容器里面乾坤大: 采用容器部署应用的性能考虑

Memory-related Performance Pitfalls and Solutions for Linux Cgroup's based Deployments

庄振运

Outline

- ❑ Introduction
- ❑ Memory related performance pitfalls
- ❑ Strategies
- ❑ Discussions
- ❑ Conclusion

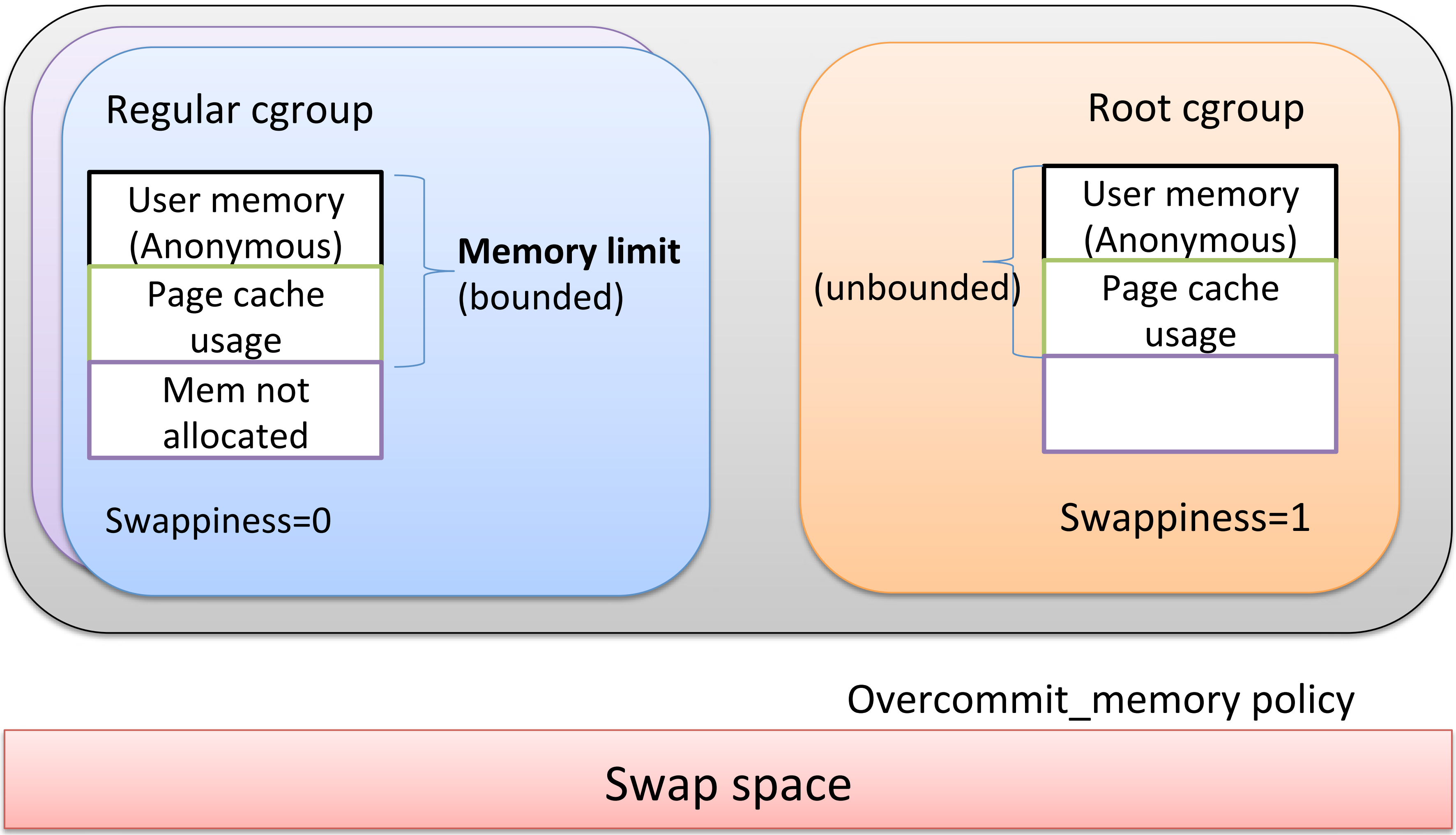
Self Introduction

- LinkedIn or Microsoft?
  - N/A
- Performance engineering
  - Applications (HTTP, P2P, Hadoop, streaming, etc.)
  - Java (JVM, GC, etc.)
  - VM/Container(cgroups, etc.)
  - Linux (Memory management, file system, cpu scheduling, etc.)
  - Networking (Wireless/mobile, TCP/IP, etc.)
  - Storage (HDD, SSD, etc.)
- Other interest
  - Chinese culture (History, Poems, etc.)

Problem context

- Container
  - Linux cgroups, Docker, CoreOS
- New challenges in APM
  - Performance metrics monitoring
  - Deployment concerns
  - Debugging/alerting
- Knowledge sharing, discussions
  - Cgroups, performance, memory
- Blog
  - https://engineering.linkedin.com/blog/2016/08/don_t-let-linux-control-groups-uncontrolled

Technical backgrounds (Cgroups)



Outline

- Introduction
- **Memory related performance pitfalls**
- Strategies
- Discussions
- Conclusion

Memory related performance pitfalls

- ❑Memory is not allocated
- ❑Page cache is part of memory limit, can be evicted by anonymous memory request
- ❑OS can reclaim system-wide page cache
- ❑OS can swap system-wide anonymous memory
- ❑Virtual memory space is not limited

Experiment setup

- Hardware
  - Intel Xeon E5-2680, dual sockets (12 physical cores)
  - 64 GB RAM (NUMA setup)
- OS
  - RHEL (RedHat Linux Enterprise) 7, 3.10.0-327.10.1
  - 16GB swap, swappiness=1 for root, 0 for regular cgroups
- Workload
  - Java application
- Other performance metrics
  - Cgroup stat (swap, rss, page cache), “free”

Pitfall 1: Memory is not allocated (as with VM)

