Будьте готовы к G1 GC,

или Эволюция G1 GC

Нурисламов Ильдар
Adserver Team Leader, GetIntent
https://twitter.com/absorbb
inurislamov@getintent.com
absorbb@gmail.com
skype: absorbb

План

- Цели
- Предварительные условия
- Как мы тестировали
- Испытание 1. Статья на habrhabr.ru
- Испытание 2. Миграция на быстрый сервер
- Испытание 3. Тонкая настройка G1
- Испытание 4. Взгляд в будущее. Java 9
- Итоги

Цели

Кто мы?

Цели

Кто мы?

Highload Adtech-стартап!

Цели

Кто мы? Highload Adtech-стартап!

Что мы хотим?

Цели

Кто мы?	Highload Adtech-стартап!
Что мы хотим?	Сотни тысяч запросов в сек Успешных ответов > 99% Время ответа < 50ms

Цели

Кто мы? Highload Adtech-стартап!

Что мы хотим?
Сотни тысяч запросов в сек
Успешных ответов > 99%
Время ответа < 50ms

Что нам мешает?

Цели

Кто мы?	Highload Adtech-стартап!
Что мы хотим?	Сотни тысяч запросов в сек Успешных ответов > 99% Время ответа < 50ms
Что нам мешает?	Garbage collector!

Цели

Кто мы?	Highload Adtech-стартап!
Что мы хотим?	Сотни тысяч запросов в сек Успешных ответов > 99% Время ответа < 50ms
Что нам мешает?	Garbage collector!
	Правда??

План

- Цели
- **Предварительные условия**
- Как мы тестировали
- Испытание 1. Статья на habrhabr.ru
- Испытание 2. Миграция на быстрый сервер
- Испытание 3. Тонкая настройка G1
- Испытание 4. Взгляд в будущее. Java 9
- Итоги

Предварительные условия

- Требования ко времени ответа или проценту успешных ответов (SLA)
- Оценка влияния пауз GC на общее время ответа
- Метрики производительности
- Анализ логов GC

Метрики производительности

Что измерять:

- Время ответа – AVG, MAX, перцентили
- Количество таймаутов

Метрики производительности

Что измерять:

- Время ответа – AVG, MAX, перцентили
- Количество таймаутов

Как измерять:

- Java agents
- Серверные логи
- Системы мониторинга

Причины долгих ответов

- Третьи сервисы, хранилища
- Внешняя сеть
- Паузы GC
- Кривой код

```
public Date getTomorrow() {
      try {
            Thread.sleep(1000 * 60 * 60 * 24);
            return new Date();
      } catch (Exception ex) {
            return getTomorrow();
      }
}
```

Сбор логов GC

-XX:+PrintGC -XX:+PrintGCDetails

-XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps

-XX:+PrintTenuringDistribution

-XX:+PrintReferenceGC

-XX:+PrintCMSInitiationStatistics -XX:PrintCMSStatistics=1

-XX:+PrintAdaptiveSizePolicy

-Xloggc:/path/gc.log

Анализ логов GC

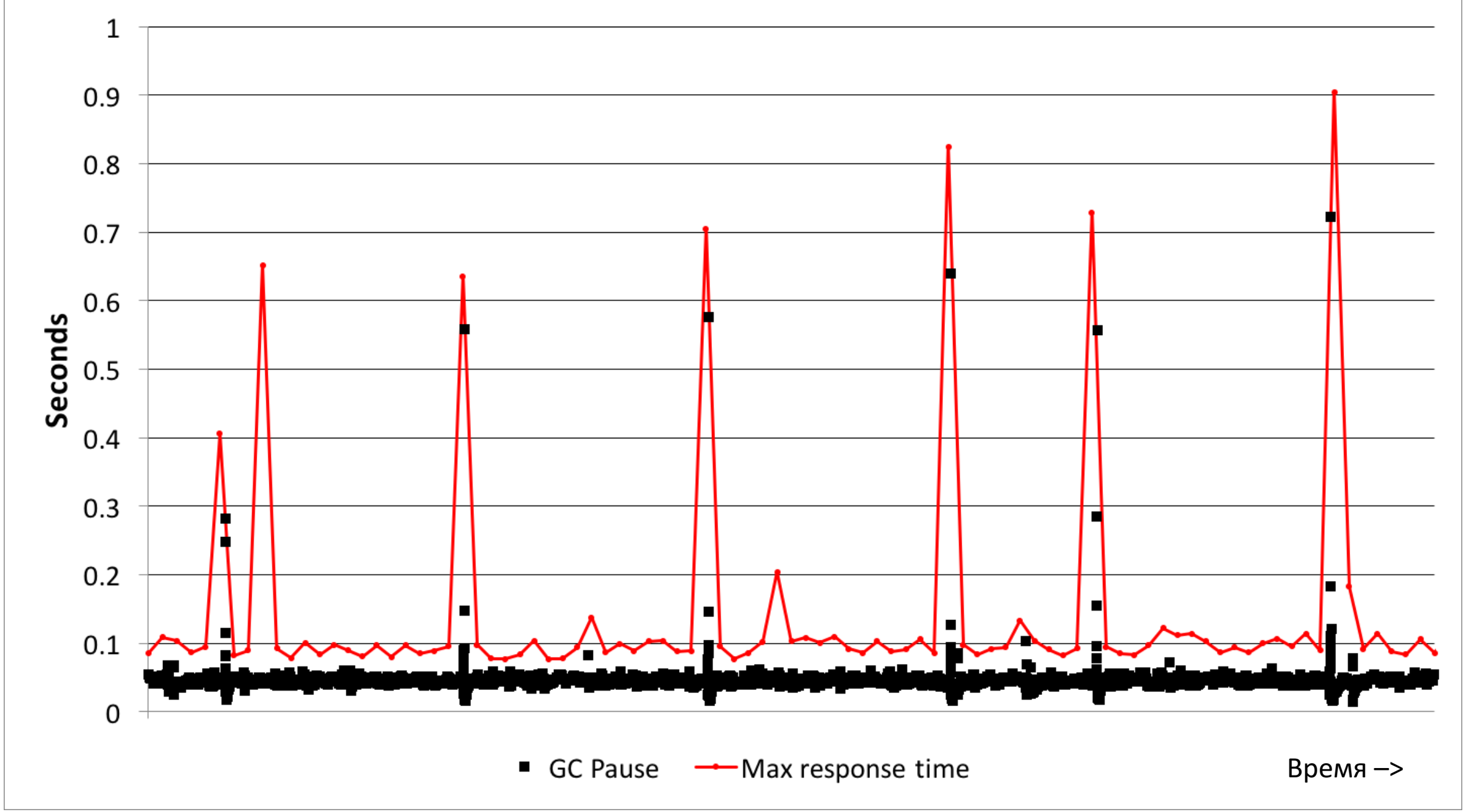
- GCeasy.io
- GCViewer
- Netflix gcviz
- ….

Анализ логов GC

- GCeasy.io
- GCViewer
- Netflix gcviz
- ….
- Сделать самому

Наш случай

Так правда ли Garbage collector мешает?


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Seconds
GC Pause
Max response time
Время –>

План

- Цели
- Предварительные условия
- **Как мы тестировали**
- Испытание 1. Статья на habrhabr.ru
- Испытание 2. Миграция на быстрый сервер
- Испытание 3. Тонкая настройка G1
- Испытание 4. Взгляд в будущее. Java 9
- Итоги

Как мы тестировали
Конфигурация

	System 1	System 2
CPU	Intel® Xeon® Processor E3-1246 v3 @ 3.50Ghz	2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz
Cores (Threads)	4 (8)	20 (40)
RAM	32Gb	128Gb
HDD	4HDD RAID10	
OS	Ubuntu Server 14.04	
JVM	Oracle JDK 1.8.0_102 JDK 9-ea+135	
SWAP	**OFF**	

Как мы тестировали
Нагрузка

- Запросы:
  S2S JSON/PROTOBUF HTTP(S) POST

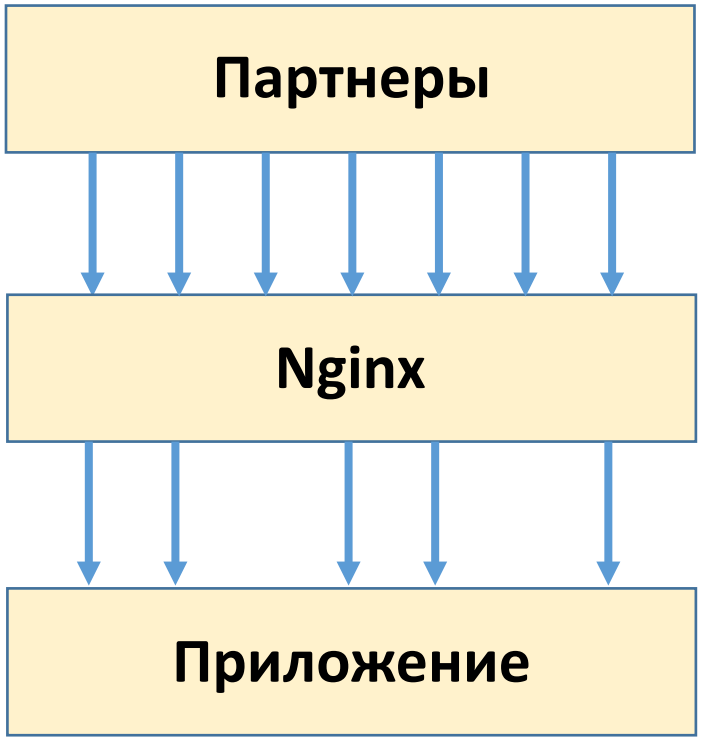
	System 1	System 2
QPS	6_000	25_000

- Фоновые задачи. Загрузки больших JSON.

Как мы тестировали
Условия

- 24 часа
- Условия боевые
- Ноды кластера одновременно запускались с разными настройками GC
- Результаты сравнивались

Схема тестирования



– Боевая загрузка

– Метрики, **подсчет таймаутов**

– Метрики, **логи GC**

Как мы тестировали
Параметры JVM

-XX:-TieredCompilation
-XX:ReservedCodeCacheSize=256m

Выключает C1 компилятор. Улучшает повторяемость результатов

-XX:+PerfDisableSharedMem

JDK-8076103 Writing to the mmaped PerfData file hsperfdata can cause long stalls on Linux

-XX:+DisableExplicitGC

Выключает System.gc()

