
-verbose:gc
-XX:+PrintGC
-XX:+PrintGCDetails
-XX:+PrintGCApplicationStoppedTime
-XX:+PrintHeapAtGC
-XX:+PrintGCDateStamps
-XX:+PrintAdaptiveSizePolicy
-XX:+PrintTenuringDistribution
-XX:+PrintSafepointStatistics
-XX:PrintSafepointStatisticsCount=1
-XX:PrintFLSStatistics=1

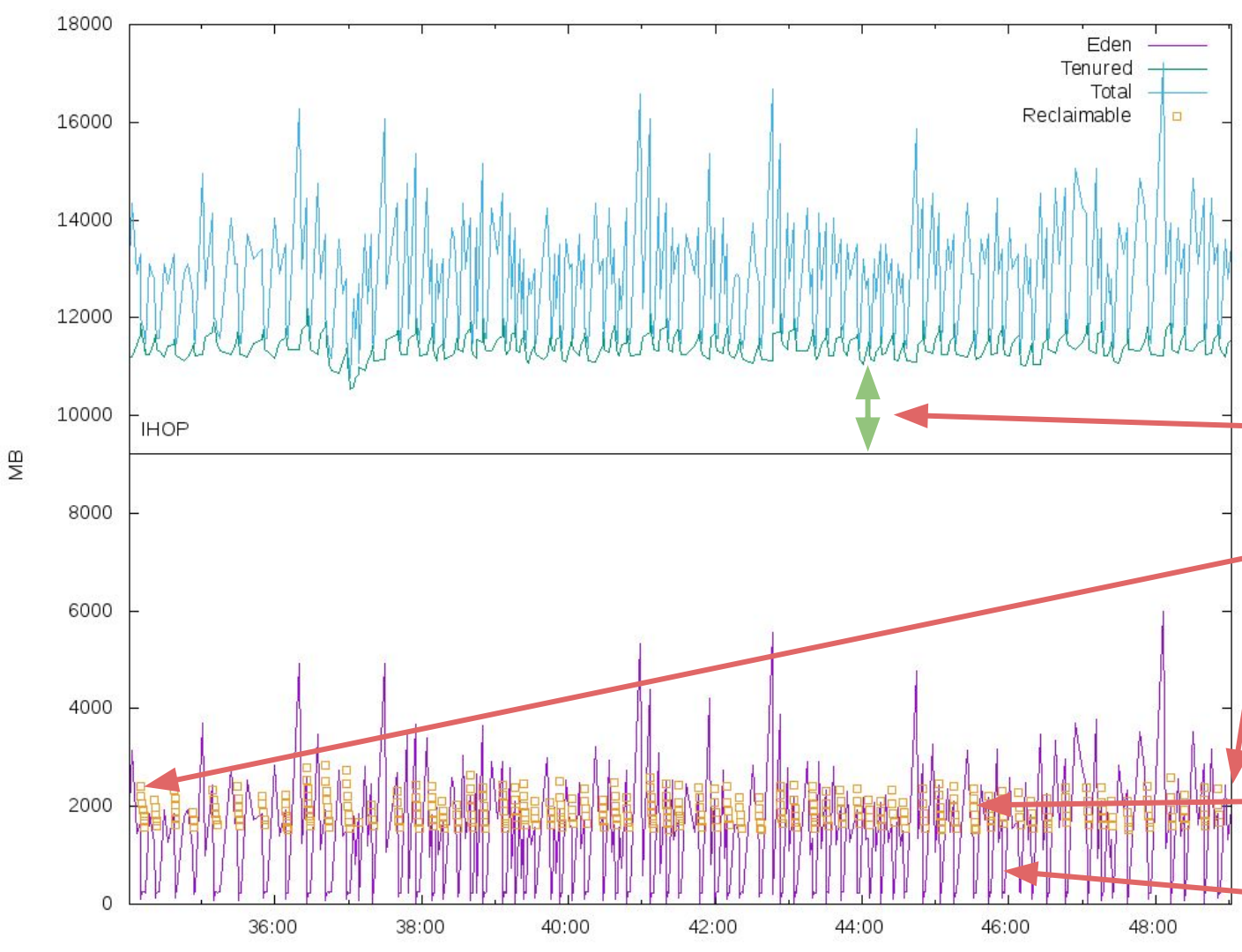
Options to print gc log

Important G1 Options

MaxGCPauseMillis	Sets a target for the maximum GC pause time. Soft Limit. (value: 90)
G1NewSizePercent	Minimum size for young generation. (value: 1)
InitiatingHeapOccupancyPercent	The Java heap occupancy threshold that triggers a concurrent GC cycle. (value: 65)
MaxTenuringThreshold	Maximum value for tenuring threshold. (value: 1)
G1HeapRegionSize	Sets the size of a G1 region. (value: 32m)
G1MixedGCCountTarget	The target number of mixed garbage collections after a marking cycle to collect old regions. (value: 32)
G1OldCSetRegionThresholdPercent	Sets an upper limit on the number of old regions to be collected during a mixed garbage collection cycle. (value: 5)