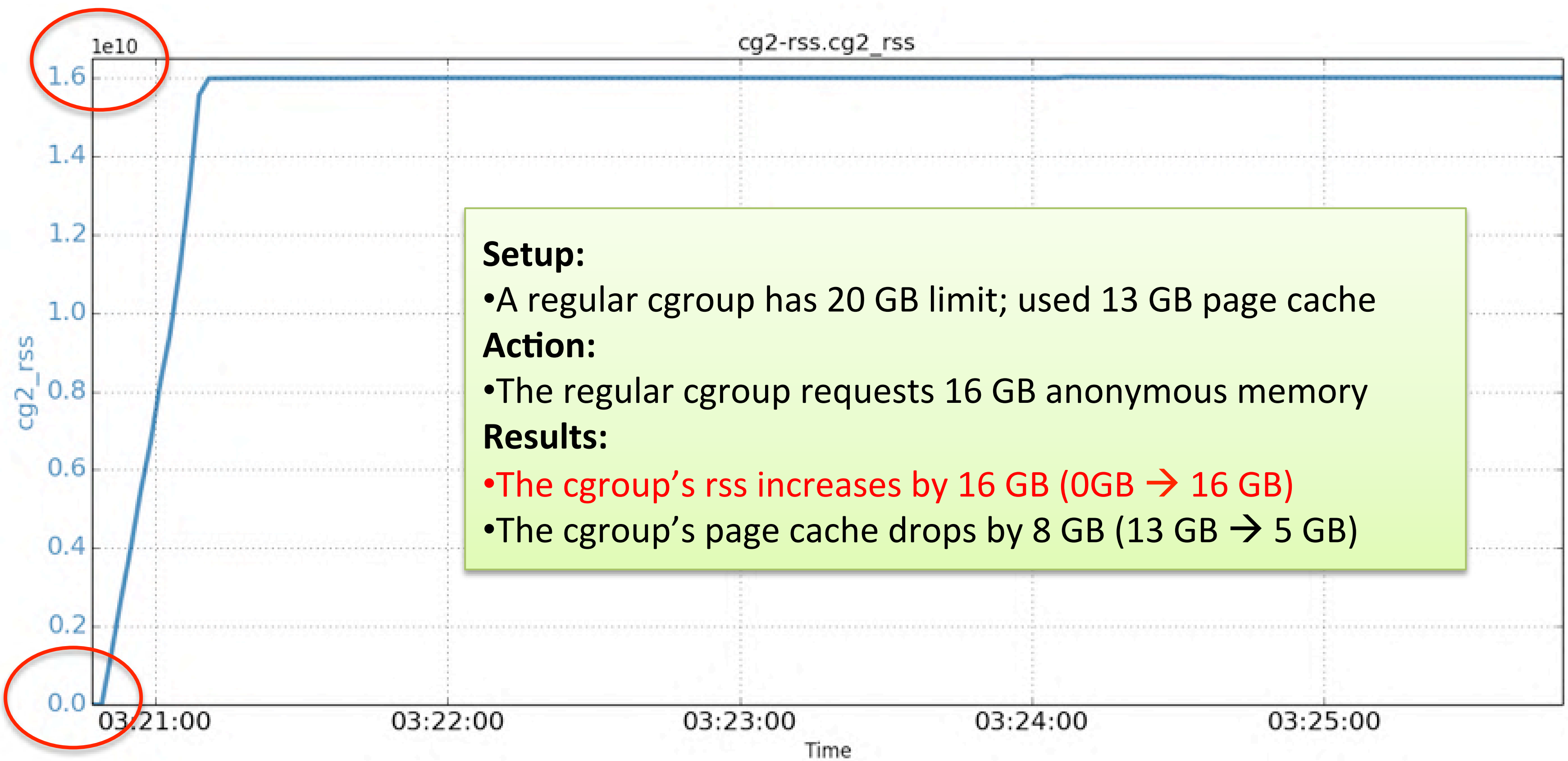
- Memory limit of a cgroup
  - Only upper bound
  - “Use as you go” model
- Memory request from cgroups
  - Free memory
  - OS reclaiming (page cache or swapping)
- Performance when write-backing dirty caches
  - Taking 20 seconds to obtain 16GB of memory
  - Varies depending on the dirty cache size and IO capacity

Pitfall 2: Page cache is part of memory limit

- Memory limit of a cgroup
  - Anonymous memory (user space)
  - Page cache used (Kernel space)
  - Need to estimate footprints of both types
- Anonymous memory requests evicting page cache
  - Insufficient page cache causes under-performing application
  - Write-back IO may affect other cgroups

Pitfall 2: **Page cache is part of memory limit**

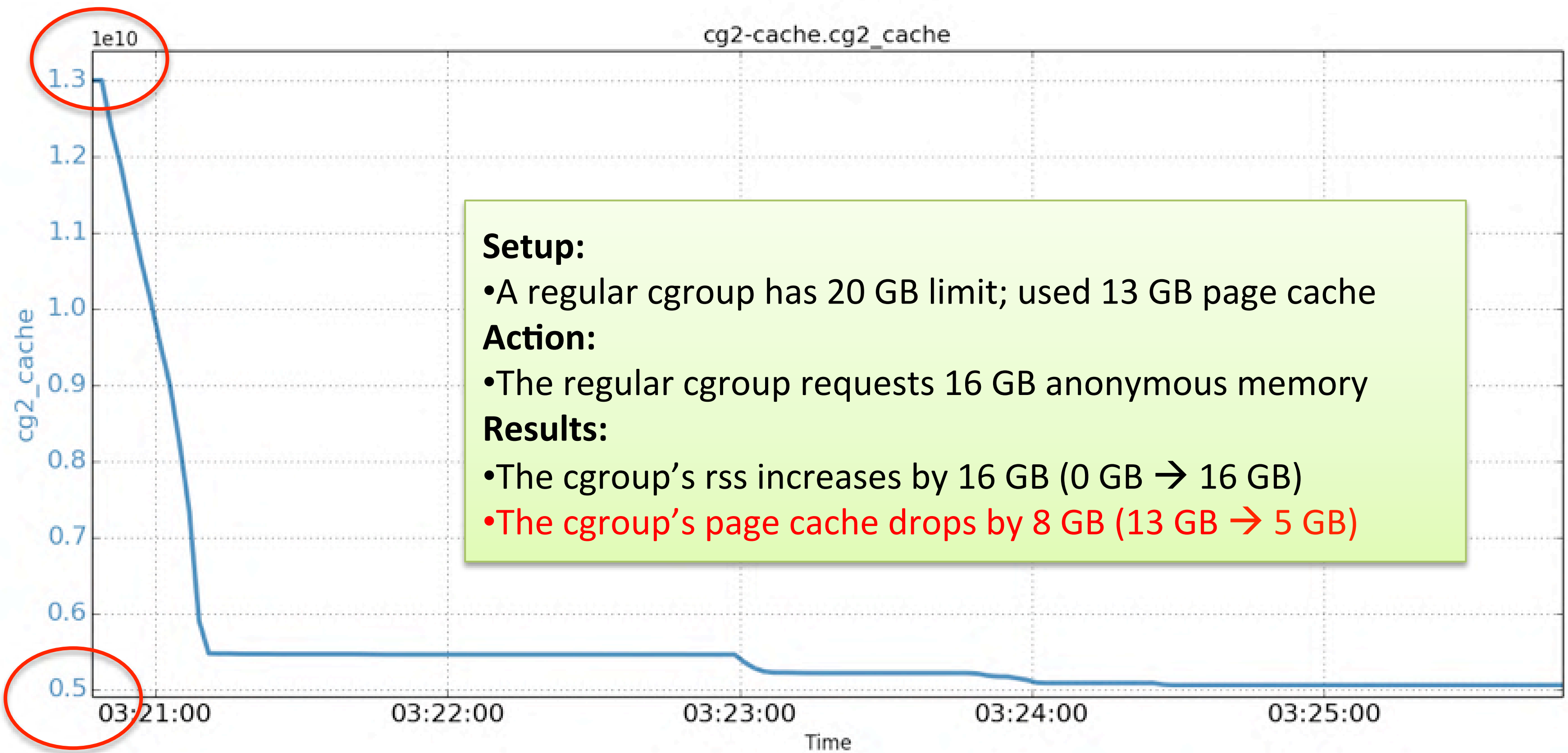
Experiment results (cgroup's rss)