Disclaimer

Ваши результаты могут не совпадать с нашими

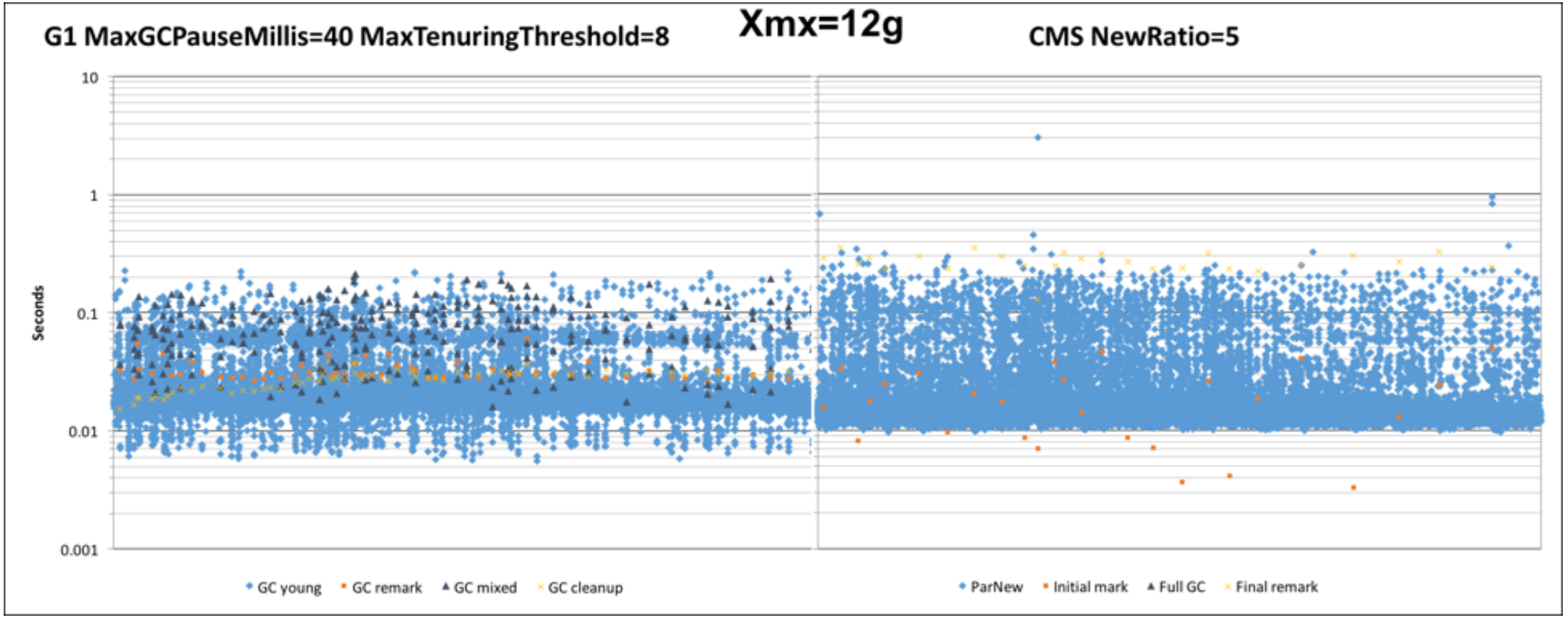
План

- Цели
- Предварительные условия
- Как мы тестировали
- **Испытание 1. Статья на habrhabr.ru**
- Испытание 2. Миграция на быстрый сервер
- Испытание 3. Тонкая настройка G1
- Испытание 4. Взгляд в будущее. Java 9
- Итоги

Испытание 1. Статья на habrhabr.ru

Выбор и настройка Garbage Collector для Highload системы в Hotspot JVM
https://habrahabr.ru/company/getintent/blog/302910/

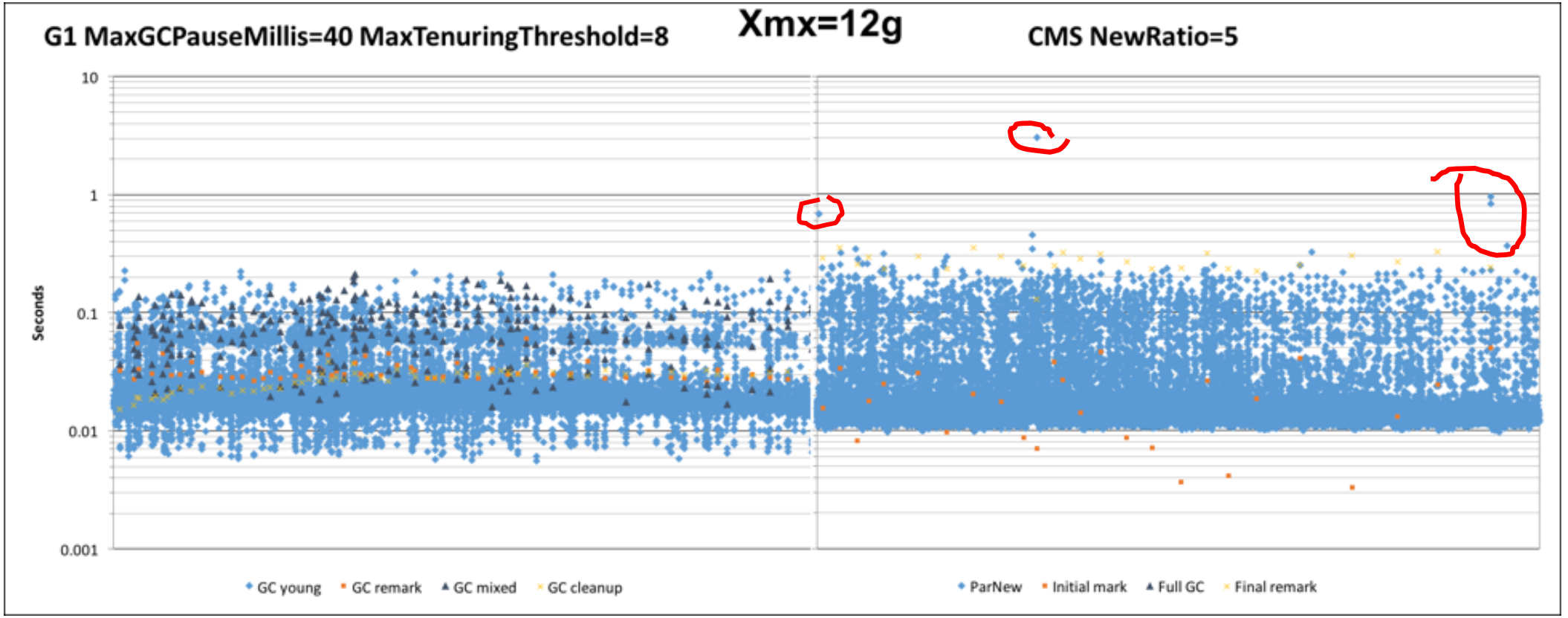
G1 vs CMS



https://habrahabr.ru/company/getintent/blog/302910/

Время –>

G1 vs CMS



https://habrahabr.ru/company/getintent/blog/302910/

Время –>

Параметры CMS
Какие использовались в статье?

-XX:NewRatio=5

Размер young generation. 1/6 от размера heap.

Параметры CMS
Какие используются сейчас. Лист 1

-Xmn=2500m

Размер young generation. 1/5 от размера heap.

-XX:+CMSScavengeBeforeRemark

Запускает young сборку перед remark фазой. Уменьшает кол-во работы для remark фазы.

-XX:+UseCMSInitiatingOccupancyOnly
-XX:CMSInitiatingOccupancyFraction=70

CMS больше не решает за нас, когда надо запускать сборку old generation. Позволяет избежать слишком поздних сборок.

Параметры CMS
Какие используются сейчас. Лист 2

-XX:+UnlockDiagnosticVMOptions

-XX:ParGCCardsPerStrideChunk=8192

Настраивает размер «chunk», на которые ParNew сборщик разбивает Old Gen для равномерного рапределения задач между потоками GC. По умолчанию 256.
С учетом того что размер card=512 байт, при размере Old Gen = 10G
кол-во таких «chunks» будет равно 76336. Излишние накладные расходы.

http://blog.ragozin.info/2012/03/secret-hotspot-option-improving-gc.html *by Alexey Ragozin*

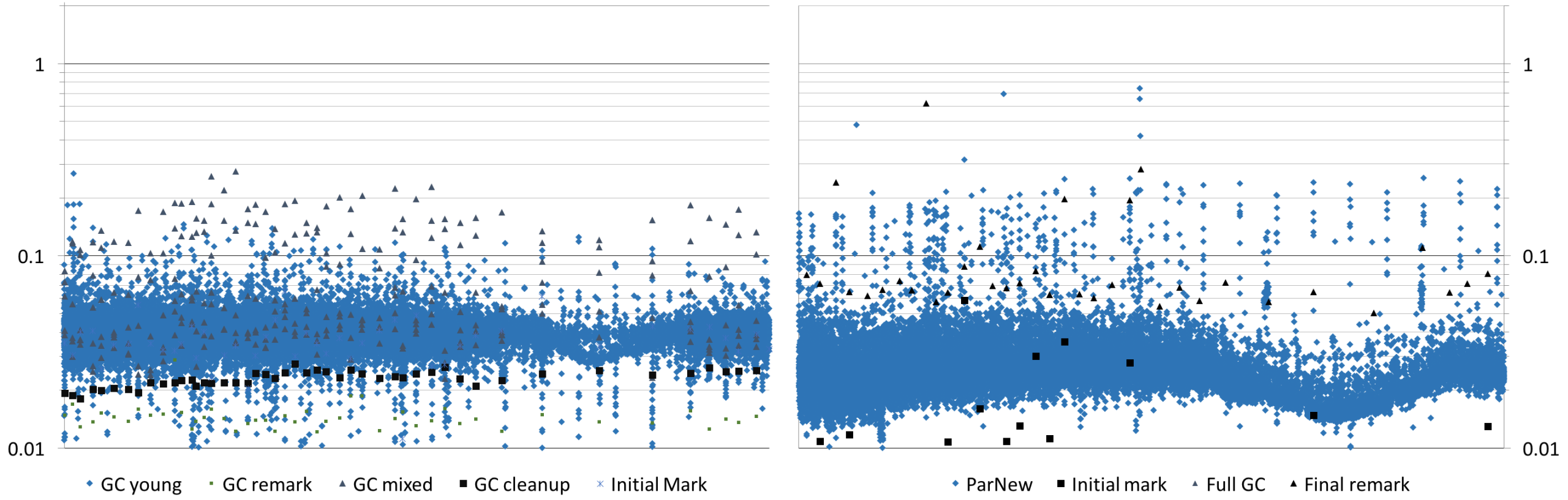
JDK-8086056 ParNew: auto-tune ParGCCardsPerStrideChunk

Параметры G1

-XX:MaxGCPauseMillis=40

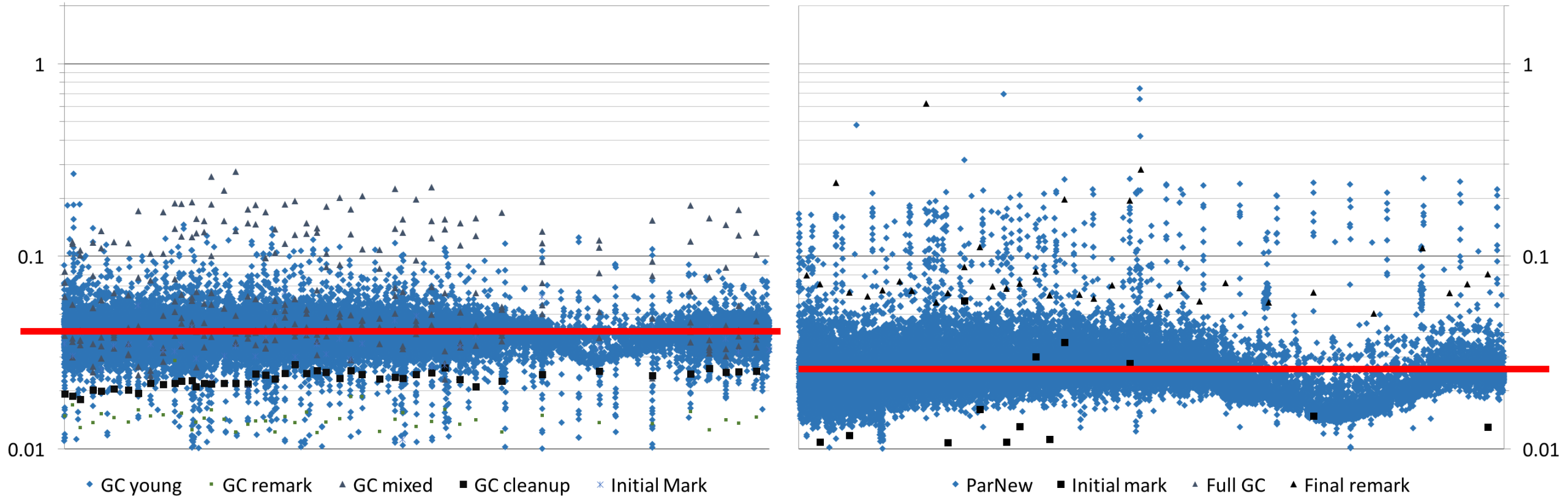
Основной параметр. Желаемая максимальная длительность паузы GC.
По умолчанию: 200

