WOTD

51CTO

全球软件开发技术峰会

World Of Tech 2017

2017年12月1日-2日 • 深圳中洲万豪酒店

DEVELOPMENT

WOTD

51CTO

Java Performance

自底向上的分析实践

裴文谦 资深研发工程师 阿里巴巴

Optimizing HotSpot at Alibaba

Multi-tenant 多租户

JVM 级别virtualization, CPU/线程/内存资源隔离与管理, GC改造等

Wisp 协程

Stackful Coroutine for JVM, 自动异步化，免费的性能提升

JWarmup 预热

解决Java应用启动时RT超时及load彪高问题, 帮助应用在高峰时发布或重启

JTenant 租户虚拟化容器

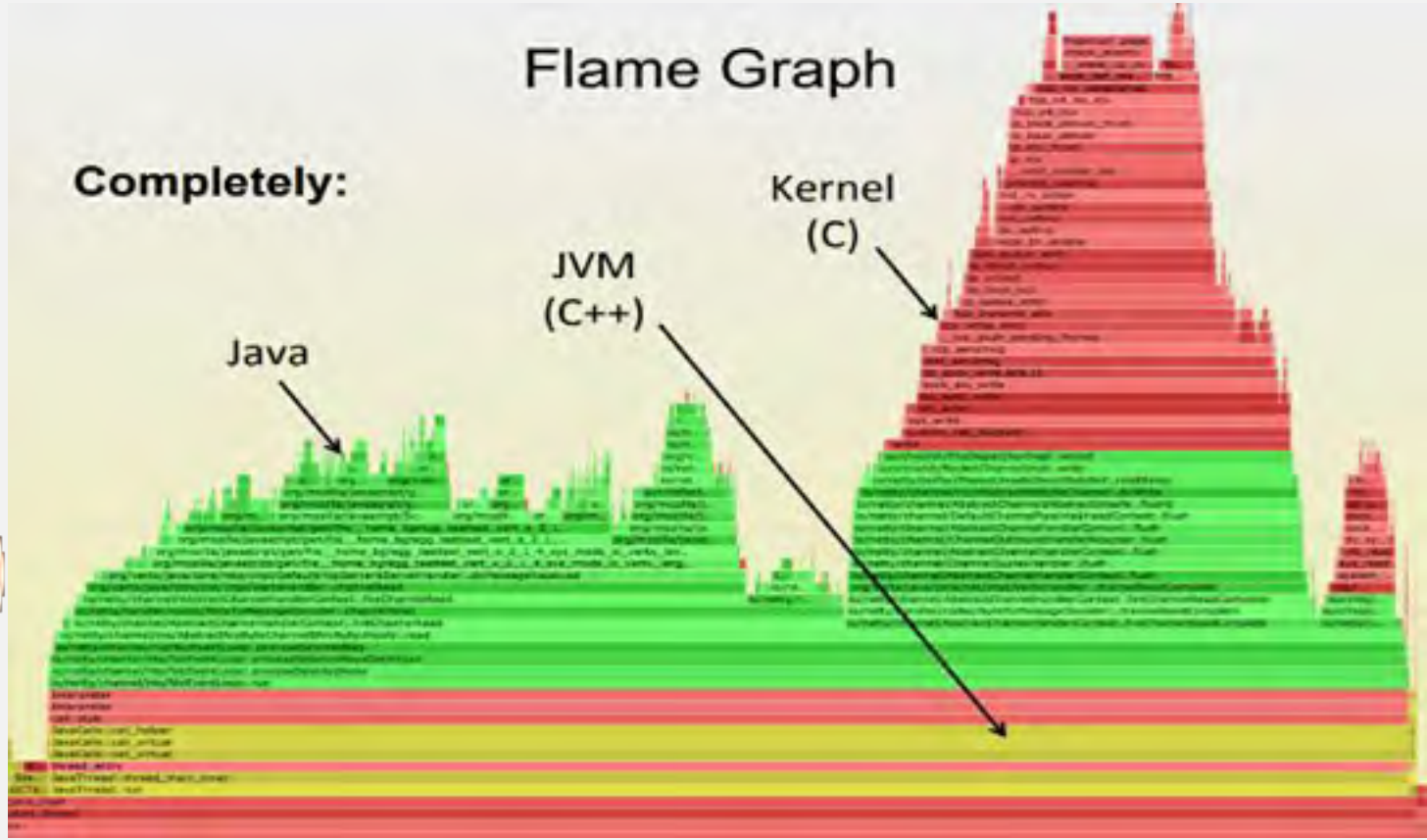
基于多租户的容器技术, 提供运维监控功能等, 屏蔽复杂性

“What We Talk About

When We Talk About Performance(?)”