APMCon

Pitfall 2: **Page cache is part of memory limit**

Experiment results (cgroup's page cache)



Setup:

•A regular cgroup has 20 GB limit; used 13 GB page cache

Action:

•The regular cgroup requests 16 GB anonymous memory

Results:

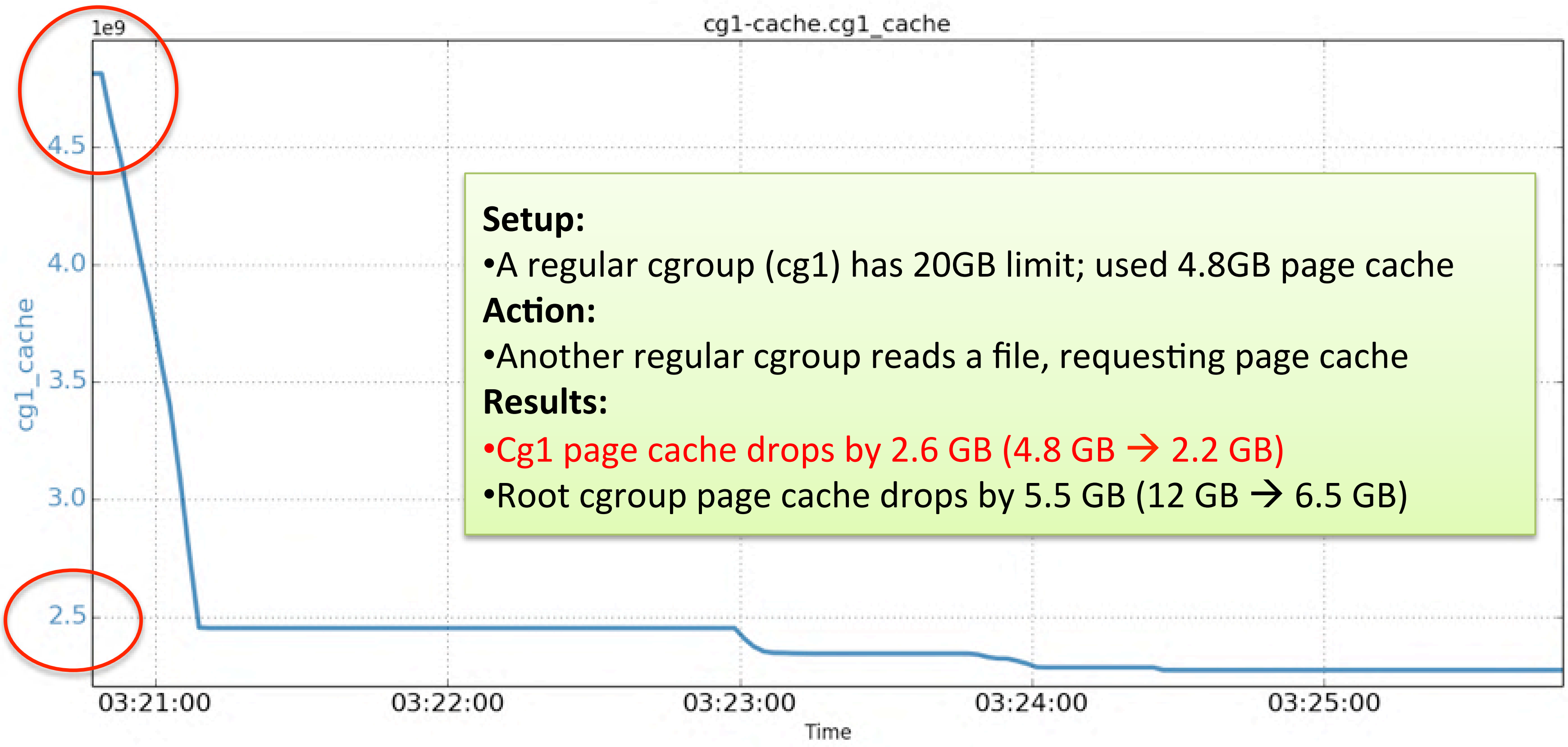
•The cgroup's rss increases by 16 GB (0 GB → 16 GB)

•The cgroup's page cache drops by 8 GB (13 GB → 5 GB)

Pitfall 3: OS can reclaim system-wide page cache

- Page cache used charged to cgroups
  - Anonymous memory + page cache < memory limit
- OS maintains the entire page cache
  - Kernel space
  - Replacement algorithm applies to all pages
  - Does not respect the owners

Pitfall 3: Experiment results (regular cgroup's page cache)



Setup:

•A regular cgroup (cg1) has 20GB limit; used 4.8GB page cache

Action:

•Another regular cgroup reads a file, requesting page cache

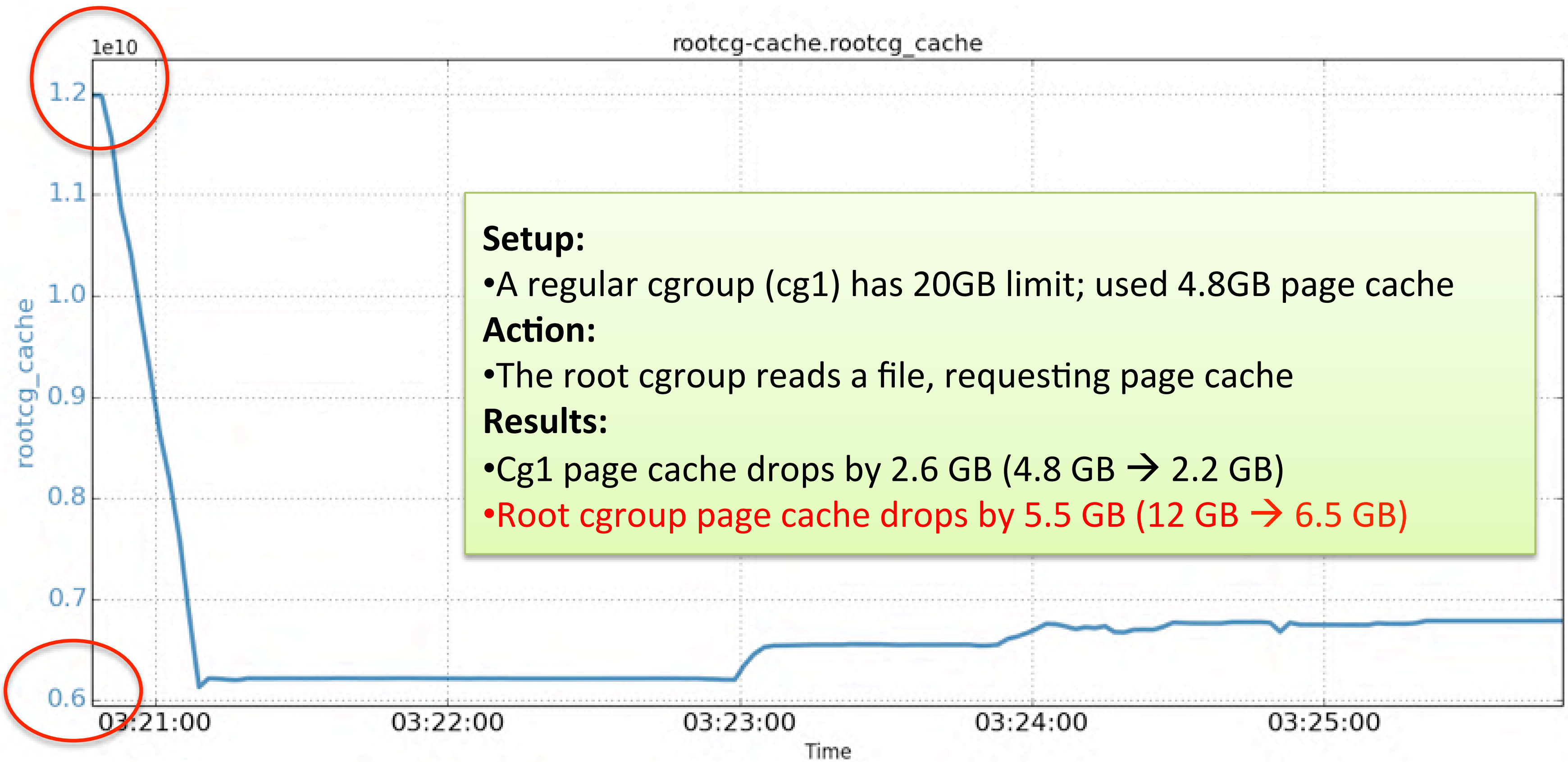
Results:

•Cg1 page cache drops by 2.6 GB (4.8 GB → 2.2 GB)

•Root cgroup page cache drops by 5.5 GB (12 GB → 6.5 GB)

APMCon

Pitfall 3: Experiment results (root cgroup's page cache)



Setup:

•A regular cgroup (cg1) has 20GB limit; used 4.8GB page cache

Action:

•The root cgroup reads a file, requesting page cache

Results:

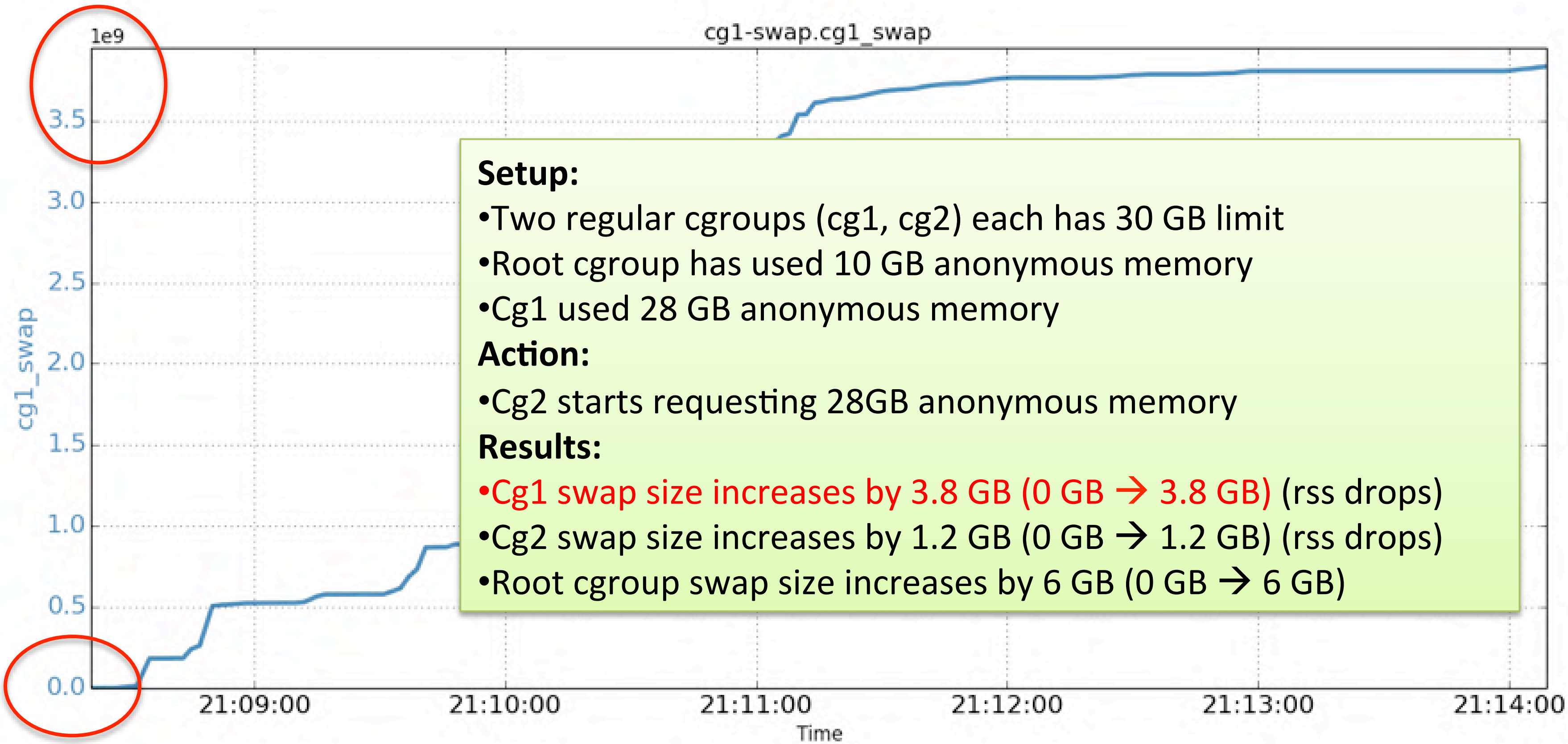
•Cg1 page cache drops by 2.6 GB (4.8 GB → 2.2 GB)

