Fault Localization

Objective

- Explore the existing tools and methods for fault localization.
- Identify the exact location of the contended locks.

Perf tool

We can see the symbols from the memory spaces which are consuming the highest percentage of the CPU. This happens because multiple threads are running creating lock contention.

```
   1   0.01% psi_task_change
   1   0.01% amd_pmu_disable_all
   1   0.01% newidle_balance
   1   0.01% schedule
   1   0.01% perf_event_task_tick
   1   0.01% timekeeping_advance
   1   0.01% update_load_avg
   1   0.01% interrupt_entry
   1   0.01% load_balance
   1   0.01% x86_pmu_disable_all
   1   0.01% update_dl_rq_load_avg
   1   0.01% find_next_and_bit
   1   0.01% __sched_text_start
   1   0.01% irq_work_run_list
   1   0.01% task_tick_fair
   1   0.01% sched_clock_cpu
   2   0.02% __update_load_avg_se
   2   0.02% sleep_micros
   2   0.02% update_nohz_stats
   2   0.02% _raw_spin_lock_irqsave
   2   0.02% MM_CompactSchemeFixupObject::verifyForwardingPtr
   2   0.02% __update_load_avg_cfs_rq
   3   0.03% usleep
   4   0.04% native_read_msr
   6   0.07% ctx_sched_in
   6   0.07% common_nsleep
   6   0.07% visit_groups_merge
   6   0.07% flexible_sched_in
   7   0.08% native_write_msr
  12   0.13% delay_mwaitx
  13   0.14% MM_CompactSchemeFixupObject::fixupArrayObject
  18   0.20% MM_CompactScheme::fixupObjectSlot
 224   2.43% MM_CompactSchemeFixupObject::fixupObject
 769   8.34% MM_CompactScheme::fixupSubArea
 836   9.07% MM_CompactSchemeFixupObject::fixupMixedObject
7275  78.89% MM_CompactScheme::getForwardingPtr
9222         --total--
```

JLM Trace

The java inflated monitors records mean that one or more threads are trying to acquire the object-monitor simultaneously.
GETS is the no of time that a thread tries to acquire a lock. As we can see below the high no of GETS along with TIER2 and TIER3 spinning indicates lock contention in the following class.

```
740      0        0        0        0        0        0        0     0        0  [00007FA4F4069140] MM_PacketList:_sublists[]._lock
741      0        0        0        0        0        0        0     0        0  [00007FA4F4069298] MM_PacketList:_sublists[]._lock
742      0        0        0        0        0        0        0     0        0  [00007FA4F40693F0] MM_PacketList:_sublists[]._lock
743      0        0        0        0        0        0        0     0        0  [00007FA4F4069AA8] MM_PacketList:_sublists[]._lock
744      0        0        0        0        0        0        0     0        0  [00007FA4F4069C00] MM_PacketList:_sublists[]._lock
745      0        0        0        0        0        0        0     0        0  [00007FA4F406A160] MM_PacketList:_sublists[]._lock
746      0        0        0        0        0        0        0     0        0  [00007FA4F406A2B8] MM_PacketList:_sublists[]._lock
747
748  Java (Inflated) Monitors
749
750  %MISS     GETS   NONREC     SLOW      REC    TIER2    TIER3 %UTIL AVER_HTM  MON-NAME
751
752      0   121731   121731        0        0   783074     5421     0     1658  [00007FA4F4687958] net/adoptopenjdk/bumblebench/examples/SynchronizedMethodBench@00000000E00
     17BF8 (Class)
753
754  LEGEND:
755                 %MISS : 100 * SLOW / NONREC
756                  GETS : Lock Entries
757                NONREC : Non Recursive Gets
758                  SLOW : Non Recursives that Block
759                   REC : Recursive Gets
760                 TIER2 : SMP: Total try-enter spin loop cnt (middle for 3 tier)
761                 TIER3 : SMP: Total yield spin loop cnt (outer for 3 tier)
762                 %UTIL : 100 * Hold-Time / Total-Time
763               AVER-HT : Hold-Time / NONREC
                                                                                                                  763,15          Bot
```

Either of these tools cannot locate to the method that is being contended by multiple threads.

Call Stack Traces:

- This is used for call graph profiling
- It has a mode for profiling based on contended monitor entry
- Monitors that are going through massive levels of spin cs can record the call stacks

There's a tool called **mxview.jar**
this tool is a java utility for viewing scs output.