—— Raymond Carver

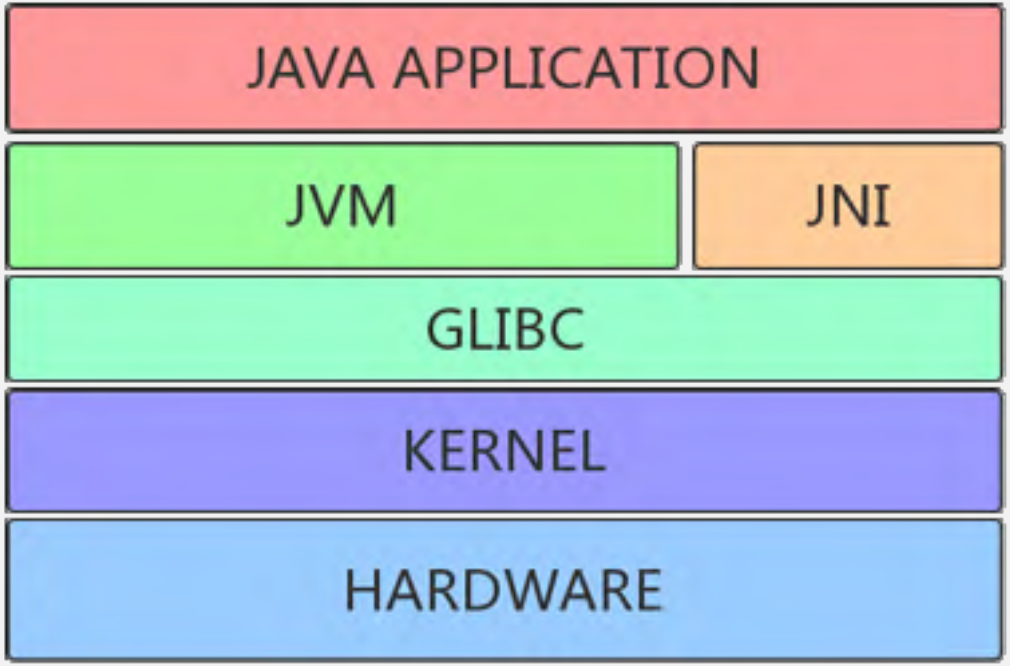
Performance and Stability



https://www.slideshare.net/brendangregg/scale2015-linux-perfprofiling

当业务指标异常的时候, 其实每层都可能出问题

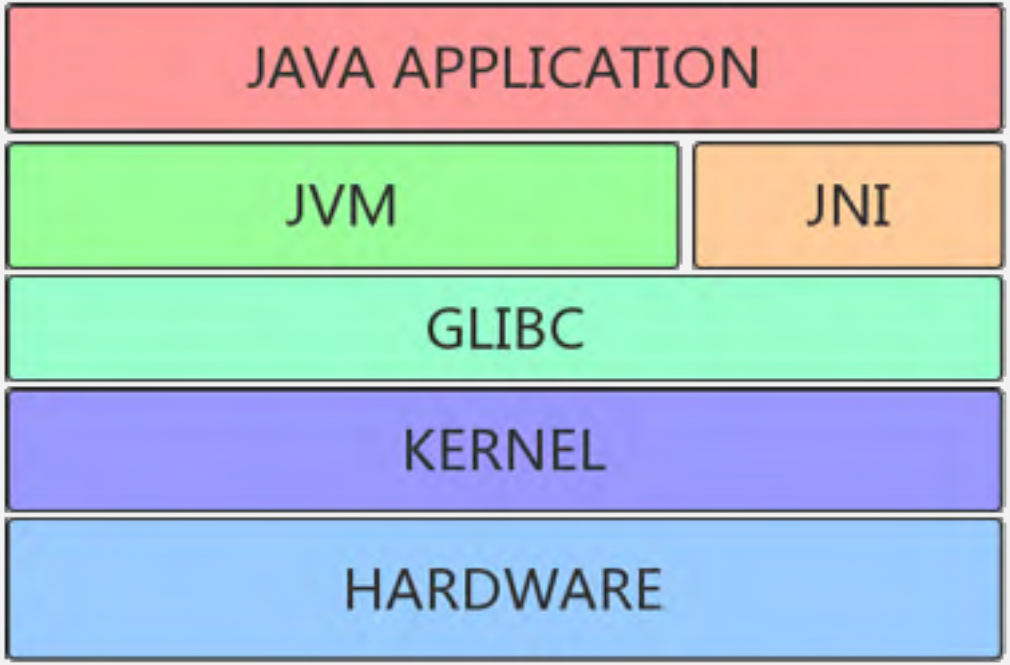
App Owner



突然超时！！

1. 来看看系统指标 (CPU 网络等)
2. 来看看 GC LOG
3. 来看看JVM状态
4. 看下你依赖的服务 (DB DNS等)

什么会导致应用响应超时？



- 业务后端超时等
- FullGC等
- DNS等
- Swap, 网络等
- 磁盘错误影响GC暂停

Profiling Tools in Alibaba

Zprofiler

云端MAT + Hot Method + GC Analysis

AIA(All In Analysis)

Java Live Profiler, wall time tracing

扁鹊

Slow sample, kernel + User(Java) profiling

Sampling vs Tracing

原则上，Profiling 对系统性能的影响越小越好

· slow sampling

· sample + tracing

Sampling vs Tracing

Sample 方法：

· PMU(hardware)