•Root cgroup page cache drops by 5.5 GB (12 GB → 6.5 GB)

Pitfall 4: OS can swap system-wide anonymous memory

- Anonymous memory usage of a cgroup
  - Anonymous memory + page cache < memory limit
  - Swappiness=0 can protect memory from inside requests
- OS controls the swapping mechanism
  - All cgroups share the same swap space
  - OS can swap any anonymous memory pages
  - Does not respect the owners

Pitfall 4: Experiment results (Cg1's swap)



Setup:

- Two regular cgroups (cg1, cg2) each has 30 GB limit
- Root cgroup has used 10 GB anonymous memory
- Cg1 used 28 GB anonymous memory

Action:

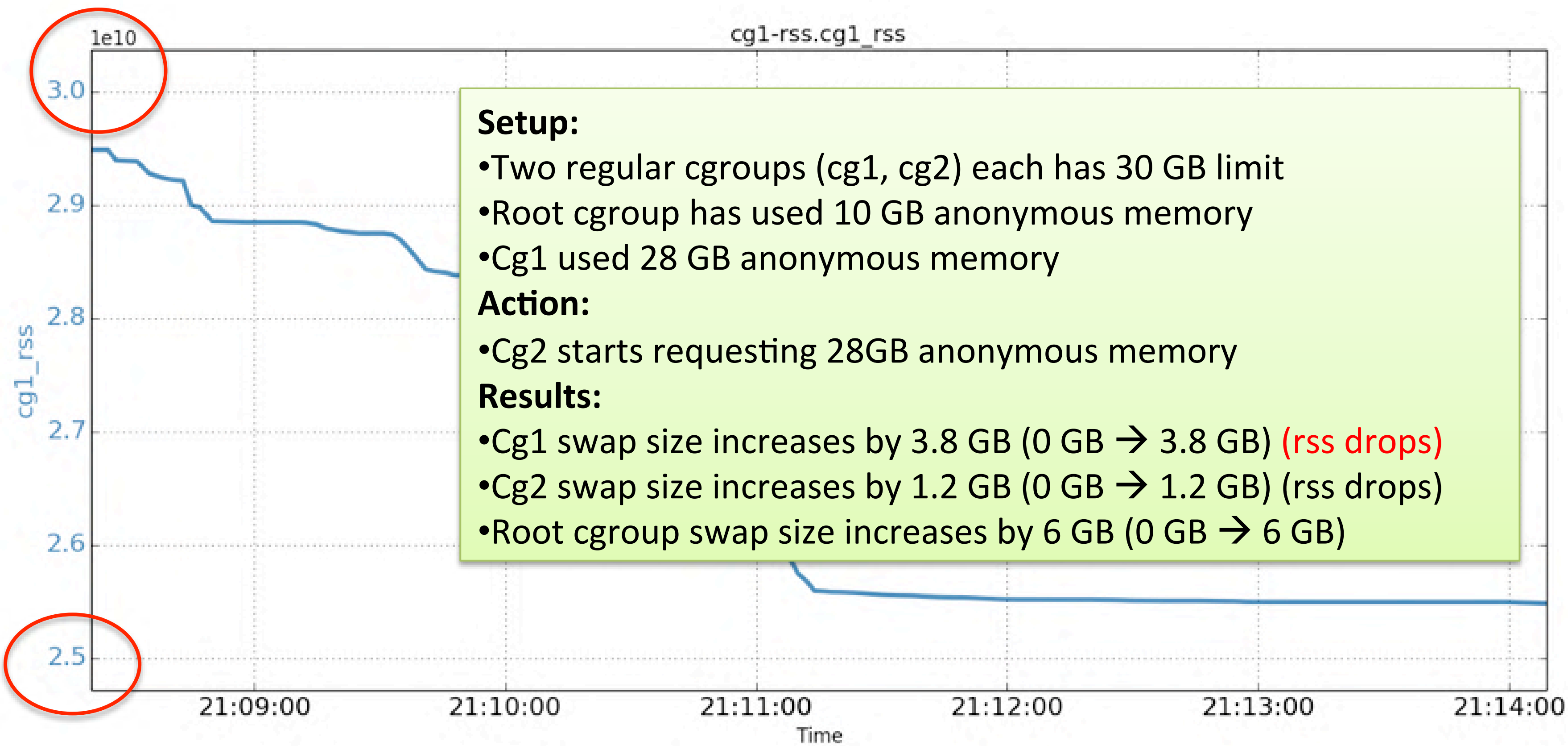
- Cg2 starts requesting 28GB anonymous memory

Results:

- Cg1 swap size increases by 3.8 GB (0 GB → 3.8 GB) (rss drops)
- Cg2 swap size increases by 1.2 GB (0 GB → 1.2 GB) (rss drops)
- Root cgroup swap size increases by 6 GB (0 GB → 6 GB)

APMCon

Pitfall 4: Experiment results (Cg1’s rss)