- The agent lib command is modified for this particular operation:
  agentlib:jprof=scs=monitor_contended_entered+
  - The jprof is told to run in a different mode, set itself up for call stack sampling, and monitor contended entered event is told to use.
  - The plus sign is there to **record the type of the object** that's being contended.

agentlib: jprof -jar : uses this command for recording jlm and perf data

agentlib: jprof=scs=monitor_contended_entered+ : uses this command to record the call stack traces

```
[root@animal training]# /home/ajcraik/sdk/bin/java -DBumbleBench.parallelInstances=4
-agentlib:jprof=scs=monitor_contended_entered+,logpath=$PWD -jar ./BumbleBench.jar Sy
nchronizedMethodBench
```

Rtdriver is used for the operation:
rtdriver -a 127.0.0.1 -c start 15 -c stop 20

After the rtdriver is done recording we can see the **log-rt.1_111633** which has the actual scs output but is not human readable.

Prefs < > ○ xarc ◉ xprof ○ xslice

0	100.00	100.00	26567	26567	(java/lang/Class)
0	0.00	100.00	0	26567	net/adoptopenjdk/bumblebench/examples/SynchronizedMethodBench.computeFib(J)J
1	0.00	25.74	0	6838	0001adf4_Thread-8_pidtid
0	0.00	100.00	0	26567	net/adoptopenjdk/bumblebench/core/Launcher.runBumbleMainOn(Lnet/adoptopenjdk/bumblebench/core/BumbleBench;)V
0	0.00	100.00	0	26567	sun/reflect/DelegatingMethodAccessorImpl.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
0	0.00	100.00	0	26567	net/adoptopenjdk/bumblebench/core/BumbleBench$WorkerThread.run()V
0	0.00	100.00	0	26567	net/adoptopenjdk/bumblebench/core/MiniBench.doBatch(J)J
0	0.00	100.00	0	26567	net/adoptopenjdk/bumblebench/core/MicroBench.attempt(F)F
1	0.00	24.84	0	6599	0001adf5_Thread-9_pidtid
0	0.00	100.00	0	26567	java/lang/reflect/Method.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
0	0.00	100.00	0	26567	net/adoptopenjdk/bumblebench/core/BumbleBench.bumbleMain()V
1	0.00	24.80	0	6589	0001adf6_Thread-10_pidtid
0	0.00	100.00	0	26567	sun/reflect/NativeMethodAccessorImpl.invoke0(Ljava/lang/reflect/Method;Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
0	0.00	100.00	0	26567	net/adoptopenjdk/bumblebench/core/BumbleBench.runAsWorker()V
0	0.00	100.00	0	26567	sun/reflect/NativeMethodAccessorImpl.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
1	0.00	24.62	0	6541	0001adf3_Thread-7_pidtid

The first value from the list **(java/lang/Class)** is the type of an object that's being locked. There's only 1 lock in the system and there are above 26K locking events.

Prefs < > xarc xprof xslice

Parent	0	0.00	100.00	0	26567	net/adoptopenjdk/bumblebench/core/MiniBench.doBatch(J)J
Self	0	0.00	100.00	0	26567	net/adoptopenjdk/bumblebench/examples/SynchronizedMethodBench.computeFib(J)J
Child	0	100.00	100.00	26567	26567	(java/lang/Class)

We can view the **xarc** view of the contended object.
We can see that the object was locked from the synchronized method and can locate the parent of each class.

Prefs < > xarc xprof xslice (

```
(java/lang/Class)
java/lang/reflect/Method.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
net/adoptopenjdk/bumblebench/core/BumbleBench$WorkerThread.run()V
net/adoptopenjdk/bumblebench/core/BumbleBench.bumbleMain()V
net/adoptopenjdk/bumblebench/core/BumbleBench.runAsWorker()V
net/adoptopenjdk/bumblebench/core/Launcher.runBumbleMainOn(Lnet/adoptopenjdk/bumblebench/core/BumbleBench;)V
net/adoptopenjdk/bumblebench/core/MicroBench.attempt(F)F
net/adoptopenjdk/bumblebench/core/MiniBench.doBatch(J)J
net/adoptopenjdk/bumblebench/examples/SynchronizedMethodBench.computeFib(J)J
sun/reflect/DelegatingMethodAccessorImpl.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
sun/reflect/NativeMethodAccessorImpl.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
sun/reflect/NativeMethodAccessorImpl.invoke0(Ljava/lang/reflect/Method;Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
```