· kernel module

· signal

Trace 方法：

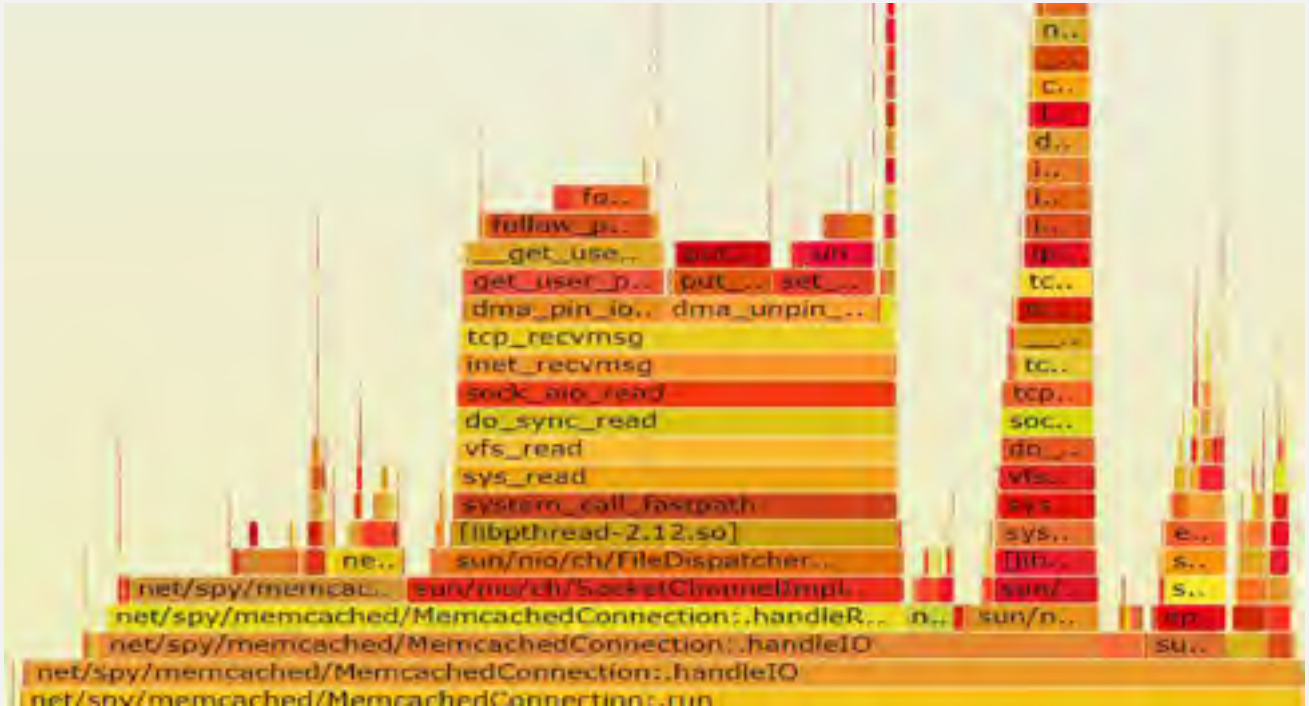
· 方法插桩

Case study

Case: jemalloc causes low performance

Linux 2.6.32 换上AJDK后导致性能下降：

Normal：



Case: jemalloc causes low performance

Linux 2.6.32 换上AJDK后导致性能下降：

Abnormal：

Case: high sys usage

perf 看到大量page fault

正常缺页？

非正常缺页？

```
-   22.42%   14.97%  java     [kernel.kallsyms]  [k] page_fault
  - page_fault
     - 98.82% 0
        - 60.63% 0x7f8cbbfa8544
           - 30.84% 0x7f8cb9f212a8
              - 27.38% 0x7f8cbc26a3ec
                   0x7f8cbbecdd84
                   0x7f8cbb3458c8
                 - 0x7f8cb8c44b08
                    - 93.38% 0x7f8cbbafec8c
                         0x7f8cbb808c1c
                       + 0x7f8cbbb09330
                    + 3.53% 0x7f8cbe2d7020
                    + 3.09% 0x7f8cbe2d7180
              + 27.30% 0x7f8cbc26a3b0
              + 26.10% 0x7f8cbc26a390
              + 18.68% 0x7f8cbc26a5ac
```

Case: high sys usage

打上符号(工具: perf-map-agent)

Case: high sys usage

写一段Systemtap看下缺页的地址：

```
1 global addr
2 probe vm.pagefault {
3   addr[$address % (4 * 1024)] <<< 1;
4 }
5 probe end {
6   foreach (p in addr) {
7     printf("%p %d\n", p, @count(addr[p]));
8   }
9 }
```

```
0x148 10
0xbe0 25
0x6f0 31
0x9b8 9
0xc60 52
0xb40 21
0x530 15
0x1b0 39