15 min period





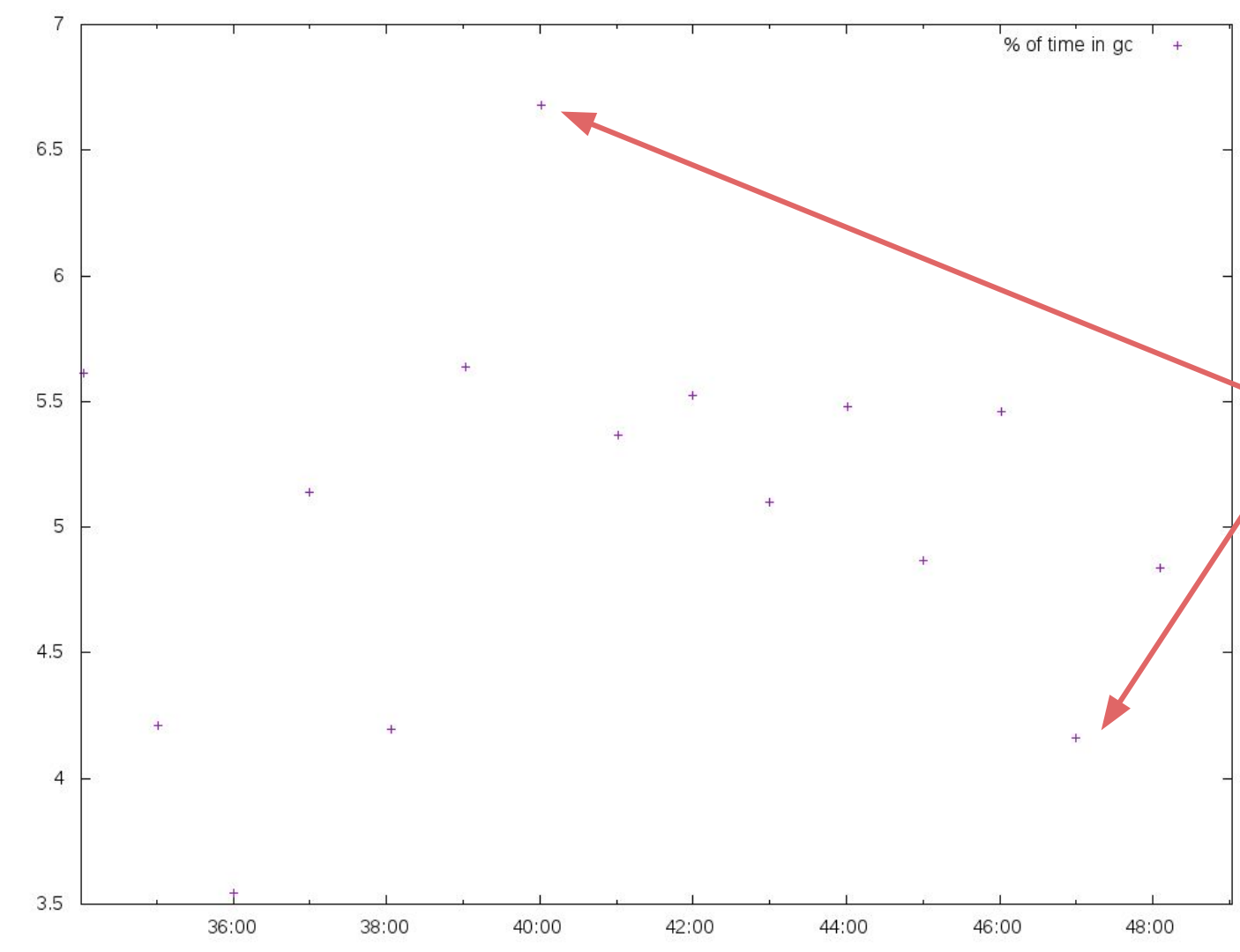
15 min period

Real heap usage is much higher than IHOP (Bad)

70 Mixed GC cycles in 15 min Too Frequently (Bad)

GC few garbage for one mixed gc cycle (Bad)

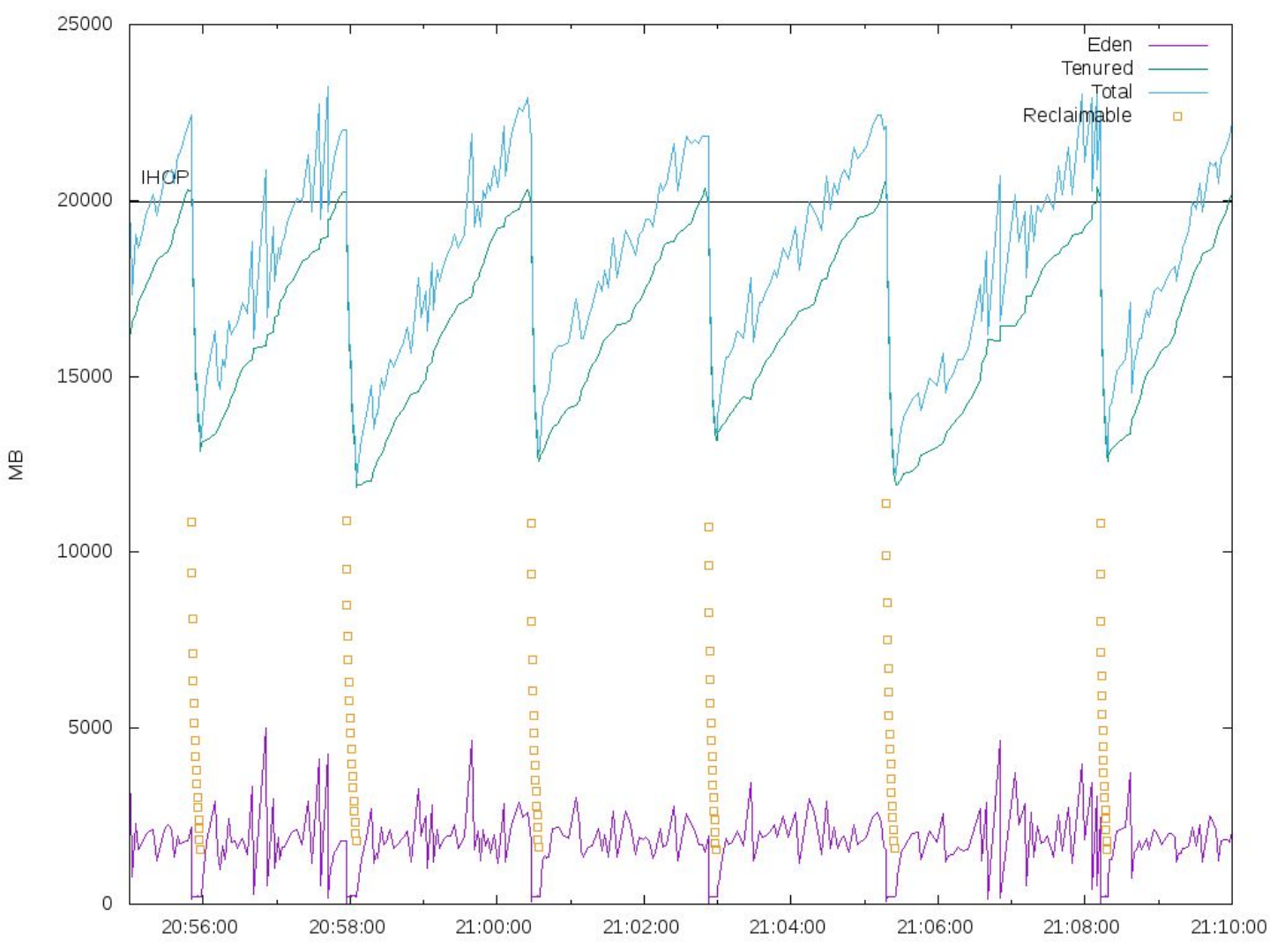
Young gen adaptivity(Good)