We can search from **xslice** for parentheses to find contended objects.

```java
package net.adoptopenjdk.bumblebench.examples;

import net.adoptopenjdk.bumblebench.core.MiniBench;

public final class SynchronizedMethodBench extends MiniBench {

        public static long next;

        public static long computeFib(long num) {
            long prev = 0, curr = 1;
            while (num-- > 0) {
                synchronized (SynchronizedMethodBench.class) {
                next = prev + curr;
                prev = curr;
                curr = next;
                if (pauseDuration > 0) {
                   try {
                     Thread.sleep(0, pauseDuration);
                   } catch (InterruptedException e) {
                   }
                }
                }
            }
            return prev;
        }

        // Illustrates a MicroBench with an infinite score, for testing purposes

        private volatile long result;

        protected int maxIterationsPerLoop() { return maxIterations; }

        protected long doBatch(long numLoops, int numIterationsPerLoop) throws InterruptedException {
                startTimer();
                for (long i = 0; i < numLoops; ++i) {
                result = computeFib(numIterationsPerLoop);
                }
                pauseTimer();
                return numLoops * numIterationsPerLoop;
        }

        private static int maxIterations;
        private static int pauseDuration;
        static {
             String iters = System.getProperty("SynchronizedMethodBench.iterationCount");
             maxIterations = 9072;
```

Contended object found:
java/lang/class

Parent:
net/adoptopenjdk/bumblebench/SynchronisedMethodBench.computeFib

- We can figure out the name of the method here: **computeFib**

Parent:
net/adoptopenjdk/bumblebench/core/Minibench

- The class SynchronizedMethodBench was extended from Minibench.

```
  LV       EVENT  NAME
   0           0  0001b464_Thread-7
   1           0  net/adoptopenjdk/bumblebench/core/BumbleBench$WorkerThread.run()
V
   2           0  net/adoptopenjdk/bumblebench/core/Launcher.runBumbleMainOn(Lnet/
adoptopenjdk/bumblebench/core/BumbleBench;)V
   3           0  java/lang/reflect/Method.invoke(Ljava/lang/Object;[Ljava/lang/Ob
ject;)Ljava/lang/Object;
   4           0  sun/reflect/DelegatingMethodAccessorImpl.invoke(Ljava/lang/Objec
t;[Ljava/lang/Object;)Ljava/lang/Object;
   5           0  sun/reflect/NativeMethodAccessorImpl.invoke(Ljava/lang/Object;[L
java/lang/Object;)Ljava/lang/Object;
   6           0  sun/reflect/NativeMethodAccessorImpl.invoke0(Ljava/lang/reflect/
Method;Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
   7           0  net/adoptopenjdk/bumblebench/core/BumbleBench.bumbleMain()V
   8           0  net/adoptopenjdk/bumblebench/core/BumbleBench.runAsWorker()V
   9           0  net/adoptopenjdk/bumblebench/core/MicroBench.attempt(F)F
  10           0  net/adoptopenjdk/bumblebench/core/MiniBench.doBatch(J)J
  11           0  net/adoptopenjdk/bumblebench/examples/SynchronizedMethodBench.do
Batch(JI)J
```

```
74    0           0  0001b467_Thread-10
75    1           0  net/adoptopenjdk/bumblebench/core/BumbleBench$WorkerThread.run()
V
76    2           0  net/adoptopenjdk/bumblebench/core/Launcher.runBumbleMainOn(Lnet/
adoptopenjdk/bumblebench/core/BumbleBench;)V
77    3           0  java/lang/reflect/Method.invoke(Ljava/lang/Object;[Ljava/lang/Ob
ject;)Ljava/lang/Object;
78    4           0  sun/reflect/DelegatingMethodAccessorImpl.invoke(Ljava/lang/Objec
t;[Ljava/lang/Object;)Ljava/lang/Object;
79    5           0  sun/reflect/NativeMethodAccessorImpl.invoke(Ljava/lang/Object;[L
java/lang/Object;)Ljava/lang/Object;
80    6           0  sun/reflect/NativeMethodAccessorImpl.invoke0(Ljava/lang/reflect/
Method;Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
81    7           0  net/adoptopenjdk/bumblebench/core/BumbleBench.bumbleMain()V
82    8           0  net/adoptopenjdk/bumblebench/core/BumbleBench.runAsWorker()V
83    9           0  net/adoptopenjdk/bumblebench/core/MicroBench.attempt(F)F
84   10           0  net/adoptopenjdk/bumblebench/core/MiniBench.doBatch(J)J
85   11           0  net/adoptopenjdk/bumblebench/examples/SynchronizedMethodBench.do
Batch(JI)J
86   12           0  net/adoptopenjdk/bumblebench/examples/SynchronizedMethodBench.co
mputeFib(J)J
87   13        7545  (java/lang/Class)
```