0x0 2872          <-- WTF!
0xe88 27
0x910 68
0xc58 5
0x9f8 12
```

Case: high sys usage

原来是aliperf的bug导致的...

```
top - 16:15:05 up 1 day, 23:01,  4 users,  load average: 5.80, 5.01, 3.91
Tasks:  67 total,   3 running,  53 sleeping,   8 stopped,   3 zombie
Cpu(s): 44.3%us, 54.2%sy,  0.0%ni,  1.3%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:   4194304k total,  4151572k used,    42732k free,        0k buffers
Swap:  2097148k total,        0k used,  2097148k free,  1405720k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
86355 admin     20   0 5410m 2.5g  25m S 88.2 61.4  20:14.74 java
84853 root      20   0  157m 2276  928 R 52.9  0.1   1468:53 aliperf     <-- !!!!!
118326 root     20   0  157m 4216 2872 R 51.6  0.1   3:11.20 aliperf
```

执行 killall aliperf 解决

Case: java coroutine creation slow

现象：在创建百万协程测着中，CentOS 5u的机器明显慢于CentOS 7u的机器。

时间开销：30s -> 2s

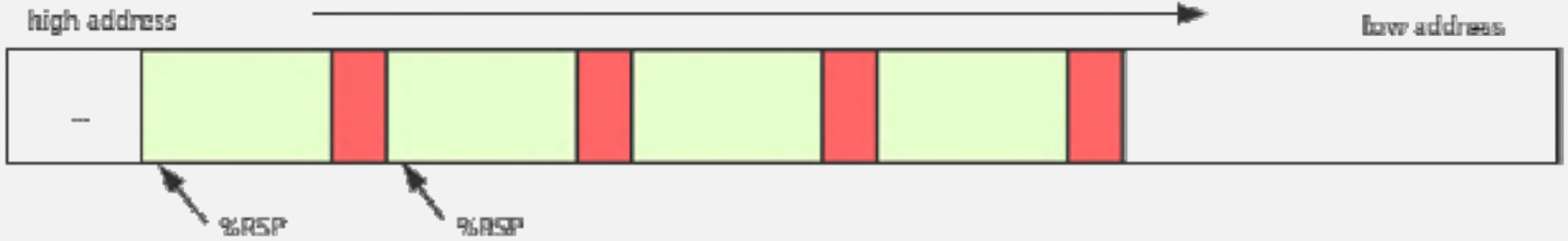
```
1  public static void main(String... args) throws Exception {
2      ExecutorService pool = new WispMultiThreadExecutor(processors, cnt);
3      CountDownLatch latch = new CountDownLatch(cnt);
4      for (int i = 0; i < 30000; ++i) {
5          pool.submit(() -> {
6              TimeUnit.SECONDS.sleep(2);
7              latch.countDown();
8          });
9      }
10     latch.await();
11 }
```