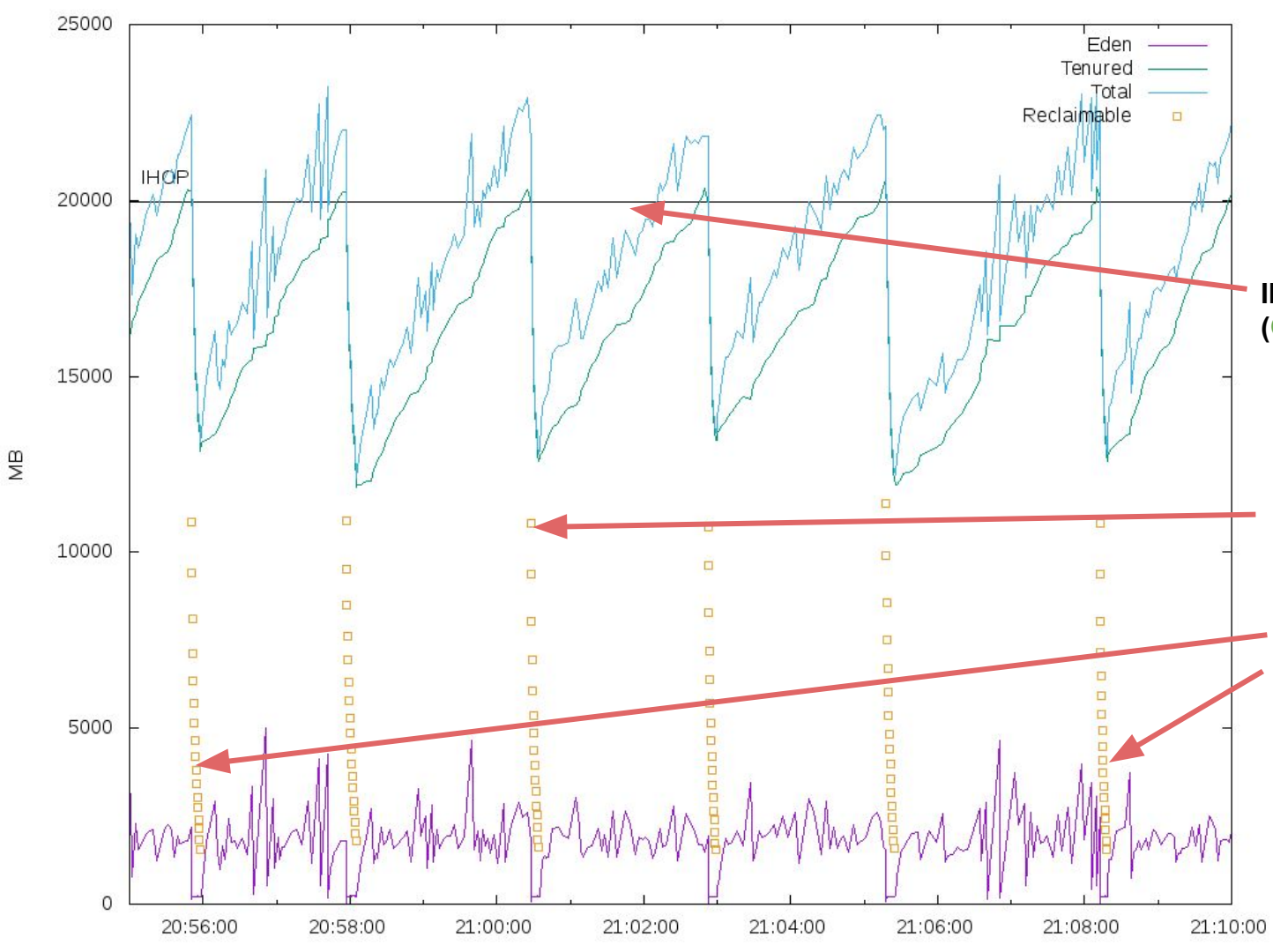


GC cost 4.2%~6.7% time (Bad)

Tuning #1

- How do we tuning ?
  - IHOP > MaxMemstoreSize%Heap + L1CacheSize%Heap + Delta(~ 10%)
  - Bucket Cache is offheap, need NO consideration when tuning IHOP
- Next Tuning
  - MemstoreSize = 45% , L1CacheSize ~ 10%, Delta ~ 10%
  - Increase InitiatingHeapOccupancyPercent to 65

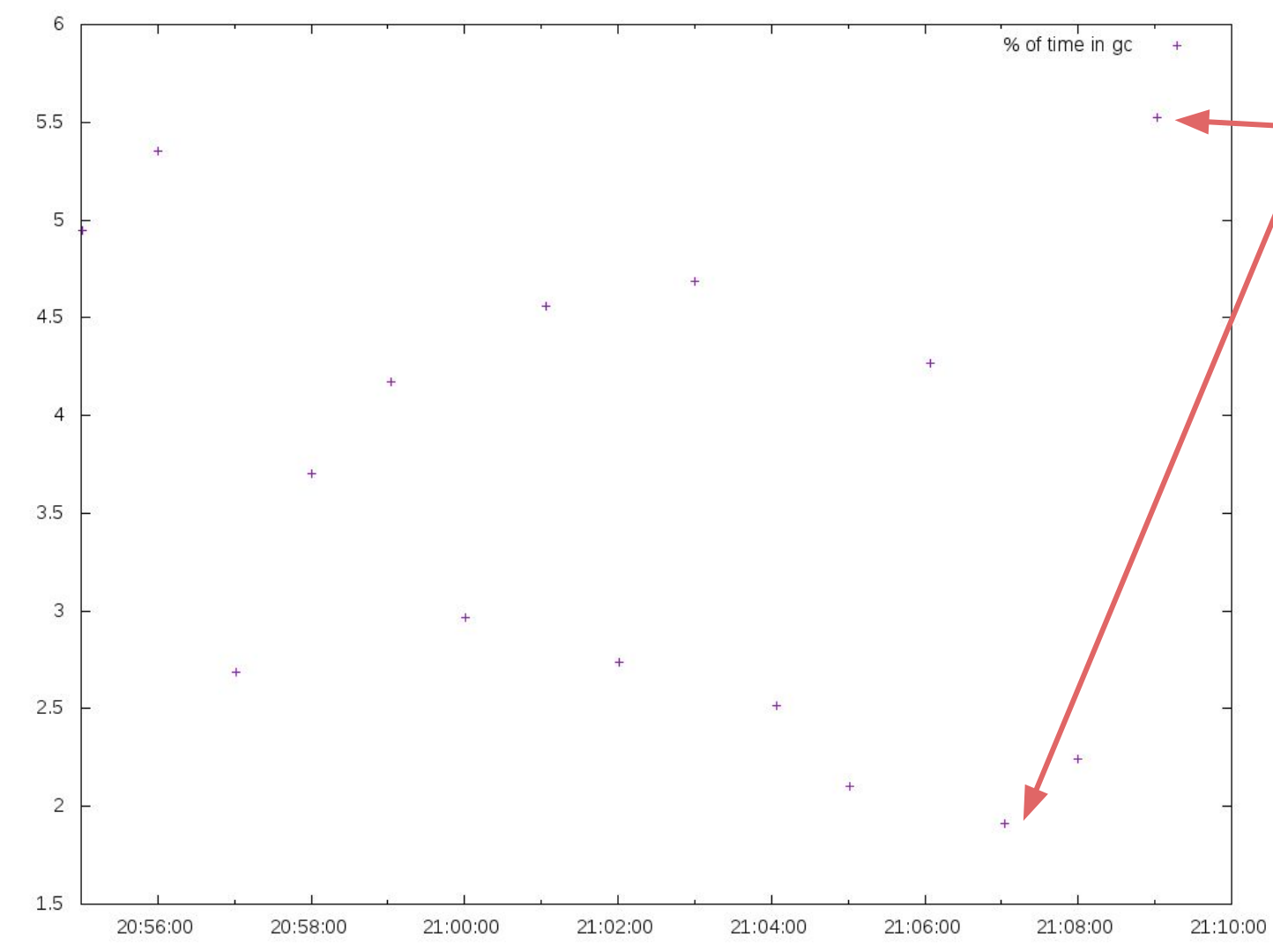

Eden
Tenured
Total
Reclaimable
IHOP
MB
25000
20000
15000
10000
5000
0
20:56:00
20:58:00
21:00:00
21:02:00
21:04:00
21:06:00
21:08:00
21:10:00