Log-rt data for computeFib

```
Java (Inflated) Monitors

%MISS     GETS   NONREC     SLOW      REC    TIER2    TIER3 %UTIL AVER_HTM  MON-NAME

    0   121731   121731        0        0   783074     5421     0     1658  [00007FA4F4687958] net/adoptopenjdk/bumblebench/examples/SynchronizedMethodBench@00000000E00
17BF8 (Class)

LEGEND:
               %MISS : 100 * SLOW / NONREC
                GETS : Lock Entries
              NONREC : Non Recursive Gets
                SLOW : Non Recursives that Block
                 REC : Recursive Gets
               TIER2 : SMP: Total try-enter spin loop cnt (middle for 3 tier)
               TIER3 : SMP: Total yield spin loop cnt (outer for 3 tier)
               %UTIL : 100 * Hold-Time / Total-Time
             AVER-HT : Hold-Time / NONREC
```

Jlm data for computeFib example.

```
LV      EVENT  NAME
 0          0  000310c7_Thread-4
 1          0  Hot.run()V
 2          0  Hot.retriveData(I)Ljava/lang/Object;
 3         42  (java/lang/Object)
 0          0  000310c8_Thread-5
 1          0  Hot.run()V
 2          0  Hot.retriveData(I)Ljava/lang/Object;
 3         55  (java/lang/Object)
 0          0  000310c9_Thread-6
 1          0  Hot.run()V
 2          0  Hot.retriveData(I)Ljava/lang/Object;
 3         47  (java/lang/Object)
 0          0  000310ca_Thread-7
 1          0  Hot.run()V
 2          0  Hot.retriveData(I)Ljava/lang/Object;
 3         47  (java/lang/Object)
```

Hot_1

```
Java (Inflated) Monitors

%MISS     GETS   NONREC     SLOW      REC    TIER2    TIER3 %UTIL AVER_HTM  MON-NAME

    0   184751   184751       28        0 72647767  2185191   100   205012  [00007F9ECC0029B8] java/lang/Object@00000007FFF686
    0        0        0        0        0        0        0     0        0  [00007F9ECC001E08] java/lang/J9VMInternals$ClassIn
```

```
LV      EVENT  NAME
 0          0  00039571_Thread-4
 1          0  Hot.run()V
 2          0  Hot.calculateC()I
 3       2305  (java/lang/Object)
 3       2304  (java/util/ArrayList)
 2          0  Hot.calculateB()I
 3          1  (java/util/ArrayList)
 0          0  00039572_Thread-5
 1          0  Hot.run()V
 2          0  Hot.calculateC()I
 3       2305  (java/lang/Object)
 3       2302  (java/util/ArrayList)
 2          0  Hot.calculateB()I
 3          1  (java/util/ArrayList)
 0          0  00039573_Thread-6
 1          0  Hot.run()V
 2          0  Hot.calculateC()I
 3       2304  (java/lang/Object)
 3       2304  (java/util/ArrayList)
 2          0  Hot.calculateB()I
 3          1  (java/util/ArrayList)
 0          0  00039574_Thread-7
 1          0  Hot.run()V
 2          0  Hot.calculateC()I
 3       2305  (java/util/ArrayList)
 3       2304  (java/lang/Object)
```

Hot_2

```
Java (Inflated) Monitors

%MISS     GETS   NONREC     SLOW      REC    TIER2    TIER3 %UTIL AVER_HTM  MON-NAME

    3    79363    79363     2344        0 65023034  1991308    99   475804  [00007F59B4003358] java/lang/Object@000000060A25E1
    2     9842     9842      172        0  4092122   124311     4   166084  [00007F59B4002D28] java/util/ArrayList@000000060A2
```

```
LV      EVENT  NAME
 0          0  000310c7_Thread-4
 1          0  Hot.run()V
 2          0  Hot.retriveData(I)Ljava/lang/Object;
 3         42  (java/lang/Object)
 0          0  000310c8_Thread-5
 1          0  Hot.run()V
 2          0  Hot.retriveData(I)Ljava/lang/Object;
 3         55  (java/lang/Object)
 0          0  000310c9_Thread-6
 1          0  Hot.run()V
 2          0  Hot.retriveData(I)Ljava/lang/Object;
 3         47  (java/lang/Object)
 0          0  000310ca_Thread-7
 1          0  Hot.run()V
 2          0  Hot.retriveData(I)Ljava/lang/Object;
 3         47  (java/lang/Object)
```

```java
public void run(){
  while(true){
    retriveData(0);
  }
}
public Object retriveData(int ID) {
 try{
  long threadId = Thread.currentThread().getId();
  synchronized (dataBase) {
    if(sleepType == 0){
        Thread.sleep(sleepTime);
    }else {
      Thread.sleep(0, sleepTime);
    }
    System.out.println("Task done from thread : " + threadId);
  }
```