G1 vs CMS



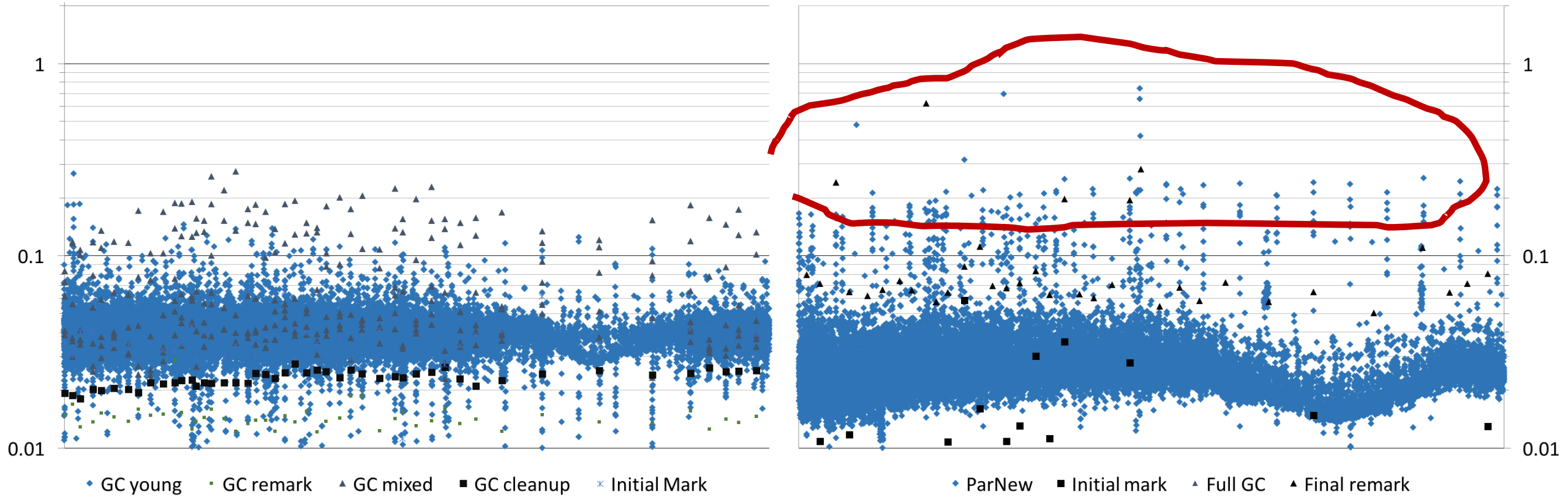
Intel® Xeon® Processor E3-1246 v3 @ 3.50Ghz / Oracle JDK 1.8.0_102 / -Xmx12g

G1 vs CMS



Intel® Xeon® Processor E3-1246 v3 @ 3.50Ghz / Oracle JDK 1.8.0_102 / -Xmx12g

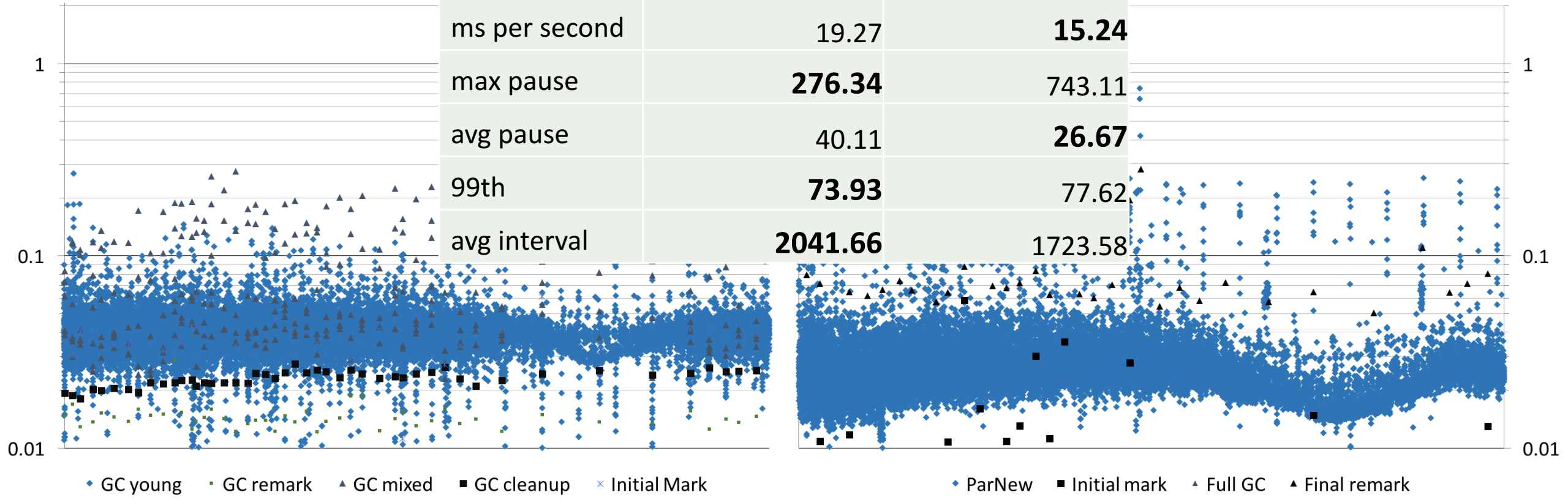
G1 vs CMS



Intel® Xeon® Processor E3-1246 v3 @ 3.50Ghz / Oracle JDK 1.8.0_102 / -Xmx12g

G1 vs CMS

	G1	CMS
ms per second	19.27	**15.24**
max pause	**276.34**	743.11
avg pause	40.11	**26.67**
99th	**73.93**	77.62
avg interval	**2041.66**	1723.58



Intel® Xeon® Processor E3-1246 v3 @ 3.50Ghz / Oracle JDK 1.8.0_102 / -Xmx12g

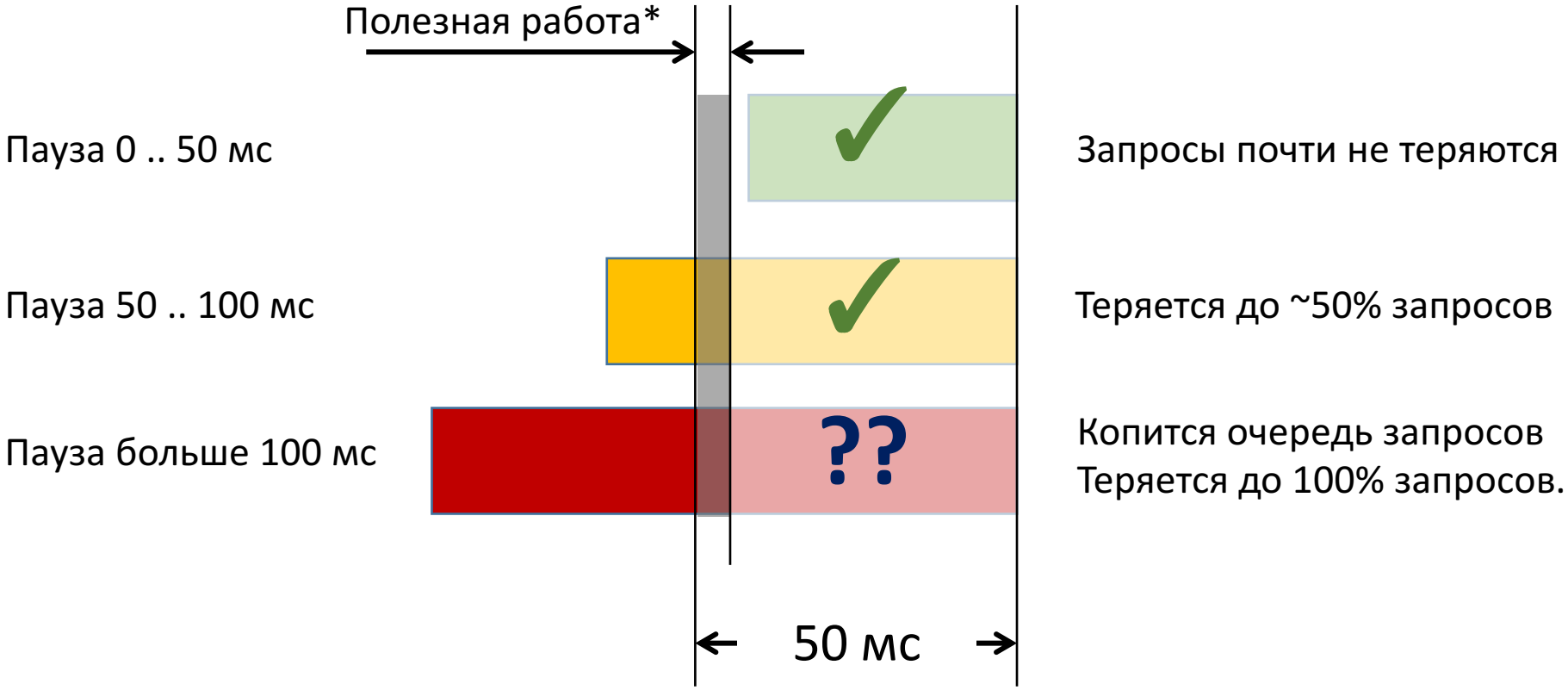
Анализ и представление результатов

- Цель – не бенчмарк GC
- Цель – выбор оптимального GC для нашей системы с учетом требований

Анализ и представление результатов

- Цель – не бенчмарк GC
- Цель – выбор оптимального GC для нашей системы с учетом требований
- Сколько времени GC препятствует выполнению требований
- Сколько запросов мы теряем

Сколько времени GC мешает



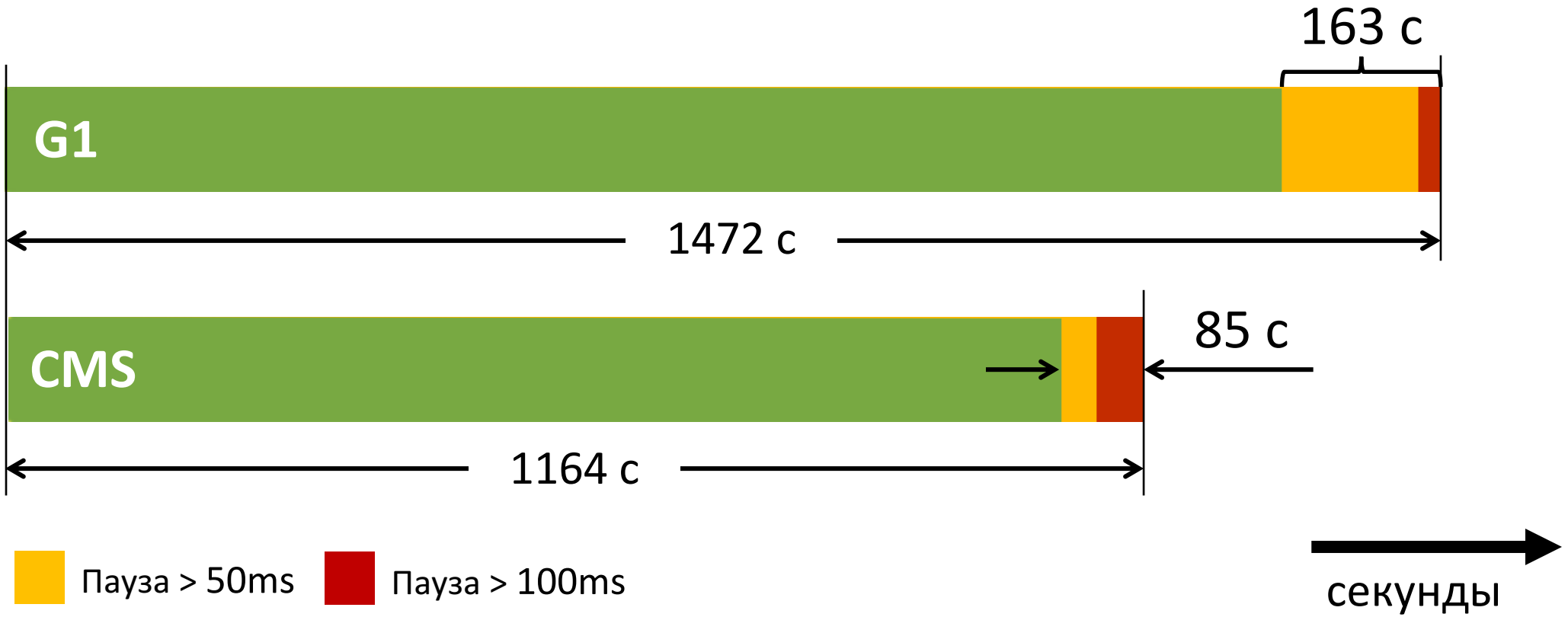
Запросы почти не теряются

Теряется до ~50% запросов

Копится очередь запросов
Теряется до 100% запросов.