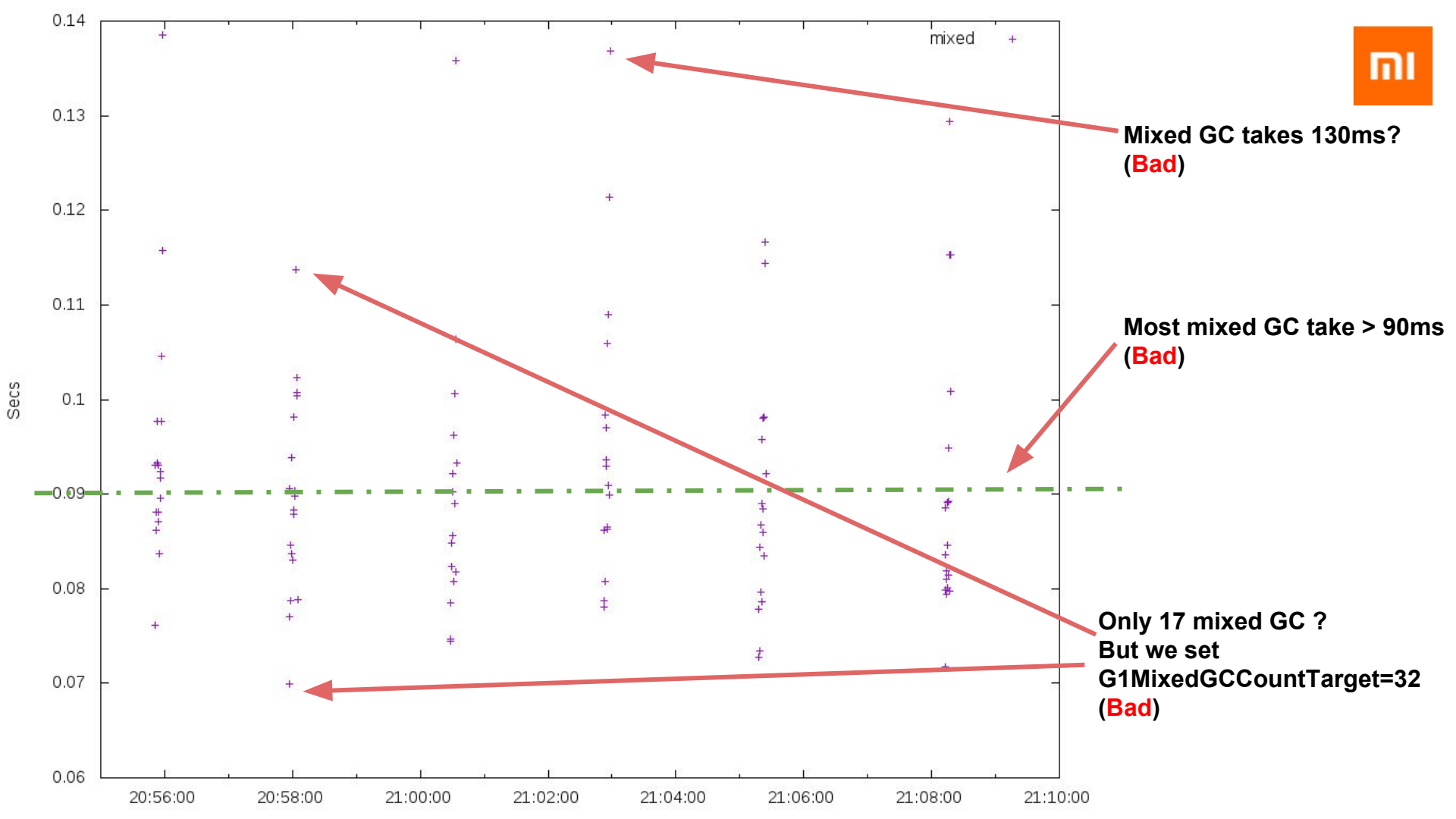
IHOP works as expected (Good)

Mixed GC reclaim more garbage(Good)

6 Mixed GC cycles in 15 min(Better)



Time in GC decrease(Good)
[4.2%~6.7%] -> [1.8%~5.5%]


mixed
Secs
0.14
0.13
0.12
0.11
0.1
0.09
0.08
0.07
0.06
20:56:00
20:58:00
21:00:00
21:02:00
21:04:00
21:06:00
21:08:00
21:10:00
Mixed GC takes 130ms?
(Bad)
Most mixed GC take > 90ms
(Bad)
Only 17 mixed GC ?
But we set
G1MixedGCCountTarget=32
(Bad)

Tuning #2

Let’s analyze our mixed gc logs

```
candidate old regions: 550 regions, reclaimable: 11197967672 bytes (34.76 %), threshold: 5.00 %]
candidate old regions: 502 regions, reclaimable: 9721616576 bytes (30.18 %), threshold: 5.00 %]
candidate old regions: 454 regions, reclaimable: 8380703408 bytes (26.02 %), threshold: 5.00 %]
candidate old regions: 417 regions, reclaimable: 7347629848 bytes (22.81 %), threshold: 5.00 %]
candidate old regions: 382 regions, reclaimable: 6468144704 bytes (20.08 %), threshold: 5.00 %]
candidate old regions: 354 regions, reclaimable: 5796441648 bytes (17.99 %), threshold: 5.00 %]
candidate old regions: 332 regions, reclaimable: 5226120128 bytes (16.22 %), threshold: 5.00 %]
candidate old regions: 310 regions, reclaimable: 4747368896 bytes (14.74 %), threshold: 5.00 %]
candidate old regions: 288 regions, reclaimable: 4253081576 bytes (13.20 %), threshold: 5.00 %]
candidate old regions: 268 regions, reclaimable: 3837972680 bytes (11.91 %), threshold: 5.00 %]
candidate old regions: 250 regions, reclaimable: 3446920000 bytes (10.70 %), threshold: 5.00 %]
candidate old regions: 232 regions, reclaimable: 3039399712 bytes (9.44 %), threshold: 5.00 %]
candidate old regions: 214 regions, reclaimable: 2726339720 bytes (8.46 %), threshold: 5.00 %]
candidate old regions: 196 regions, reclaimable: 2364987776 bytes (7.34 %), threshold: 5.00 %]
candidate old regions: 178 regions, reclaimable: 2089797048 bytes (6.49 %), threshold: 5.00 %]
candidate old regions: 160 regions, reclaimable: 1834664816 bytes (5.70 %), threshold: 5.00 %]
```