Case: java coroutine creation slow

strace观察系统调用：

```
mmap(NULL, 1318912, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb2c0386000
mmap(0x7fb2c0386000, 1318912, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fb2c0386000
mprotect(0x7fb2c0386000, 4096, PROT_NONE) = 0
mmap(NULL, 1318912, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb2c0244000
mmap(0x7fb2c0244000, 1318912, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fb2c0244000
mprotect(0x7fb2c0244000, 4096, PROT_NONE) = 0
...
```

Java stack 示意图：



Case: java coroutine creation slow

针对mmap写了测试程序来perf热点。

```
Samples: 259K of event 'cycles', Event count (approx.): 154192030962
   94.98%  a.out  [kernel.kallsyms]            [k] find_vma
   - find_vma
      - 98.83% arch_get_unmapped_area_topdown
           get_unmapped_area_prot
           do_mmap_pgoff
           sys_mmap_pgoff
           sys_mmap
           system_call
         + __GI___mmap64
      + 1.17% get_unmapped_area_prot
+   2.76%  a.out  [kernel.kallsyms]            [k] arch_get_unmapped_area_topdown
+   1.55%  a.out  [kernel.kallsyms]            [k] align_addr
+   0.21%  a.out  [kernel.kallsyms]            [k] hpet_readl
+   0.08%  a.out  [kernel.kallsyms]            [k] do_softirq
+   0.06%  a.out  [kernel.kallsyms]            [k] perf_event_mmap_ctx
+   0.05%  a.out  [kernel.kallsyms]            [k] find_vma_prepare
```

Case: java coroutine creation slow

借助Systemtap来追踪kernel

```
probe kernel.function("find_vma").return {
  if (pid() == target()) {
    printf("find_vma mmap_base=%p, hole=%lx, cache=%p, vars: %s\n",
      $mm->mmap_base, $mm->cached_hole_size, $mm->free_area_cache, $$vars);
  }
}

probe kernel.function("arch_get_unmapped_area_topdown").call {
    ...
```

Case: java coroutine creation slow

借助Systemtap来追踪kernel

```
1  find_vma mmap_base=0x7fd9d6f2e000, hole=1000, cache=0x7fd9ceeef000, vars: mm=0xffff880bdf34d800 addr=0x7fd9ceeef000 vma=0xffff880c14ac71c0
2
3  // start of an f1 call
4  get_unmapped filp=0x0 addr0=0x0 len=0x2000 pgoff=0x0 flags=0x22
5
6  // free_area_cache lookup
7  find_vma mmap_base=0x7fd9d6f2e000, hole=1000, cache=0x7fd9ceeef000, vars: mm=0xffff880bdf34d800 addr=0x7fd9ceeed000 vma=0xffffffff
8
9  // linear search
10 find_vma mmap_base=0x7fd9d6f2e000, hole=1000, cache=0x7fd9ceeef000, vars: mm=0xffff880bdf34d800 addr=0x7fd9d6f2c000 vma=0xffffffff
11 find_vma mmap_base=0x7fd9d6f2e000, hole=1000, cache=0x7fd9ceeef000, vars: mm=0xffff880bdf34d800 addr=0x7fd9d6f2a000 vma=0xffffffff
12 find_vma mmap_base=0x7fd9d6f2e000, hole=1000, cache=0x7fd9ceeef000, vars: mm=0xffff880bdf34d800 addr=0x7fd9d6f28000 vma=0xffffffff
13 find_vma mmap_base=0x7fd9d6f2e000, hole=1000, cache=0x7fd9ceeef000, vars: mm=0xffff880bdf34d800 addr=0x7fd9d6f25000 vma=0xffffffff
14
```