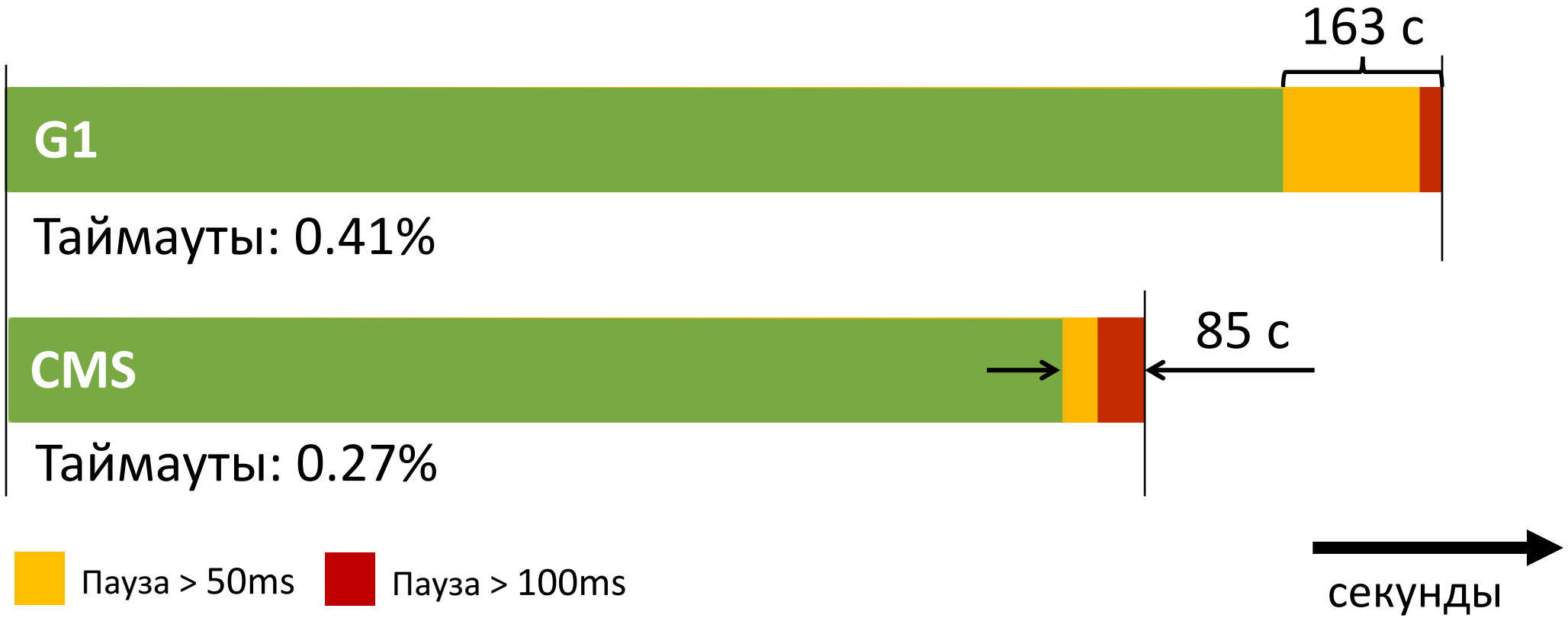
* Среднее время ответа ~2 мс

G1 vs CMS
суммарное время, проведенное в Stop The World



Intel® Xeon® Processor E3-1246 v3 @ 3.50Ghz / Oracle JDK 1.8.0_102 / -Xmx12g

G1 vs CMS
Процент таймаутов



Intel® Xeon® Processor E3-1246 v3 @ 3.50Ghz / Oracle JDK 1.8.0_102 / -Xmx12g

Что мы узнали

- Сигаретный™ метод оценки и выбора GC:
  - Оценивать суммарную длительность нежелательных пауз и общее время проведенное в Stop The World
  - Сопоставлять результаты с независимыми измерениями производительности
- Настроенный CMS весьма неплох

План

- Цели
- Предварительные условия
- Как мы тестировали
- Испытание 1. Статья на habrhabr.ru
- **Испытание 2. Миграция на быстрый сервер**
- Испытание 3. Тонкая настройка G1
- Испытание 4. Взгляд в будущее. Java 9
- Итоги

Испытание 2
Миграция на быстрый сервер

- 20 ядер – больше в 5 раз
- 128 Gb RAM
- QPS увеличен в 4 раза

Испытание 2
Миграция на быстрый сервер

- 20 ядер – больше в 5 раз
- 128 Gb RAM
- QPS увеличен в 4 раза
- Прежние настройки GC не подходят

Испытание 2
Миграция на быстрый сервер

- 20 ядер – больше в 5 раз
- 128 Gb RAM
- QPS увеличен в 4 раза
- Прежние настройки GC не подходят
- Настроили **-Xmx64g**
- PROFFIT

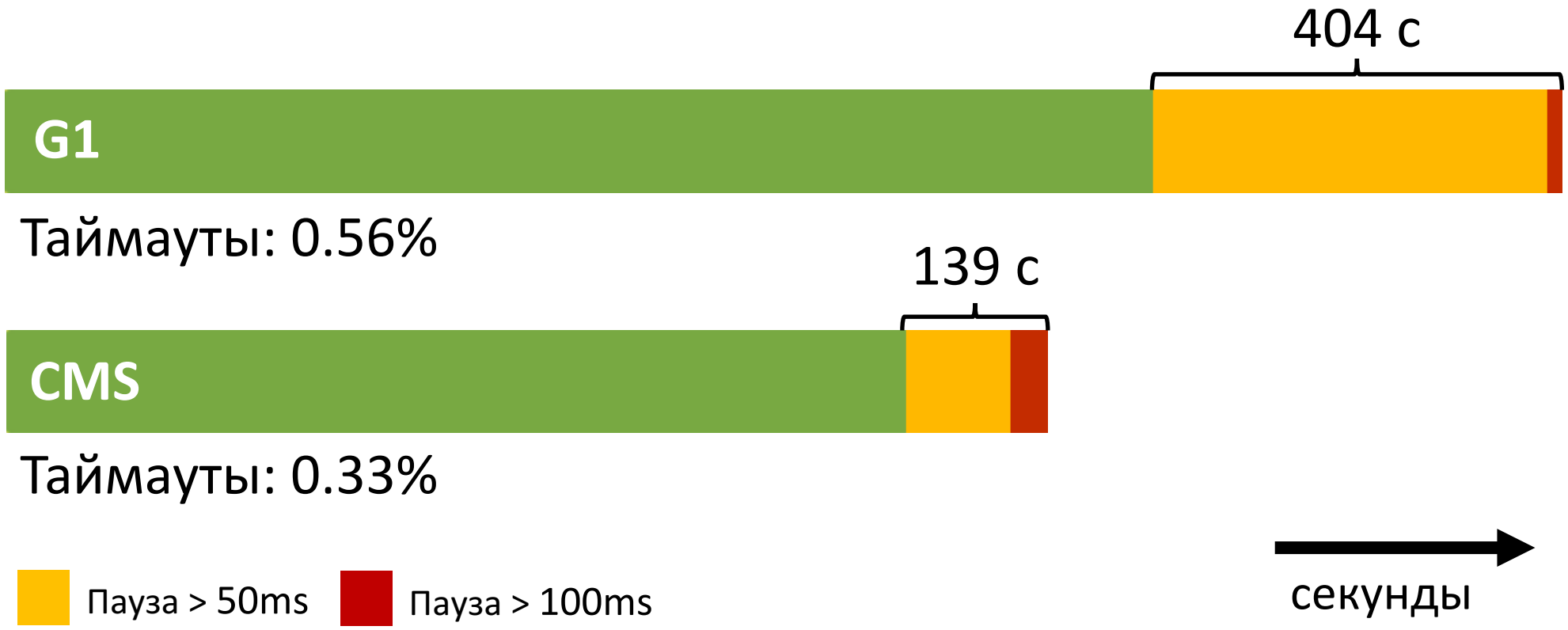
Параметры CMS
Отличия от Испытания 1

Для 64 Gb heap:

-Xmn=12000m

-XX:ParGCCardsPerStrideChunk=32768

G1 vs CMS 64Gb Heap
Stop The World и таймауты



2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz / Oracle JDK 1.8.0_102 / -Xmx64g

Выбор оптимального размера heap

- Working set size ~5Gb
- Подбирали итеративно.
  Остановились на варианте 24Gb

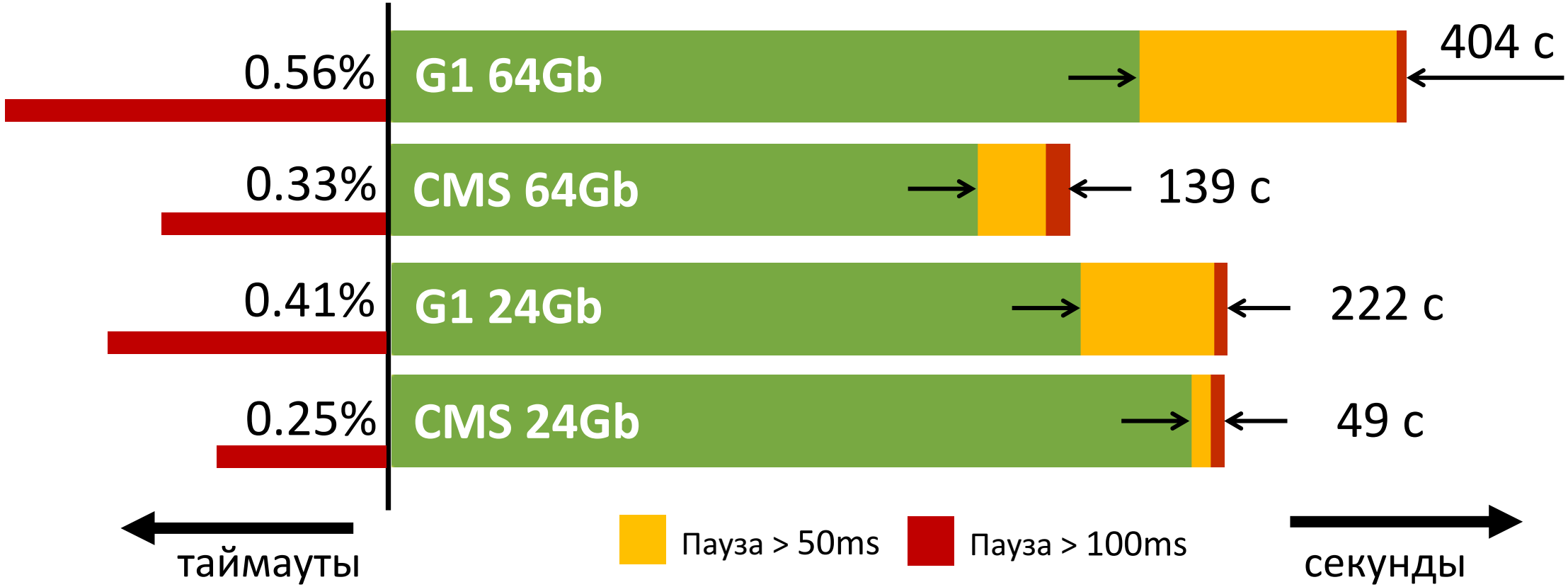
Параметры CMS
Отличия от Испытания 1

Для 24 Gb heap:

-Xmn=4500m

-XX:ParGCCardsPerStrideChunk=16384

Выбор оптимального размера heap



2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz / Oracle JDK 1.8.0_102

Что мы узнали

- Проверили сигаретный™ метод выбора GC на новом сервере, в новых условиях
- Оценили влияние размера heap процент таймаутов

План

- Цели
- Предварительные условия
- Как мы тестировали
- Испытание 1. Статья на habrhabr.ru
- Испытание 2. Миграция на быстрый сервер
- **Испытание 3. Тонкая настройка G1**
- Испытание 4. Взгляд в будущее. Java 9
- Итоги

Тонкая настройка G1

- В качестве базы использовалась конфигурация с heap 64Gb

Тонкая настройка G1

- Garbage First Garbage Collector Tuning *by Monica Beckwith*: http://www.oracle.com/technetwork/articles/java/g1gc-1984535.html

Тонкая настройка G1

-XX:InitiatingHeapOccupancyPercent=45

порог общей занятости Heap, после которого запускается сборка Old Generation

(**IHOP)**

Тонкая настройка G1
Сборка Old Generation

Mixed GC

Нелинейный рост

2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz / Oracle JDK 1.8.0_102

Mixed Garbage Collection 1st

1. [G1Ergonomics (Mixed GCs) start mixed GCs, reason: candidate old regions available, candidate old regions: **893** regions, reclaimable: 28033685448 bytes (40.79 %), threshold: 5.00 %]
2. [G1Ergonomics (CSet Construction) add young regions to CSet, **eden: 91** regions, survivors: 11 regions, **predicted young region time**: **15.28** ms]
3. [G1Ergonomics (CSet Construction) finish adding old regions to CSet, reason: predicted time is too high, predicted time: 0.45 ms, remaining time: 0.00 ms, **old: 112 regions, min: 112 regions**]
4. [G1Ergonomics (CSet Construction) added **expensive** regions to CSet, reason: old CSet region num not reached min, old: 112 regions, **expensive: 81** regions, min: 112 regions, remaining time: 0.00 ms]
5. **Total: 84 ms**

Mixed Garbage Collection 7th

1. [G1Ergonomics (CSet Construction) add young regions to CSet, **eden: 94** regions, survivors: 8 regions, predicted young region time: **16.23 ms**]
2. [G1Ergonomics (CSet Construction) finish adding old regions to CSet, reason: reclaimable percentage not over threshold, old: 92 regions, max: 205 regions, reclaimable: 3415200120 bytes (4.97 %), threshold: 5.00 %]
3. [G1Ergonomics (CSet Construction) added **expensive** regions to CSet, reason: old CSet region num not reached min, old: **92** regions, **expensive: 87** regions, min: 112 regions, remaining time: 0.00 ms]
4. [G1Ergonomics (Mixed GCs) do not continue mixed GCs, reason: reclaimable percentage not over threshold, candidate old regions: 129 regions, reclaimable: 3415200120 bytes (4.97 %)**, threshold: 5.00 %**]
5. **Total: 279 ms**

Тонкая настройка G1
Параметры

-XX:G1NewSizePercent=2

Минимальный размер young generation в процентах от размера heap (по умолчанию 5)
Требует -XX:+UnlockExperimentalVMOptions

-XX:G1HeapWastePercent=10

Сколько мусора можно оставлять после Mixed сборок (по умолчанию 5)

-XX:G1MixedGCCountTarget=12

Кол-во Mixed коллекций. Влияет на минимальный размер Old CSet. (по умолчанию 8)

/ Oracle JDK 1.8.0_102

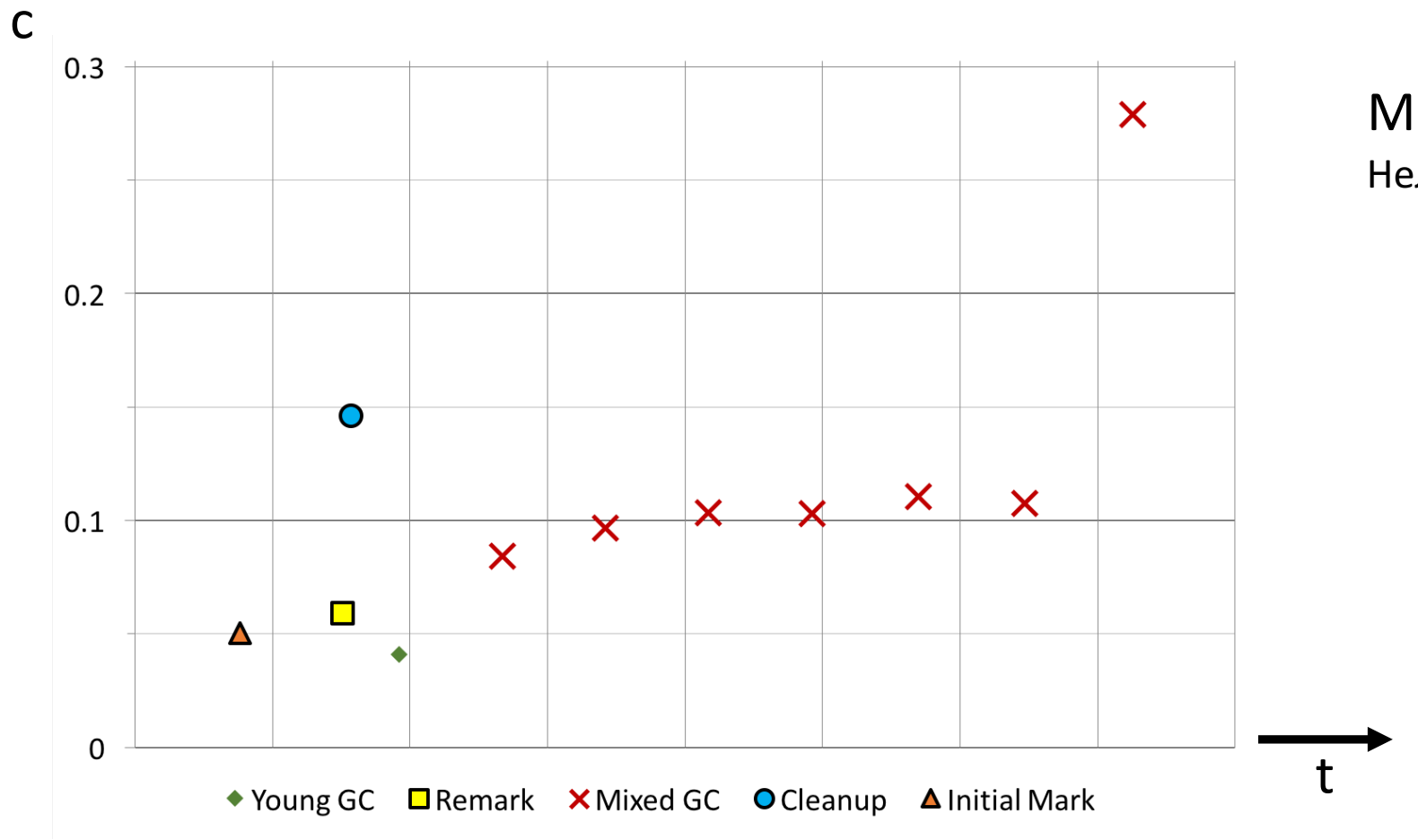
Tuned Mixed Garbage Collection 1st

1. [G1Ergonomics (Mixed GCs) start mixed GCs, reason: candidate old regions available, candidate old regions: **870** regions, reclaimable: 27550635048 bytes (40.09 %), threshold: **10.00 %**]
2. [G1Ergonomics (CSet Construction) add young regions to CSet, eden: **34** regions, survivors: 6 regions, predicted young region time: **8.92 ms**]
3. [G1Ergonomics (CSet Construction) finish adding old regions to CSet, reason: predicted time is too high, predicted time: 0.20 ms, remaining time: 0.00 ms, old: 73 regions, min: 73 regions]
4. [G1Ergonomics (CSet Construction) added expensive regions to CSet, reason: old CSet region num not reached min, old: **73** regions, **expensive: 2** regions, min: 73 regions, remaining time: 0.00 ms]
5. **Total: 34 ms**

Mixed Garbage Collection 9th

1. [G1Ergonomics (CSet Construction) add young regions to CSet, eden: **35** regions, survivors: 5 regions, predicted young region time: **14.72 ms**]
2. [G1Ergonomics (CSet Construction) finish adding old regions to CSet, reason: reclaimable percentage not over threshold, old: 49 regions, max: 205 regions, reclaimable: 6847488064 bytes (9.96 %), threshold: 10.00 %]
3. [G1Ergonomics (CSet Construction) added expensive regions to CSet, reason: old CSet region num not reached min, old: **49** regions, **expensive: 46** regions, min: 73 regions, remaining time: 0.00 ms]
4. [G1Ergonomics (Mixed GCs) do not continue mixed GCs, reason: reclaimable percentage not over threshold, candidate old regions: 237 regions, reclaimable: 6847488064 bytes (9.96 %), threshold: 10.00 %]
5. **Total: 138 ms**