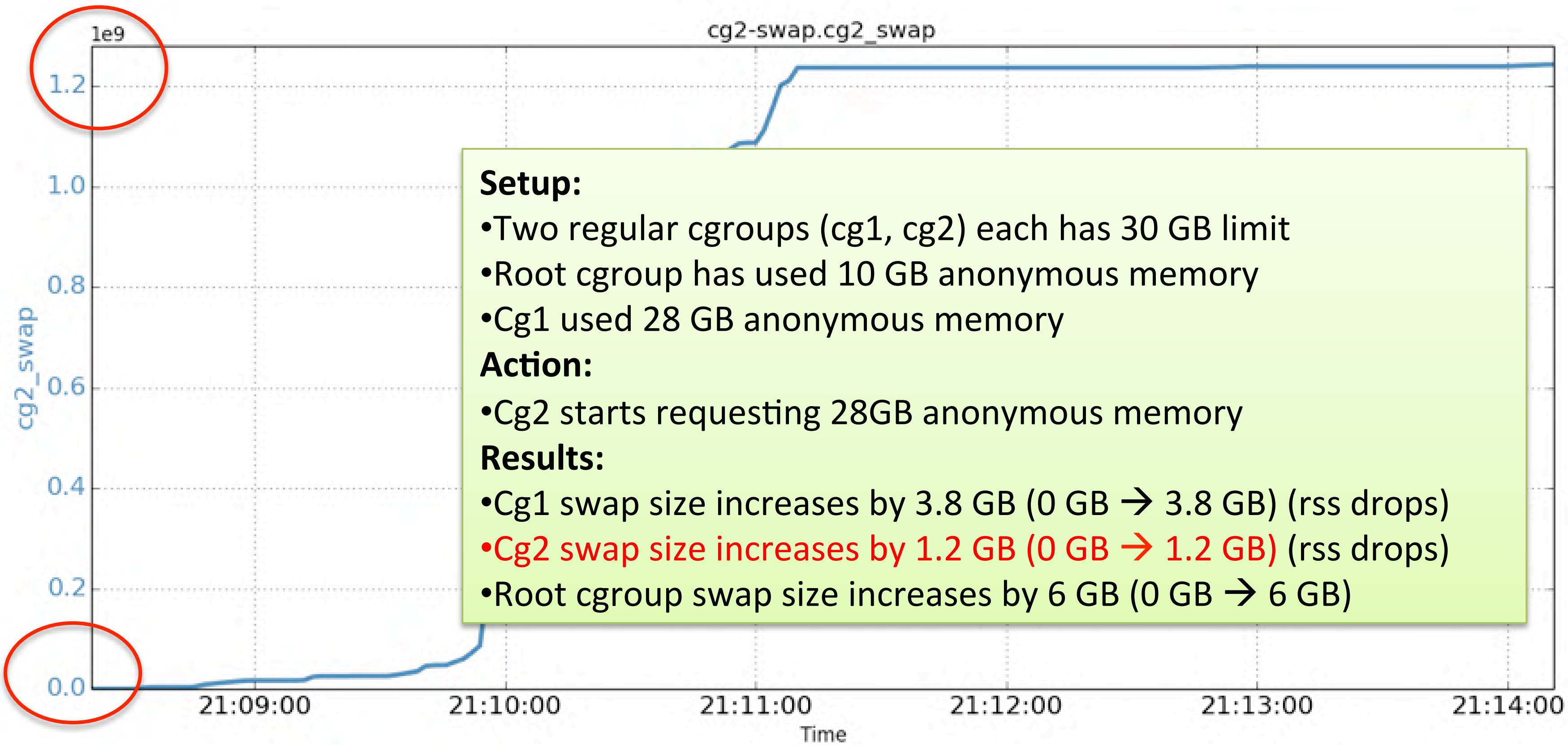


Setup:
- Two regular cgroups (cg1, cg2) each has 30 GB limit
- Root cgroup has used 10 GB anonymous memory
- Cg1 used 28 GB anonymous memory

Action:
- Cg2 starts requesting 28GB anonymous memory

Results:
- Cg1 swap size increases by 3.8 GB (0 GB → 3.8 GB) (rss drops)
- Cg2 swap size increases by 1.2 GB (0 GB → 1.2 GB) (rss drops)
- Root cgroup swap size increases by 6 GB (0 GB → 6 GB)

Pitfall 4: Experiment results (Cg2's swap)



Setup:

- Two regular cgroups (cg1, cg2) each has 30 GB limit
- Root cgroup has used 10 GB anonymous memory
- Cg1 used 28 GB anonymous memory

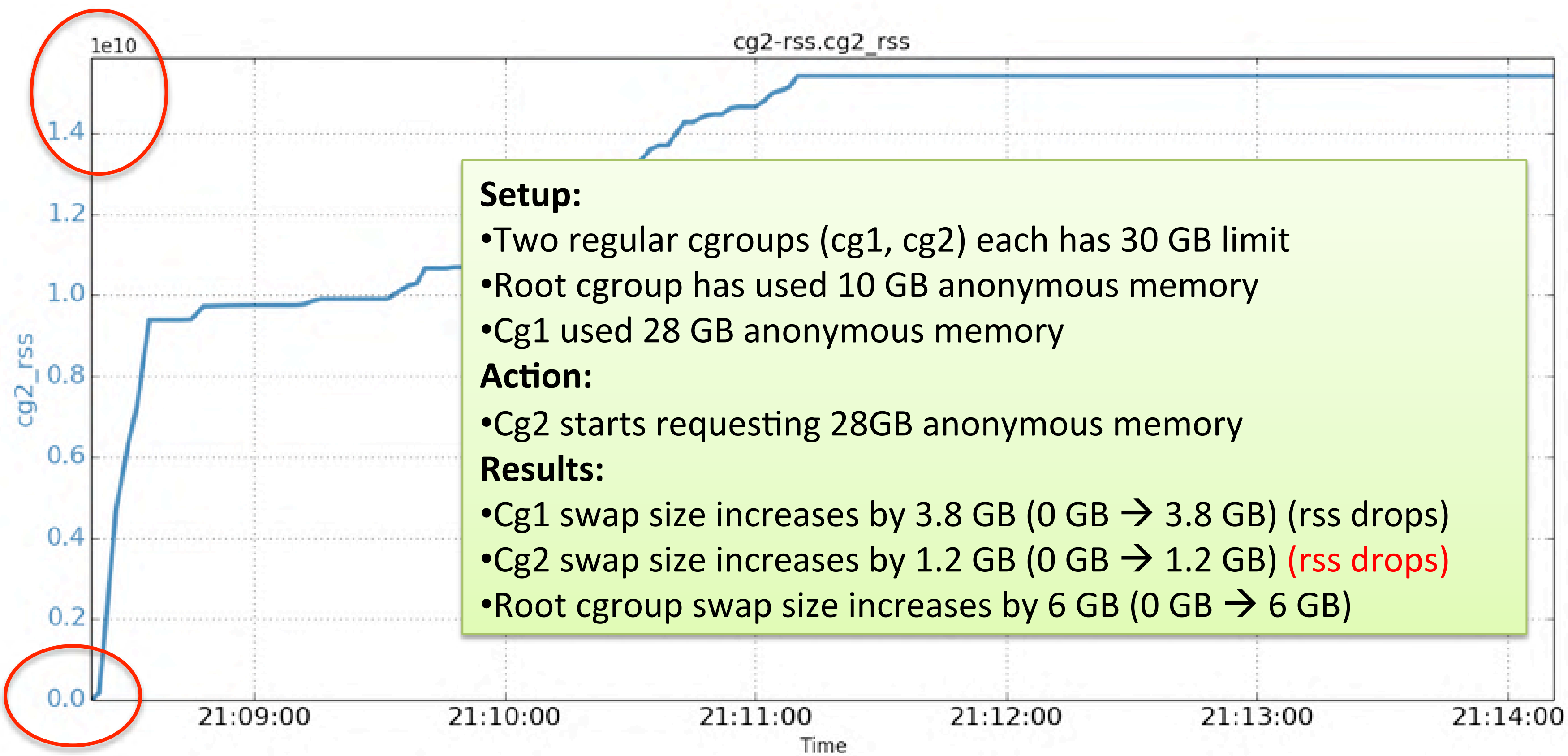
Action:

- Cg2 starts requesting 28GB anonymous memory

Results:

- Cg1 swap size increases by 3.8 GB (0 GB → 3.8 GB) (rss drops)
- Cg2 swap size increases by 1.2 GB (0 GB → 1.2 GB) (rss drops)
- Root cgroup swap size increases by 6 GB (0 GB → 6 GB)

Pitfall 4: Experiment results (Cg2's rss)



Setup:

- •Two regular cgroups (cg1, cg2) each has 30 GB limit
- •Root cgroup has used 10 GB anonymous memory
- •Cg1 used 28 GB anonymous memory

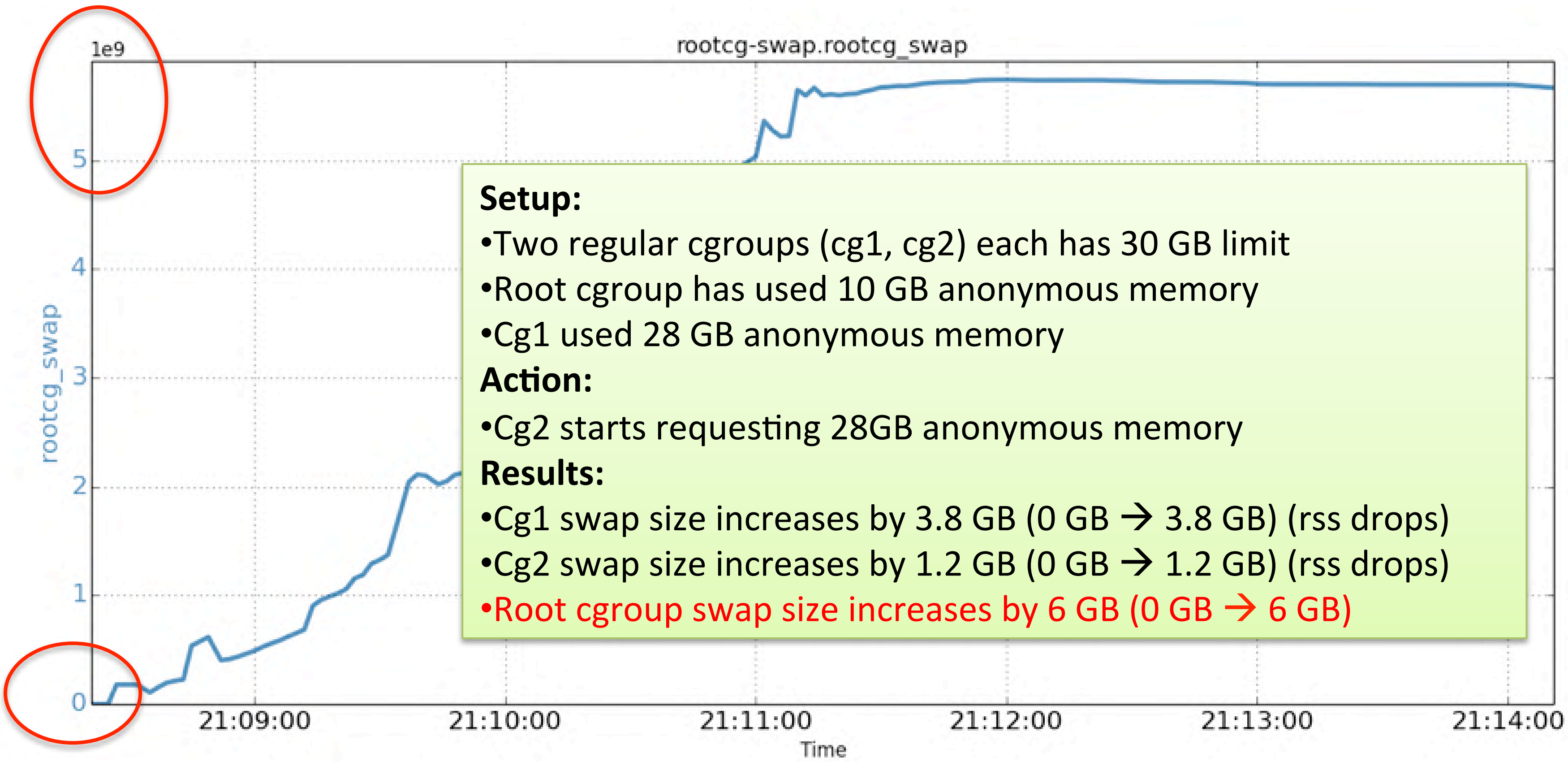
Action:

- •Cg2 starts requesting 28GB anonymous memory

Results:

- •Cg1 swap size increases by 3.8 GB (0 GB → 3.8 GB) (rss drops)
- •Cg2 swap size increases by 1.2 GB (0 GB → 1.2 GB) (rss drops)
- •Root cgroup swap size increases by 6 GB (0 GB → 6 GB)