Tuning #2

Let's analyze our mixed gc logs

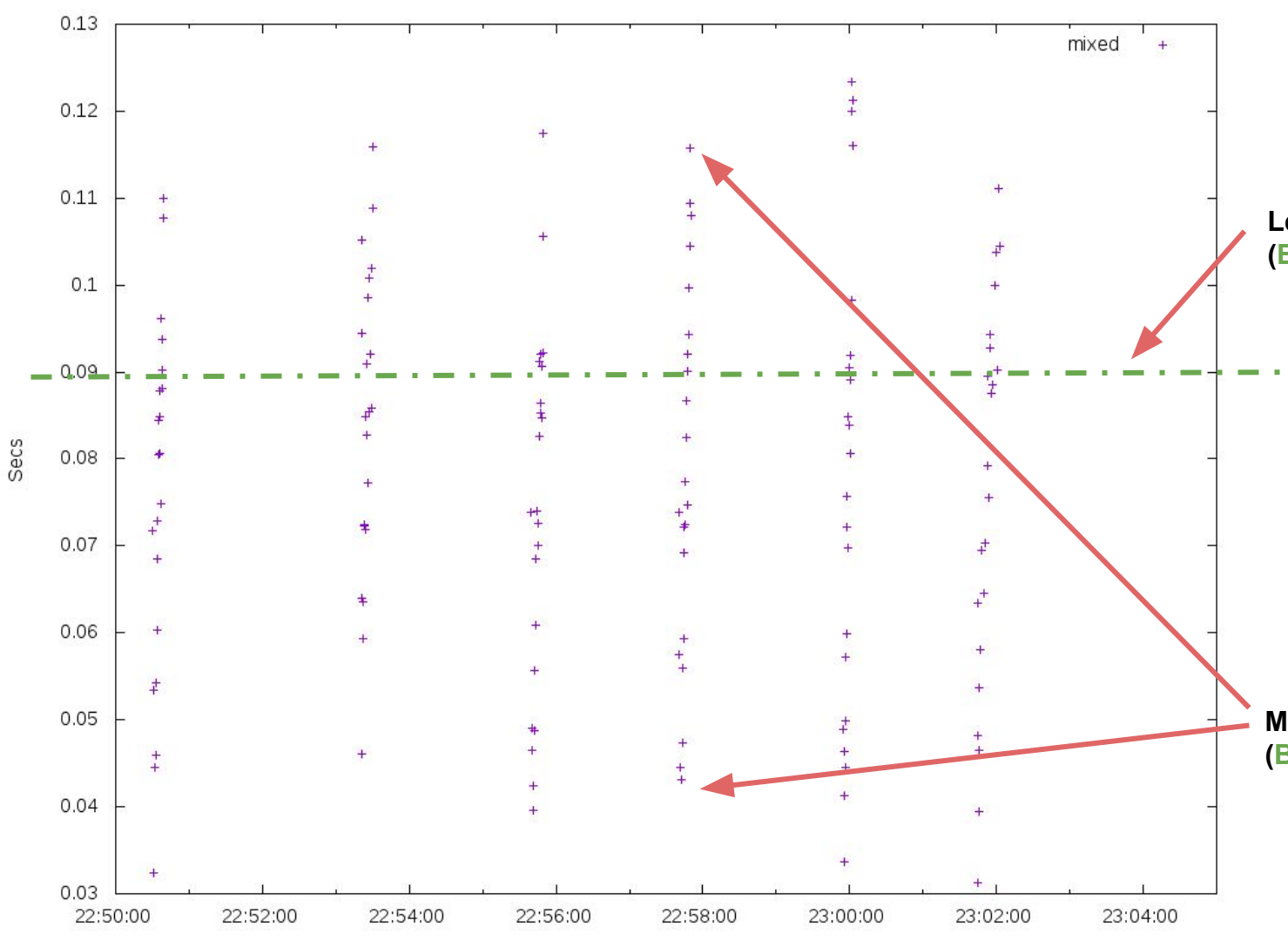
```
candidate old regions: 550 regions, reclaimable: 11197967672 bytes (34.76 %), threshold: 5.00 %]
candidate old regions: 502 regions, reclaimable: 9721616576 bytes (30.18 %), threshold: 5.00 %]
candidate old regions: 454 regions, reclaimable: 8380703408 bytes (26.02 %), threshold: 5.00 %]
candidate old regions: 417 regions, reclaimable: 7347629848 bytes (22.81 %), threshold: 5.00 %]
candidate old regions: 382 regions, reclaimable: 6468144704 bytes (20.08 %), threshold: 5.00 %]
candidate old regions: 354 regions, reclaimable: 5796441648 bytes (17.99 %), threshold: 5.00 %]
candidate old regions: 332 regions, reclaimable: 5226120128 bytes (16.22 %), threshold: 5.00 %]
candidate old regions: 310 regions, reclaimable: 4747368896 bytes (14.74 %), threshold: 5.00 %]
candidate old regions: 288 regions, reclaimable: 4253081576 bytes (13.20 %), threshold: 5.00 %]
candidate old regions: 268 regions, reclaimable: 3837972680 bytes (11.91 %), threshold: 5.00 %]
candidate old regions: 250 regions, reclaimable: 3446920000 bytes (10.70 %), threshold: 5.00 %]
candidate old regions: 232 regions, reclaimable: 3039399712 bytes (9.44 %), threshold: 5.00 %]
candidate old regions: 214 regions, reclaimable: 2726339720 bytes (8.46 %), threshold: 5.00 %]
candidate old regions: 196 regions, reclaimable: 2364987776 bytes (7.34 %), threshold: 5.00 %]
candidate old regions: 178 regions, reclaimable: 2089797048 bytes (6.49 %), threshold: 5.00 %]
candidate old regions: 160 regions, reclaimable: 1834664816 bytes (5.70 %), threshold: 5.00 %]
```

Cleanup 550-502=48 old regions every mixed gc

Stop mixed gc util G1HeapWastePercent

Tuning #2

- How do we tuning ?
  - G1OldCSetRegionThresholdPercent is 5.
  - 48 regions * 32MB(=1536MB) <= 30(g) * 1024 * 0.05 (=1536MB)
  - Try to decrease G1OldCSetRegionThresholdPercent for reducing mixed gc time.
- Next Tuning
  - Decrease G1OldCSetRegionThresholdPercent to 2



Less mixed gc take > 90ms
(Better)

More mixed gc(~22 times)
(Better)