Тонкая настройка G1. До

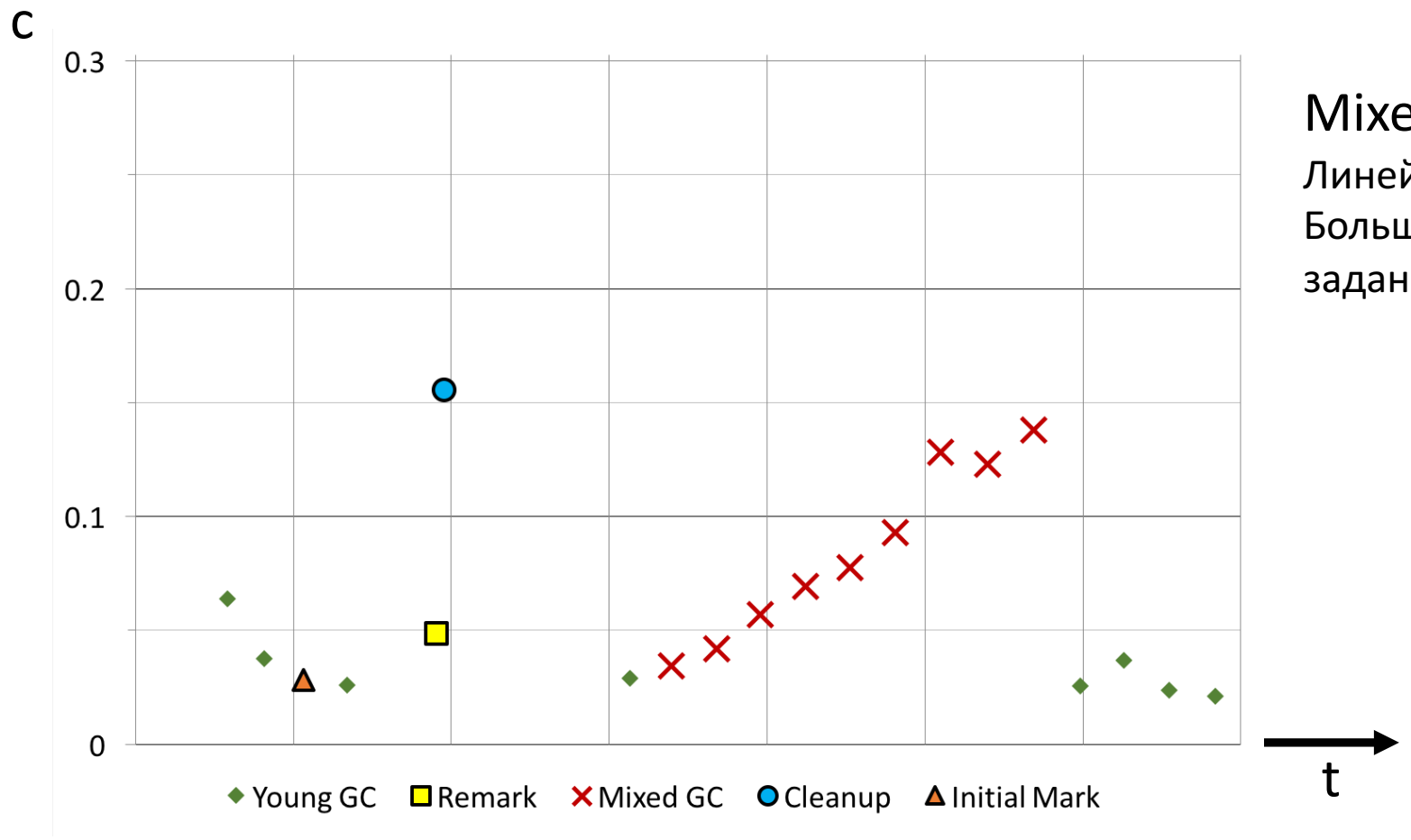


Mixed GC

Нелинейный рост

2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz / Oracle JDK 1.8.0_102

Тонкая настройка G1. После



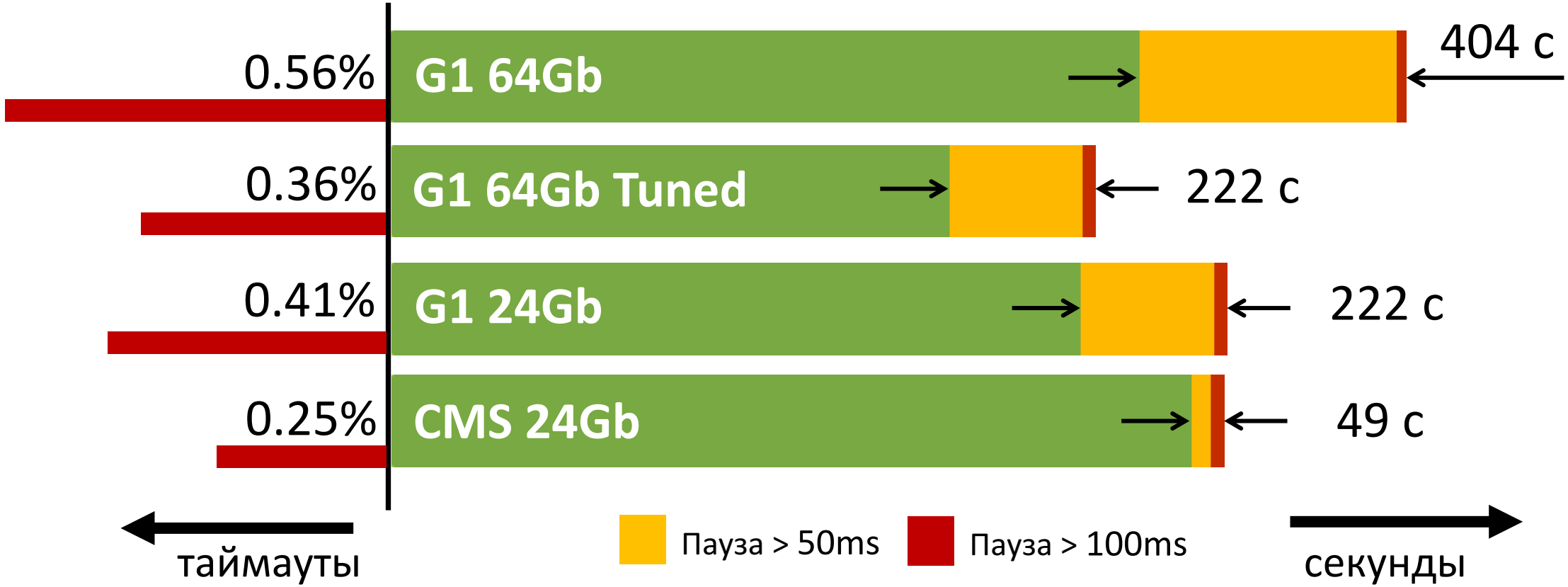
Mixed GC

Линейный рост длительности

Больше пауз укладывается в заданный лимит

2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz / Oracle JDK 1.8.0_102

Тонкая настройка G1



2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz / Oracle JDK 1.8.0_102

Что еще можно попробовать

-XX:G1MixedGCLiveThresholdPercent=65

Максимальная доля живых объектов в Old регионе, чтобы он мог рассматриваться в качестве кандидата для сборки в Mixed GC.

Требует -XX:+UnlockExperimentalVMOptions

Что мы узнали

- У G1 большой потенциал по улучшению показателей за счет тонкой настройки
- С настройками по умолчанию может работать не оптимально

План

- Цели
- Предварительные условия
- Как мы тестировали
- Испытание 1. Статья на habrhabr.ru
- Испытание 2. Миграция на быстрый сервер
- Испытание 3. Тонкая настройка G1
- **Испытание 4. Взгляд в будущее. Java 9**
- Итоги

Испытание 4. Взгляд в будущее.
Java 9

Испытание 4. Взгляд в будущее.
Java 9

Java 9

- Garbage First – GC по умолчанию

Java 9

- Garbage First – GC по умолчанию
- -XX:+G1UseAdaptiveIHOP JDK-8136677

  (Adaptive -XX:InitiatingHeapOccupancyPercent)

Java 9

- Garbage First – GC по умолчанию
- -XX:+G1UseAdaptiveIHOP JDK-8136677

  (Adaptive -XX:InitiatingHeapOccupancyPercent)

https://bugs.openjdk.java.net

labels != testbug and Subcomponent = gc AND (status = closed or status = resolved) AND resolution = fixed and type !=Backport and summary!~ "test*" and summary!~ "log*" and summary!~ "crash*" and summary!~ "fail*" ORDER BY resolved DESC

Unified GC Logging

http://openjdk.java.net/jeps/271

https://bugs.openjdk.java.net/browse/JDK-8059805

Old GC Logging

-XX:+PrintGC
-XX:+PrintGCDetails
-Xloggc:/path/gc.log
-XX:+PrintAdaptiveSizePolicy
-XX:+PrintReferenceGC
-XX:+PrintGCTimeStamps
-XX:+PrintTenuringDistribution
-XX:+Print…

Unified GC Logging

-XX:+PrintGC
-XX:+PrintGCDetails
-Xloggc:/path/gc.log
-XX:+PrintAdaptiveSizePolicy
-XX:+PrintReferenceGC
-XX:+PrintGCTimeStamps
-XX:+PrintTenuringDistribution
-XX:+Print…

-Xlog:gc*=info:file=/path/gc.log

Unified GC Logging

~~-XX:+PrintGC~~
~~-XX:+PrintGCDetails~~
~~-Xloggc:/path/gc.log~~
~~-XX:+PrintAdaptiveSizePolicy~~
~~-XX:+PrintReferenceGC~~
~~-XX:+PrintGCTimeStamps~~
~~-XX:+PrintTenuringDistribution~~
-XX:+Print…

-Xlog:gc*=info:file=/path/gc.log

Unified GC Logging

-Xlog:gc*=info:file=/path/gc.log

```
[117.941s][info][gc,start      ] GC(15) Pause Young (G1 Evacuation Pause) (117.941s)
[117.941s][info][gc,task       ] GC(15) GC Workers: using 28 out of 28
...
[118.005s][info][gc,task       ] GC(15) GC Workers: using 3 out of 28
[118.005s][info][gc,task       ] GC(15) GC Workers: using 28 out of 28
[118.006s][info][gc,mmu        ] GC(15) MMU target violated: 41.0ms (40.0ms/41.0ms)
[118.006s][info][gc,phases     ] GC(15)   Evacuate Collection Set: 51.5ms
[118.006s][info][gc,phases     ] GC(15)   Code Roots: 0.2ms
[118.006s][info][gc,phases     ] GC(15)   Clear Card Table: 0.3ms
[118.006s][info][gc,phases     ] GC(15)   Expand Heap After Collection: 0.0ms
[118.006s][info][gc,phases     ] GC(15)   Free Collection Set: 0.4ms
[118.006s][info][gc,phases     ] GC(15)   Merge Per-Thread State: 0.1ms
[118.006s][info][gc,phases     ] GC(15)   Other: 12.6ms
[118.006s][info][gc,heap       ] GC(15) Eden regions: 95->0(94)
[118.006s][info][gc,heap       ] GC(15) Survivor regions: 7->8(13)
[118.006s][info][gc,heap       ] GC(15) Old regions: 55->55
[118.006s][info][gc,heap       ] GC(15) Humongous regions: 41->34
[118.006s][info][gc,metaspace  ] GC(15) Metaspace: 76582K->76582K(86016K)
[118.006s][info][gc            ] GC(15) Pause Young (G1 Evacuation Pause) 6314M->3085M(65536M) (117.941s, 118.006s) 65.121ms
[118.006s][info][gc,cpu        ] GC(15) User=1.22s Sys=0.02s Real=0.07s
[121.245s][info][gc,start      ] GC(16) Pause Young (G1 Evacuation Pause) (121.245s)
```