Pitfall 4: Experiment results (root cgroup's swap)



Setup:

- Two regular cgroups (cg1, cg2) each has 30 GB limit
- Root cgroup has used 10 GB anonymous memory
- Cg1 used 28 GB anonymous memory

Action:

- Cg2 starts requesting 28GB anonymous memory

Results:

- Cg1 swap size increases by 3.8 GB (0 GB → 3.8 GB) (rss drops)
- Cg2 swap size increases by 1.2 GB (0 GB → 1.2 GB) (rss drops)
- Root cgroup swap size increases by 6 GB (0 GB → 6 GB)

APMCon

Pitfall 5: **Virtual memory is not isolated**

(RSS vs. Virtual Memory)

- RSS: Resident set size
- VM: Process memory map (mmap, library, etc.)

```
top - 13:06:46 up 5 days, 22:41, 18 users,  load average: 0.77, 1.02, 0.65
Tasks: 386 total,   2 running, 382 sleeping,   0 stopped,   2 zombie
Cpu(s): 19.3%us,  0.2%sy,  0.0%ni, 80.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  65893764k total, 23425308k used, 42468456k free,   892084k buffers
Swap: 67106812k total,        0k used, 67106812k free,  4369792k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
25995 zzhuang   20   0 24.7g  12g  12m S 217.5 20.2   0:52.33 java
 5759 zzhuang   20   0 4164m 2.5g  62m S 13.9  4.0 951:12.35 firefox
 4364 root      20   0  294m 142m  14m S  0.7  0.2  54:05.08 Xorg
25841 zzhuang   20   0 35312 2664 1760 R  0.7  0.0   0:01.55 top
  446 zzhuang   20   0  485m  39m  17m S  0.3  0.1   0:22.64 /usr/bin/termin
 3854 root      20   0 35348 4648 1700 S  0.3  0.0  22:32.04 cf-serverd
 5707 zzhuang   20   0  161m 4100 2780 S  0.3  0.0   0:58.76 ibus-daemon
 5715 zzhuang   20   0  375m  29m  16m S  0.3  0.0   1:12.44 python
```

Pitfall 5: **Virtual memory is not limited**

- VM space is limited with disabled overcommit
  - (Swap space size + RAM * overcommit_ratio)
  - E.g., RAM=64GB, swap=32GB, ratio=50%. VM=64GB
- VM limit is system-wide
  - All processes aggregated
  - Cgroups do not limit VM
- Impact
  - Applications may fail to start or suddenly fails

Pitfall 5: **Virtual memory is not limited** (Virtual memory of JVM applications)

- JVM heap
  - Xms=1GB, Xmx=5GB
- RSS
  - Heap + off-heap (perm, meta, direct) (4GB, 8GB)
- Virtual memory
  - JVM RSS + glibc memory pool
  - Glibc VM: threads*64MB; threads capped by cores*8

Pitfall 5: **Virtual memory is not limited**

(Virtual memory tests)

- JDK-1_8_0_49/java
  - Xms=Xmx=5G, Xss=1M
  - Glibc: 12 cores, max is 6144MB

# app threads	JVM native (MB)	# of JVM threads	Glibc mem pool VM (MB) (min of JVM TH*64, 6144)	Sum of JVM native and glibc mem pool (MB)	Actual VM size (MB, pidstat)
1	6802	24	1536	8338	8396
5	6806	28	1792	8598	8662
20	6822	43	2752	9574	9660
50	6852	73	4672	11524	11637
100	6904	123	6144	13048	13282

APMCon

Pitfall 5: **Virtual memory is not limited**

Overcommit setting

- Overcommit disabled
  - Vm.overcommit_memory=2
  - VM size limited by swap size and overcommit ratio
- JVM applications in cgroups
  - 64GB total VM (RAM=64GB, swap=32GB, ratio=50)
  - Each JVM 12GB VM
  - Max 5 cgroups (Not considering other processes)
- Applications may request more VM
  - Failure

Outline

- ❑ Introduction
- ❑ Memory related performance pitfalls
- ❑ Strategies
- ❑ Discussions
- ❑ Conclusion

Strategies

- ❑ Properly sizing memory footprint of apps
- ❑ Pre-touching cgroup memory
- ❑ Tightly controlling root cgroups
- ❑ Limiting VM usage of cgroups

APMCon

Properly sizing mem footprint of apps

- Cgroup memory limit
  - Based on app memory footprint
  - Both anonymous memory and page cache
- Anonymous memory footprint
  - Relatively easy
- Page cache footprint
  - Not possible on baremetal (non-cgroup env)
  - No Linux metrics
  - Further complexities (startup, prefetching, logging)

Sizing memory footprint on cgroups

- Metrics
  - Memory.stat (many metrics of current usage)
  - Memory.failcnt
- Anonymous memory – **rss**
  - Accurate
  - Current value is the needed value
- Page cache – **active_file**
  - Approximate
  - Current value may be less than needed (spikes)
  - Give a buffer

APMCon

Pre-touching cgroup memory

- Cgroups does not allocate memory
- Java heap
  - Xms=Xmx
  - -XX:+AlwaysPreTouch
- Protecting the memory
  - Swappiness=0

Tightly controlling root cgroup

- Root cgroup is unbounded
  - Regular cgroup is bounded
  - Root cgroup more likely starve other cgroups
- Scenarios
  - Sshd, crond, CFEngine, etc.
- Moving out as many processes as possible
  - Special cgroups with memory limit

Limiting VM usage of cgroups

- VM is a precious resource
  - Just like other types (memory, cpu, etc.)
- Overcommit disabled
  - Enough swap space
  - Limiting VM usage of each cgroup
- Overcommit enabled
  - Processes in cgroups can request *infinite* VM
  - When RSS reaching memory limit
    - OOM: swappiness = 0
    - Swapping: swappiness > 0

Discussions

- ❑ Design rational of cgroups vs virtual machine
- ❑ Taming the resource usage of system processes
- ❑ Extreme scenarios are worth to consider
- ❑ Monitoring, alerting, enforcing, debugging

Conclusion

- Cgroups based deployments are getting popular
  - Cgroups, Linux containers, Docker, CoreOS
- Performance pitfalls exist in certain scenarios
  - Focusing on memory resource
- Various types of memory-pressure problems
  - Anonymous memory, page cache, virtual memory
- Strategies to mitigate these problems

Q/A

- Thanks!
- LinkedIn: https://www.linkedin.com/in/zhenyun
- Email: zhenyun@gmail.com