- The contented monitor (java/lang/Object)
- The method name of the synchronized region (retrieveData)
- Stack trace of all the methods called

```
LV      EVENT  NAME
 0          0  00039571_Thread-4
 1          0  Hot.run()V
 2          0  Hot.calculateC()I
 3       2305  (java/lang/Object)
 3       2304  (java/util/ArrayList)
 2          0  Hot.calculateB()I
 3          1  (java/util/ArrayList)
 0          0  00039572_Thread-5
 1          0  Hot.run()V
 2          0  Hot.calculateC()I
 3       2305  (java/lang/Object)
 3       2302  (java/util/ArrayList)
 2          0  Hot.calculateB()I
 3          1  (java/util/ArrayList)
 0          0  00039573_Thread-6
 1          0  Hot.run()V
 2          0  Hot.calculateC()I
 3       2304  (java/lang/Object)
 3       2304  (java/util/ArrayList)
 2          0  Hot.calculateB()I
 3          1  (java/util/ArrayList)
 0          0  00039574_Thread-7
 1          0  Hot.run()V
 2          0  Hot.calculateC()I
 3       2305  (java/util/ArrayList)
 3       2304  (java/lang/Object)
```

```java
public int calculateC() {
  try{
    long threadId = Thread.currentThread().getId();
    synchronized (A) {
      synchronized (B) {
        if(sleepType==0){
          Thread.sleep(sleepTime);
        } else{
          Thread.sleep(0, sleepTime);
        }
        System.out.println("Task done from thread : " + threadId);
```

```java
public int calculateB() {
  try{
    long threadId = Thread.currentThread().getId();
    synchronized (B) {
      if(sleepType==0){
        Thread.sleep(sleepTime);
      } else{
        Thread.sleep(0, sleepTime);
      }
      System.out.println("Task done from thread : " + threadId);
```

- Contention in 2 methods and 2 types of monitors.
- CalculateC has nested synchronized regions.

```
LV      EVENT  NAME
 0          0  001256d2_Thread-4
 1          0  OverlySplit.run()V
 2          0  OverlySplit.doSomething()V
 3       1726  (java/lang/Object)
 3        646  (java/util/ArrayList)
 0          0  001256d3_Thread-5
 1          0  OverlySplit.run()V
 2          0  OverlySplit.doSomething()V
 3        658  (java/util/ArrayList)
 3       1763  (java/lang/Object)
 0          0  001256d4_Thread-6
 1          0  OverlySplit.run()V
 2          0  OverlySplit.doSomething()V
 3       1736  (java/lang/Object)
 3        612  (java/util/ArrayList)
 0          0  001256d5_Thread-7
 1          0  OverlySplit.run()V
 2          0  OverlySplit.doSomething()V
 3       1771  (java/lang/Object)
 3        672  (java/util/ArrayList)
 0          0  001256d6_Thread-8
 1          0  OverlySplit.run()V
 2          0  OverlySplit.doSomething()V
 3       1749  (java/lang/Object)
 3        637  (java/util/ArrayList)
```

```java
public void doSomething() {
  try{
   synchronized (A) {
     if(sleeptype == 0){
         Thread.sleep(sleepTime);
     }else {
       Thread.sleep(0, sleepTime);
    }

   }
   synchronized (B) {
     if(sleeptype == 0){
         Thread.sleep(sleepTime);
     }else {
       Thread.sleep(0, sleepTime);
    }
    if(sleeptype == 0){
        Thread.sleep(sleepTime);
    }else {
      Thread.sleep(0, sleepTime);
   }
 }
   synchronized (A) {
     if(sleeptype == 0){
         Thread.sleep(sleepTime);
     }else {
       Thread.sleep(0, sleepTime);
    }
      }
```

Limitations:

- We cannot locate the java monitor that's being contended by multiple threads.
- If multiple methods are locked, you can see multiple methods here. We can follow up and down the call graph to see where a lock is being acquired and the path that led to it.
- But we don't know if it can locate if the method is called multiple times within the same class.
- We need a parser to read the log-rt file for better understanding.