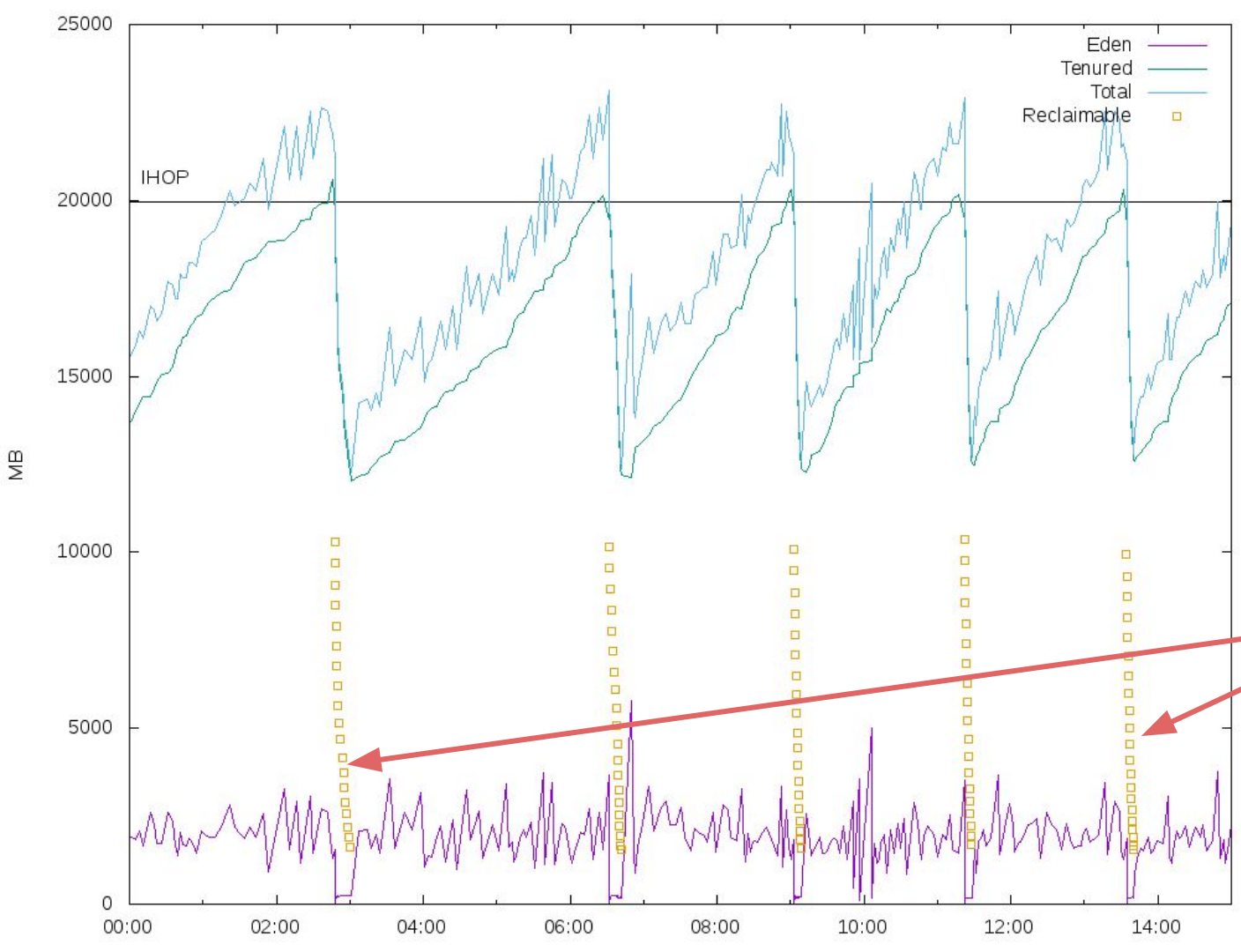
Another Problem

```
2017-08-01T23:44:24.513+0800: 6941.966: [GC pause (G1 Evacuation Pause) (young)
Desired survivor size 33554432 bytes, new threshold 1 (max 1)
- age   1:   54715128 bytes,   54715128 total
2017-08-01T23:44:25.052+0800: 6942.505: [GC pause (G1 Evacuation Pause) (young)
Desired survivor size 33554432 bytes, new threshold 1 (max 1)
- age   1:   54556840 bytes,   54556840 total
2017-08-01T23:44:25.676+0800: 6943.129: [GC pause (G1 Evacuation Pause) (young)
Desired survivor size 33554432 bytes, new threshold 1 (max 1)
- age   1:   56378696 bytes,   56378696 total
2017-08-01T23:44:26.203+0800: 6943.656: [GC pause (G1 Evacuation Pause) (young)
Desired survivor size 33554432 bytes, new threshold 1 (max 1)
- age   1:   51437840 bytes,   51437840 total
2017-08-01T23:44:27.073+0800: 6944.526: [GC pause (G1 Evacuation Pause) (young)
Desired survivor size 33554432 bytes, new threshold 1 (max 1)
- age   1:   54067080 bytes,   54067080 total
2017-08-01T23:44:27.446+0800: 6944.899: [GC pause (G1 Evacuation Pause) (mixed)
Desired survivor size 33554432 bytes, new threshold 1 (max 1)
- age   1:   51810536 bytes,   51810536 total
```

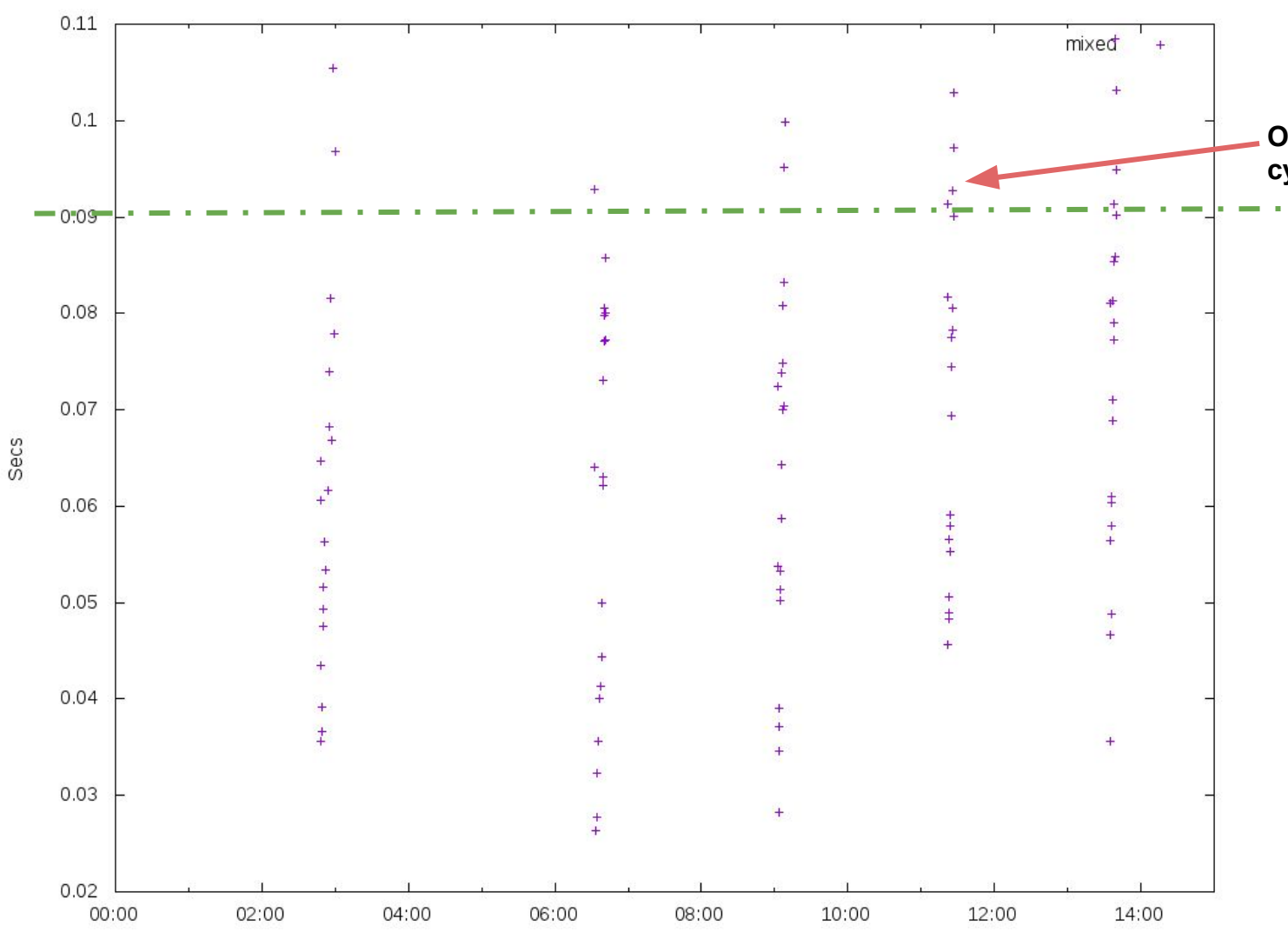
33,554,432< 54,067,080 ? Many objects with age =1 in Eden gen will be moved into old gen directly , which lead to old gen increasing so fast. finally mix gc occur frequently. (Bad)