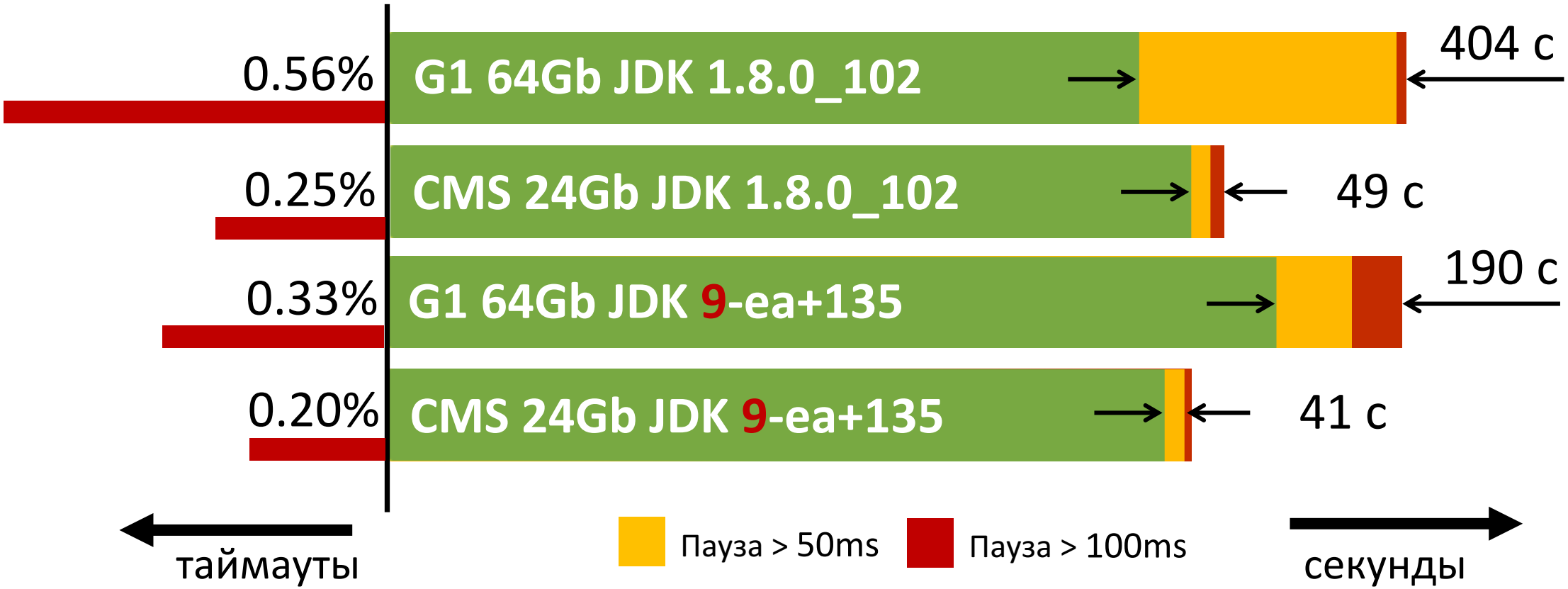
Зачем тестировали

- Вся кодовая база на Java 8
- Планируется быстрое освоение Java 9

Зачем тестировали

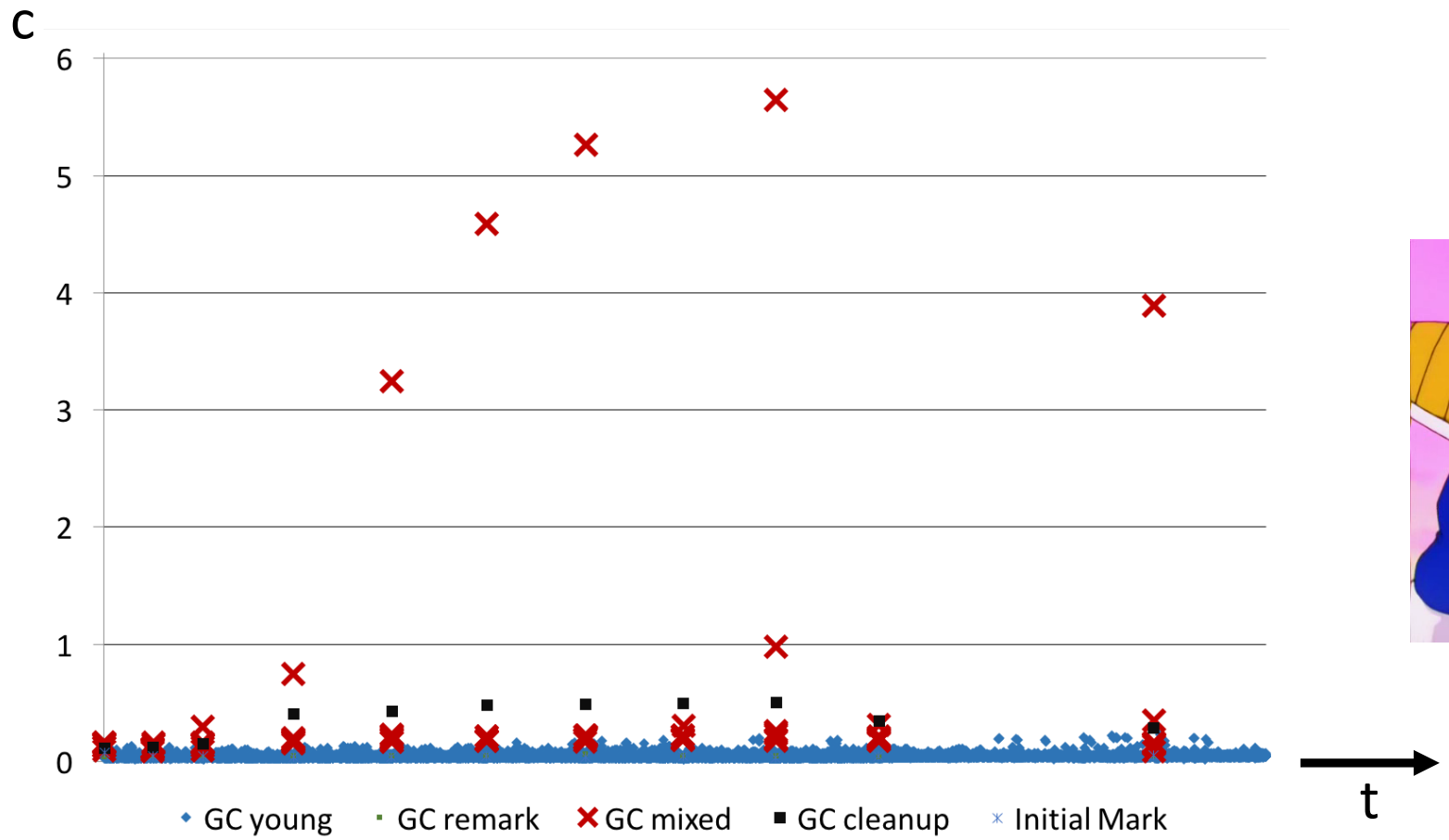
- Вся кодовая база на Java 8
- Планируется быстрое освоение Java 9
- Отказоустойчивая архитектура

Java 8 vs Java 9

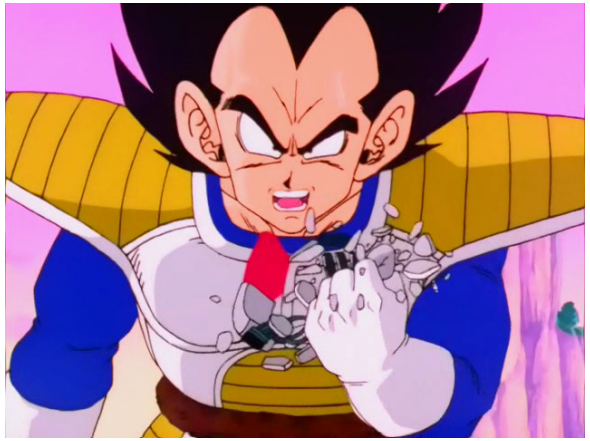


2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz / -Xmx24g

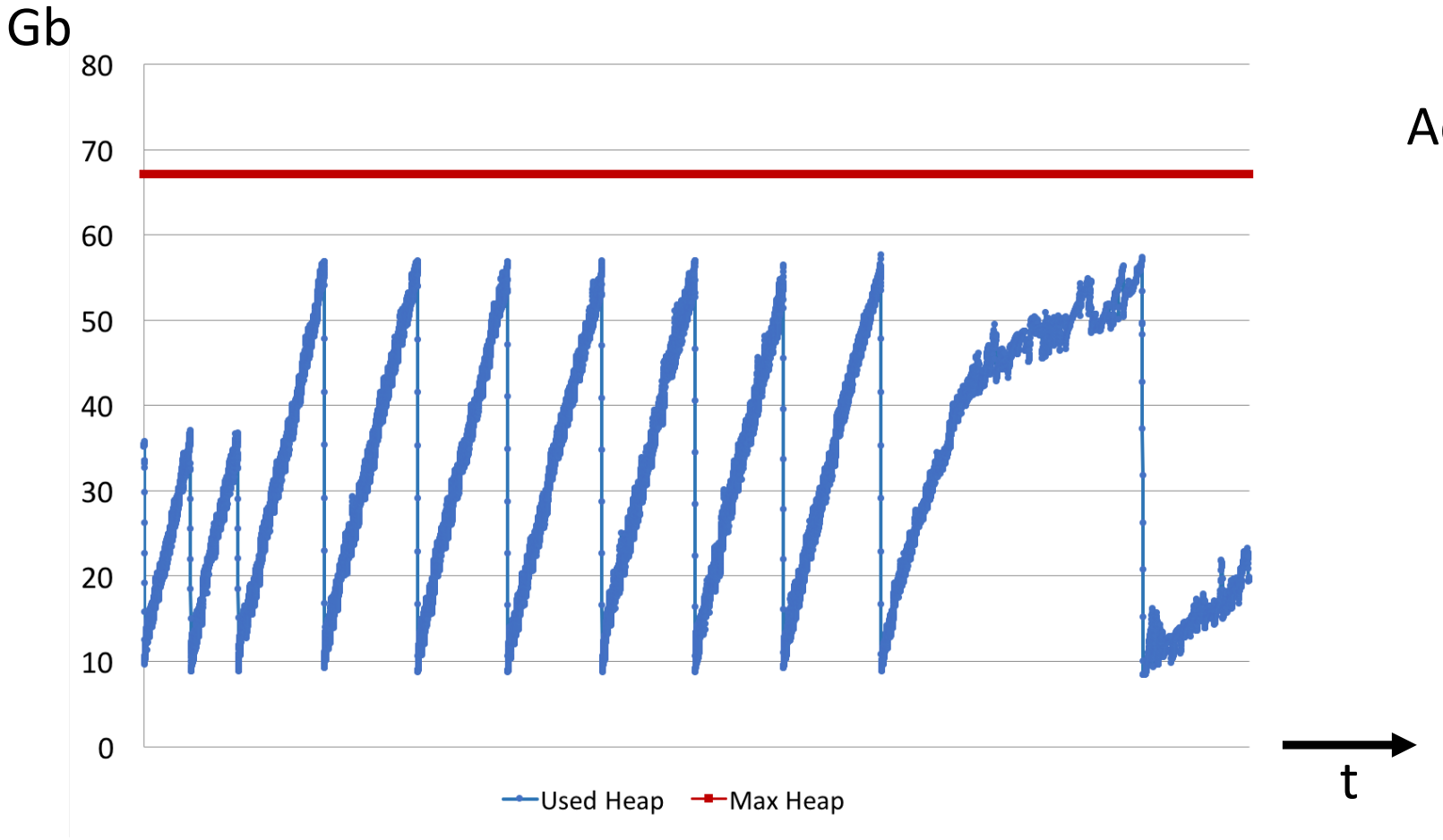
Java 9 G1. Странности



Mixed GC
Over 5000 !!



Java 9 G1. Странности



Adaptive IHOP = 80%

Java 9. Странности

- Bug JDK-8166500 Adaptive sizing for IHOP causes excessively long mixed GC pauses

Java 9. Странности

- Bug ~~JDK-8166500~~ Adaptive sizing for IHOP causes excessively long mixed GC pauses
- Bug JDK-8159697 Adaptive G1HeapWastePercent, G1MixedGCLiveThresholdPercent, G1OldCSetRegionThresholdPercent, and G1MixedGCLiveThresholdPercent and G1MixedGCCountTarget

Java 9. Странности

- Bug ~~JDK-8166500~~ Adaptive sizing for IHOP causes excessively long mixed GC pauses
- Bug JDK-8159697 Adaptive G1HeapWastePercent, G1MixedGCLiveThresholdPercent, G1OldCSetRegionThresholdPercent, and G1MixedGCLiveThresholdPercent and G1MixedGCCountTarget
  (Fix version: **10**)

Java 9. Странности

- Bug ~~JDK-8166500~~ Adaptive sizing for IHOP causes excessively long mixed GC pauses
- Bug JDK-8159697 Adaptive G1HeapWastePercent, G1MixedGCLiveThresholdPercent, G1OldCSetRegionThresholdPercent, and G1MixedGCLiveThresholdPercent and G1MixedGCCountTarget (Fix version: **10**)
- Mailing list: hotspot-gc-use http://mail.openjdk.java.net/mailman/listinfo/hotspot-gc-use

Java 9. Странности

- Bug ~~JDK-8166500~~ Adaptive sizing for IHOP causes excessively long mixed GC pauses
- Bug JDK-8159697 Adaptive G1HeapWastePercent, G1MixedGCLiveThresholdPercent, G1OldCSetRegionThresholdPercent, and G1MixedGCLiveThresholdPercent and G1MixedGCCountTarget (Fix version: **10**)
- Mailing list: hotspot-gc-use http://mail.openjdk.java.net/mailman/listinfo/hotspot-gc-use
- -XX:MaxGCPauseMillis=40 не простая цель