通过trace对比发现arch_get_unmapped_area_topdown对地址空间进行了线性扫描

（linux kernel 3.x重写了这段代码，所以高版本OS就不存在这个BUG）

Systemtap tools for Java

Systemtap introduction

Dtrace的linux版本，通过编写脚本定制化trace任务。同时支持内核态和用户态trace和插桩

How to probe Java

最关键的是.... SYMBOL!

```
Thread 136 (Thread 0x489f0940 (LWP 1934)):
#0  0x00002b26a8359d98 in epoll_wait () from /lib64/libc.so.6
#1  0x00002aaac7606fea in Java_sun_nio_ch_EPollArrayWrapper_epollWait (
#2  0x00002aaaac924d29 in ?? ()
#3  0x0000000742f47c78 in ?? ()
#4  0x00002b26a98ebf6f in os::javaTimeNanos() () from /opt/taobao/insta
#5  0x00002aaaabe112bc in ?? ()
#6  0x0000000742f47c78 in ?? ()
#7  0x0000000000001384 in ?? ()
#8  0xf30066d5438fd998 in ?? ()
#9  0xe8076ccf00000000 in ?? ()
#10 0x0000000742f47c78 in ?? ()
#11 0x00002b26e871fb36 in ?? ()
#12 0x00000000489ef0b0 in ?? ()
#13 0x0000000742f47d20 in ?? ()
#14 0x0000000000000000 in ?? ()
```

Example: Trace DNS resolution

```
1 probe process("/usr/lib64/libc-2.17.so").function("getaddrinfo") {
2   if ($condition) {
3     init_global()
4     bt = get_java_backtrace()
5     printf("%s\\n", bt);
6   }
7 }
```

```
N:getaddrinfo
N:Java_java_net_Inet4AddressImpl_lookupAllHostAddr
I:java/net/InetAddress$2:lookupAllHostAddr(Ljava/lang/String;)[Ljava/ne
I:java/net/InetAddress:getAddressesFromNameService(Ljava/lang/String;Lj
I:java/net/InetAddress:getAllByName0(Ljava/lang/String;Ljava/net/InetAd
I:java/net/InetAddress:getAllByName(Ljava/lang/String;Ljava/net/InetAdd
I:java/net/InetAddress:getAllByName(Ljava/lang/String;)[Ljava/net/InetA
I:DNSTest:foo()V
I:DNSTest:main([Ljava/lang/String;)V
C:0x7fe72198c4e7
N:_ZN9JavaCalls11call_helperEP9JavaValueP12methodHandleP17JavaCallArgum
N:_ZN9JavaCalls11call_helperEP9JavaValueP12methodHandleP17JavaCallArgum
N:_ZL17jni_invoke_staticP7JNIEnv_P9JavaValueP8_jobject11JNICallTypeP10_
N:jni_CallStaticVoidMethod
N:JavaMain
N:start_thread
N:clone
```

通过给glibc的getaddrinfo插桩拿到当时Java backtrace

Example: Lock contention

```
# begin profile
$ sudo ./lock-profile -p 8036 -s 10 > lock-profile-output.log

# you will get
# cat lock-profile-output.log

lock object: 0xd1c69968   contention total is: 2435
thread detail:
   tid : 11606, contention count : 556
   tid : 11607, contention count : 1879
(unit: us)
value |---------------------------------------------------- count
    0 |                                                     0
    1 |                                                     0
    2 |                                                     7
    4 |@                                                    20
    8 |                                                     19
   16 |                                                     6
   32 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                    630
   64 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  970
  128 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@             759
  256 |@                                                    20
  512 |                                                     1
 1024 |                                                     1
```

```
cost time: 234 us, backtrace is:
N:Unsafe_Park
C:0x7f56b928cf71
C:0x7f56b929d804
I:TestLock:loop(I)V
I:TestLock:lambda$threadLoop$1(I)V
I:TestLock$$Lambda$2:run()V
I:java/lang/Thread:run()V
C:0x7f56b9000671
N:_ZN9JavaCalls11call_helperEP9JavaValueP12methodHandleP17Jav
N:_ZN9JavaCalls11call_helperEP9JavaValueP12methodHandleP17Jav
N:_ZN2os20os_exception_wrapperEPFvP9JavaValueP12methodHandleP
N:_ZN9JavaCalls4callEP9JavaValue12methodHandleP17JavaCallArgu
N:_ZN9JavaCalls12call_virtualEP9JavaValue11KlassHandleP6Symbo
N:_ZN9JavaCalls12call_virtualEP9JavaValue6Handle11KlassHandle
N:_ZL12thread_entryP10JavaThreadP6Thread
N:_ZN10JavaThread17thread_main_innerEv
N:_ZN10JavaThread3runEv
N:_ZL10java_startP6Thread
N:start_thread
N:clone
```

Thank you!