Tuning #3

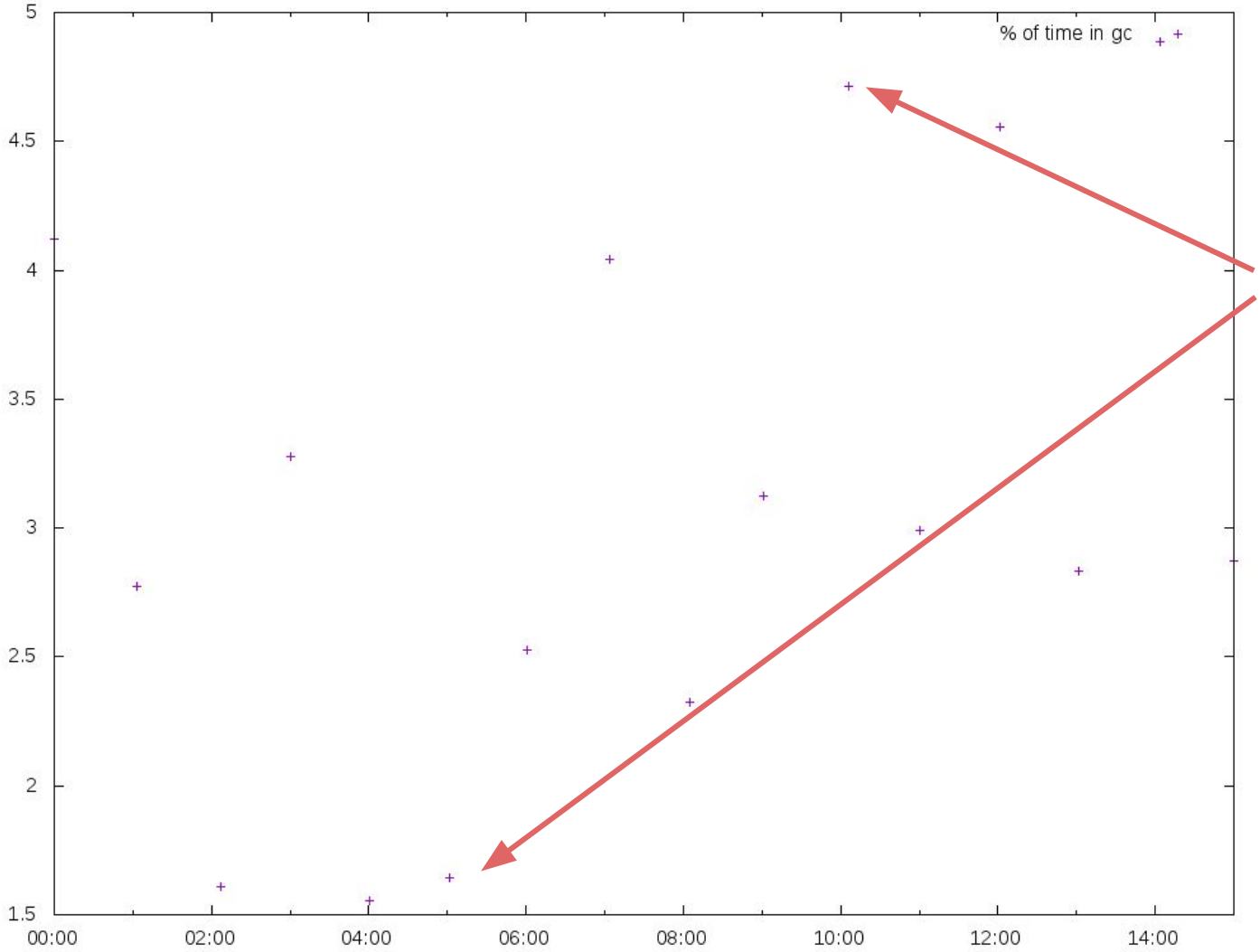
- How do we tuning ?
  - Total Survivor size >= 30(G) * 1024 * 0.01 / 8 = 38.4 MB
  - Age-1-Size is about 50 MB ~ 60MB
- Next Tuning
  - Set SurvivorRatio from 8 (default) to 4



5 Mixed GC cycles in 15 min(Better) (6 -> 5)



Only ~3 mixed gc in the cycle take > 90ms (Better)



Time in GC decrease(Better)
[1.8%~5.5%] -> [1.6%~4.7%]

Conclusion

-Xmx30g -Xms30g
-XX:MaxDirectMemorySize=30g
-XX:+UseG1GC
-XX:+UnlockExperimentalVMOptions
-XX:**MaxGCPauseMillis**=90
-XX:**G1NewSizePercent**=1
-XX:**InitiatingHeapOccupancyPercent**=**65** (Before 30)
-XX:+ParallelRefProcEnabled
-XX:ConcGCThreads=4
-XX:ParallelGCThreads=16
-XX:**MaxTenuringThreshold**=1
-XX:**G1HeapRegionSize**=32m
-XX:**G1MixedGCCountTarget**=32
-XX:**G1OldCSetRegionThresholdPercent**=**2** (Before 5)
-XX:**SurvivorRatio**=**4** (Before 8)

Conclusion

- Current state
  - Young generation adaptivity (**Good**)
  - About 5 mixed gc cycles in 15 mins (**Good**)
  - Few(~3) mixed gc take more than 90ms (**Good**)
  - Overall time in gc is [1.6%~4.7%] (**Good**)
- Next tuning
  - Let’s increase our test pressure
  - 1 billion rows whose value is 1000 bytes (100 million rows before)
  - 200 write clients + 100 read clients (40 write clients + 20 read clients before)