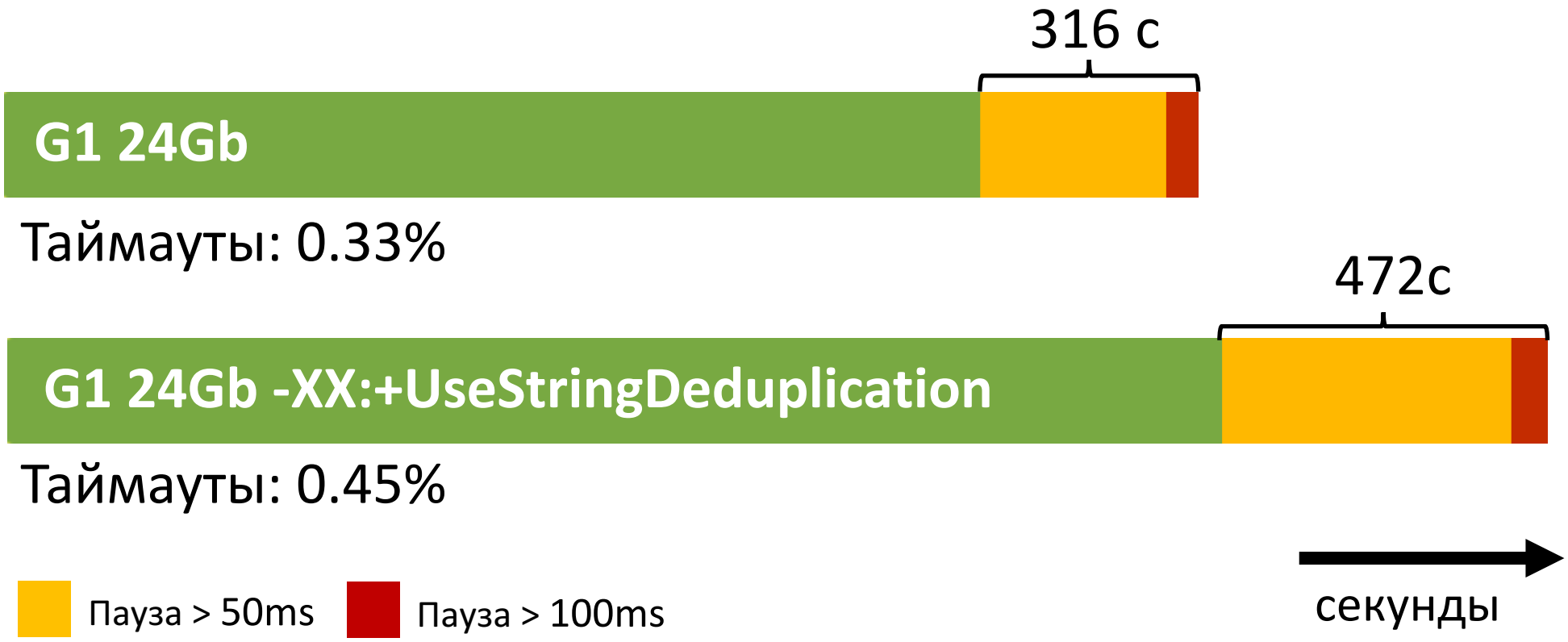
Bonus Stage

- String Deduplication
- Java 9 без Compact Strings

Java 9 String Deduplication

- JDK-8158871 Long response times with G1 and StringDeduplication (Java 1.9.0 b127 & Java 1.8.0_112)

Java 9 String Deduplication



2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz / JDK 9-ea+135 /-Xmx24g

Java 9 String Deduplication

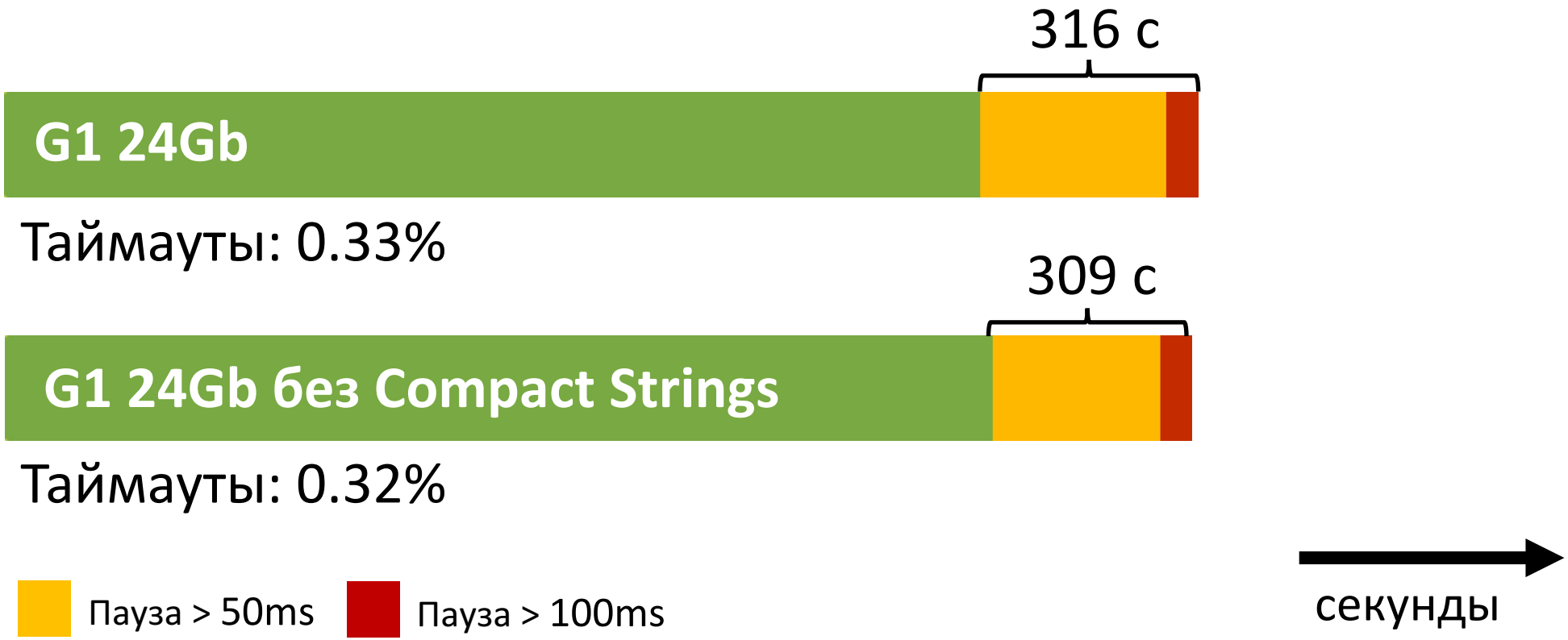
	Total heap allocation
G1 24Gb	164.2 TB
G1 24Gb -XX:+UseStringDeduplication	164.2 TB

Java 9 String Deduplication

	Total heap allocation
G1 24Gb	164.2 TB
G1 24Gb -XX:+UseStringDeduplication	164.2 TB

Из логов:
Дедуплицировано: **19.2 Gb**
Concurrent time: **176.5 s**

Java 9 без Compact Strings
-XX:-CompactStrings



2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz / JDK 9-ea+135 /-Xmx24g

Java 9 без Compact Strings
-XX:-CompactStrings

	Total heap allocation
G1 24Gb	164.2 TB
G1 24Gb без Compact Strings	167.6 TB

Compact Strings сэкономил 2%

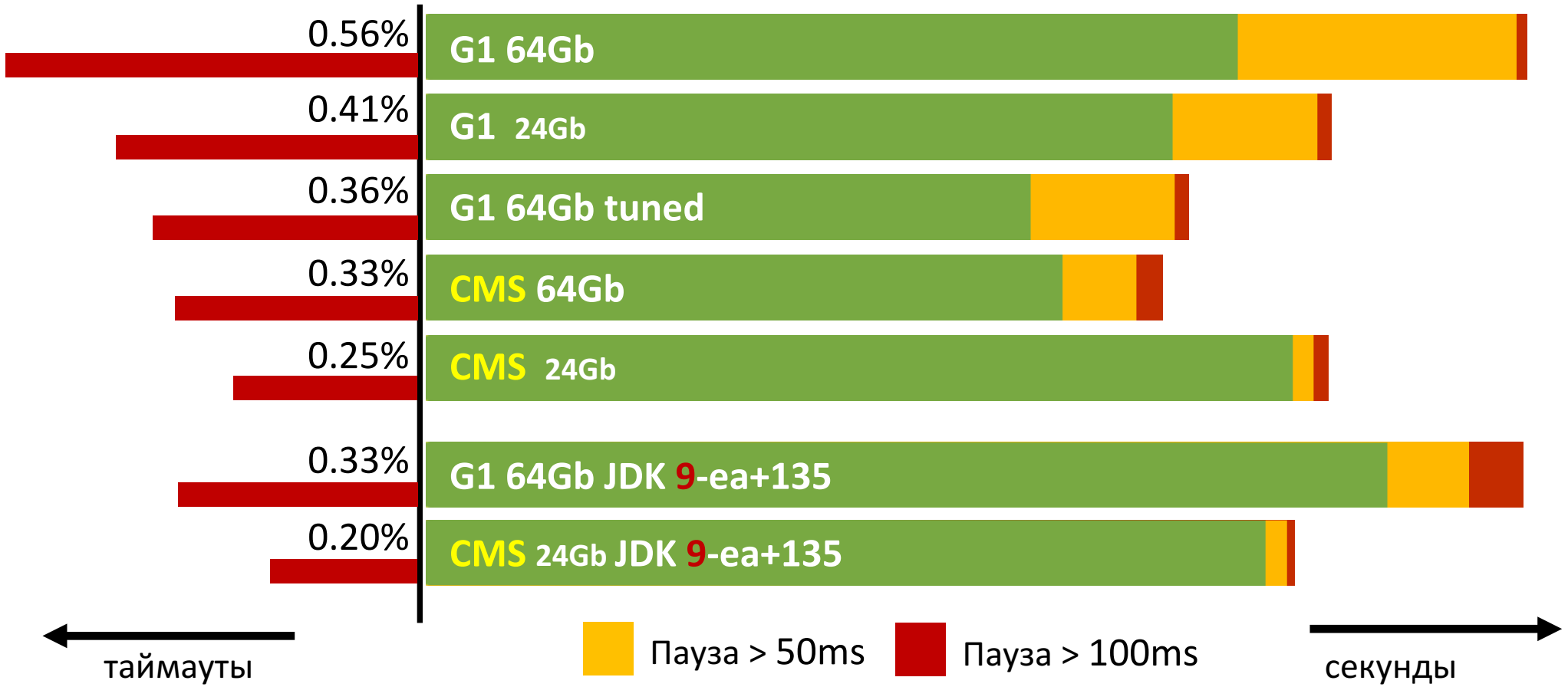
Что мы узнали

- Java 9 быстрее
- Есть странности в работе G1
- Early Access полезен. Баги надо репортить
- String Deduplication стал лучше, но нам не нужен

План

- Цели
- Предварительные условия
- Как мы тестировали
- Испытание 1. Статья на habrhabr.ru
- Испытание 2. Миграция на быстрый сервер
- Испытание 3. Тонкая настройка G1
- Испытание 4. Взгляд в будущее. Java 9
- **Итоги**

Итоги



2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz / Oracle JDK 1.8.0_102 / JDK 9-ea+135

Итоги

	G1 64 Gb	G1 24Gb	G1 64Gb Tuned	CMS 64Gb	CMS 24Gb	G1 64Gb JDK 9	CMS 24Gb JDK 9
ms per second	20.12	16.55	13.95	13.47	16.50	20.0559	15.89
max pause	228.11	394.87	240.67	644.84	354.06	4909.17	410.07
avg pause	45.28	42.59	41.63	37.92	23.50	39.7046	23.19
99th	78.97	72.86	90.24	83.64	50.96	90.5704	50.28
avg interval	2204.97	2530.65	2943.36	2777.74	1400.76	1939.99	1436.43
>50ms sum	404_292	221_711	221_610	139_342	49_393	189_862	40_739
total stw time	1_537_327	1_264_169	1_065_450	1_027_559	1_259_676	1_532_159	1_213_170
timeouts	0.56%	0.41%	0.36%	0.33%	0.25%	0.33%	0.20%

2 x Intel® Xeon® Processor E5-2650 v3 @ 2.30Ghz / Oracle JDK 1.8.0_102 / JDK 9-ea+135

Итоги

- Метод оценки и выбора GC с примерами
- Оценили потенциал G1 по настройке
- Заглянули в будущее
- Early Access бывает полезен

Disclaimer

- Ваши результаты могут не совпадать с нашими

Вопросы

Нурисламов Ильдар
Adserver Team Leader, GetIntent
https://twitter.com/absorbb
inurislamov@getintent.com
absorbb@gmail.com
skype: absorbb