Let’s analyze our mixed gc logs

```
2017-08-02T10:49:31.379+0800: 36374.552: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:31.480+0800: 36374.654: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:31.954+0800: 36375.128: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:32.052+0800: 36375.226: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:32.694+0800: 36375.867: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:32.925+0800: 36376.099: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:33.535+0800: 36376.709: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:34.180+0800: 36377.354: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:34.644+0800: 36377.818: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:34.759+0800: 36377.933: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:34.988+0800: 36378.161: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:35.541+0800: 36378.715: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:36.048+0800: 36379.221: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T10:49:36.181+0800: 36379.355: [GC pause (G1 Evacuation Pause) (mixed)
```

4 continuous mixed gc happen in 1 second (Bad)

2 continuous mixed gc happen in 101ms (Bad)

Tuning #4

- Problem
  - Multiple continuous mixed gc happen in short time interval
  - A RPC lifetime may cross multiple continuous mixed gc which lead to the RPC become very slow
- Reason
  - Mixed GC consume the expected MaxGCPauseMillis, and some mixed gc take even longer time
  - G1 adjust the young gen to 1%
  - But because of our high QPS, young gen is consumed quickly.
  - We are in a mixed gc cycle.
  - Finally, G1 trigger multiple continuous mixed gc frequently

Tuning #4

- Typical G1 GC tuning method
  - Increase MaxGCPauseMillis to meet our mixed gc cost ?
    - But young gc will also spend more time.
- Trick
  - Increase G1NewSizePercent to enlarge the young gen size
    - Young GC will not be affected.
    - More young gen during mixed gc for avoiding trigger mixed gc by running out of young gen.
- Next tuning
  - Increase our G1NewSizePercent to 4

Let’s analyze our mixed gc logs

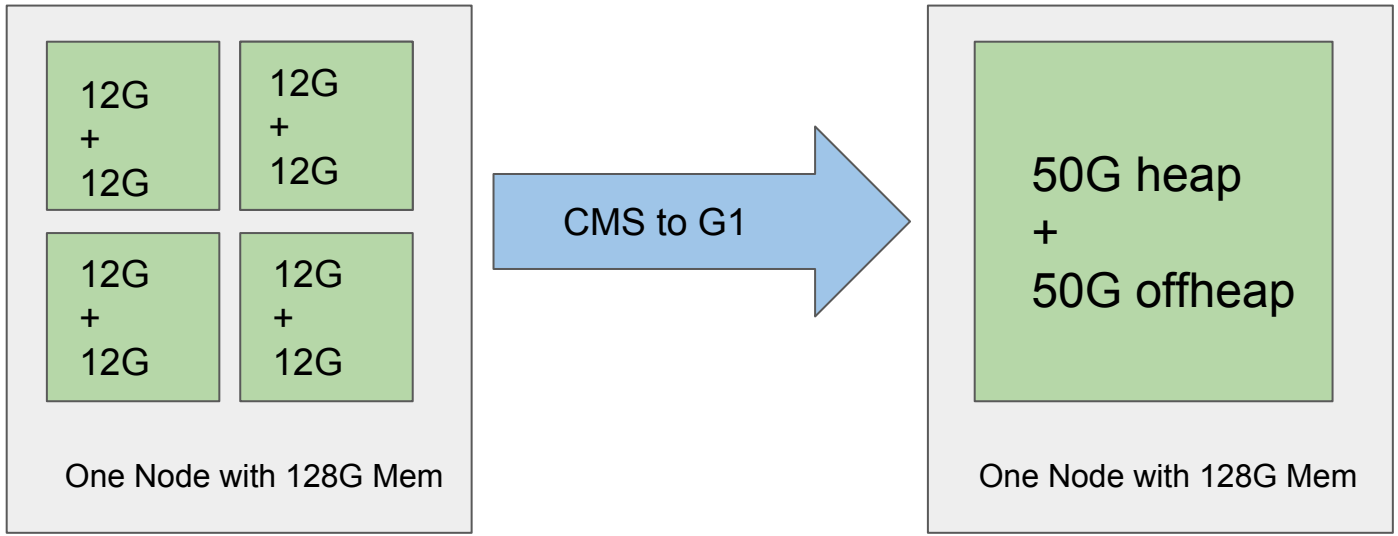
```
2017-08-02T12:50:12.551+0800: 3288.365: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:13.937+0800: 3289.751: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:15.871+0800: 3291.685: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:17.397+0800: 3293.211: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:18.851+0800: 3294.665: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:20.498+0800: 3296.312: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:22.838+0800: 3298.652: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:24.452+0800: 3300.266: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:26.477+0800: 3302.291: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:28.770+0800: 3304.584: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:31.075+0800: 3306.889: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:32.732+0800: 3308.546: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:34.820+0800: 3310.634: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:37.291+0800: 3313.105: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:39.583+0800: 3315.397: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:41.971+0800: 3317.785: [GC pause (G1 Evacuation Pause) (mixed)
2017-08-02T12:50:44.435+0800: 3320.249: [GC pause (G1 Evacuation Pause) (mixed)
```

2 continuous mixed gc interval >= 1 sec(Better)

Conclusion

- Initial IHOP
  - BucketCache OFF-Heap
    - IHOP > MaxMemstore%Heap + MaxL1CacheSize%Heap + Delta
  - BucketCache On-Heap
    - IHOP > MaxMemstore%Heap + MaxL1CacheSize%Heap + MaxL2CacheSize%Heap + Delta
- Young GC Cost
  - Choose the right G1NewSizePercent & MaxGCPauseMillis
- Mixed GC Cost
  - G1MixedGCCountTarget (Mixed GC cycle)
  - G1OldCSetRegionThresholdPercent (Single Mixed GC time)
  - Take care of your survivorRatio.

G1GC In XiaoMi Cluster



- Less Full GC, Better availability
- Less region servers.

G1GC Options In XiaoMi Cluster

-Xmx50g -Xms50g
-XX:MaxDirectMemorySize=50g
-XX:+UseG1GC
-XX:+UnlockExperimentalVMOptions
-XX:**MaxGCPauseMillis**={50/90/500} for SSD/HDD/offline cluster
-XX:**G1NewSizePercent**={2/5} for normal/heavy load cluster
-XX:**InitiatingHeapOccupancyPercent**=65
-XX:+ParallelRefProcEnabled
-XX:ConcGCThreads=4
-XX:ParallelGCThreads=16
-XX:**MaxTenuringThreshold**=1
-XX:**G1HeapRegionSize**=32m
-XX:**G1MixedGCCountTarget**=64
-XX:**G1OldCSetRegionThresholdPercent**=5

-verbose:gc
-XX:+PrintGC
-XX:+PrintGCDetails
-XX:+PrintGCApplicationStoppedTime
-XX:+PrintHeapAtGC
-XX:+PrintGCDateStamps
-XX:+PrintAdaptiveSizePolicy
-XX:+PrintTenuringDistribution
-XX:+PrintSafepointStatistics
-XX:PrintSafepointStatisticsCount=1
-XX:PrintFLSStatistics=1

Thank You !