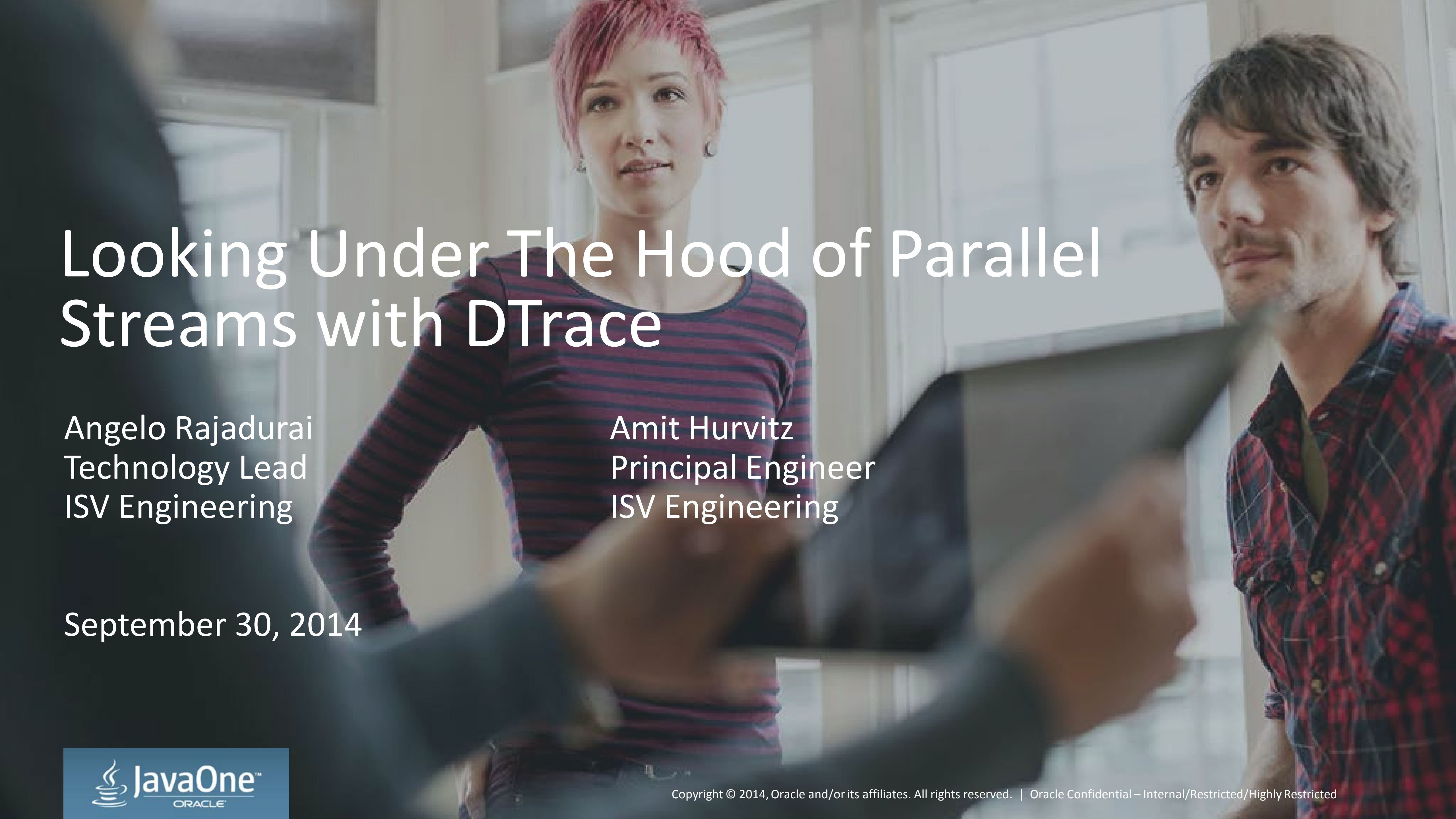
ORACLE®

JavaOne™

ORACLE®



# Looking Under The Hood of Parallel Streams with DTrace

Angelo Rajadurai
Technology Lead
ISV Engineering

Amit Hurvitz
Principal Engineer
ISV Engineering

September 30, 2014

## Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

## Agenda

## Program Agenda with Highlight

## What’s DTrace?

- Probably the most comprehensive monitoring tool...
- Dynamic instrumentation framework
- operating system and applications
- testing and production environments
- **The power of DTrace was often described as a tool that “allows you to ask arbitrary questions about what the system is doing, and get answers”**

## What’s DTrace? – cont.

- Zero performance impact when not in use
- Built for minimal impact when in use
- Completely safe; no way to cause panics, crashes, data corruption or pathological performance degradation
- Powerful data management primitives eliminate need for most post-processing

## DTrace – a quick overview

### D Language

```
probe description
 /predicate/
 {
   actions
 }
```

Description Which events we are interested in monitoring

Predicates (optional) When do we want to monitor the events

Actions (optional) What do we want to do when the above happens

One liner

```
# dtrace -n 'probe/predicate/{actions}
```

## DTrace – a quick overview

### Probes

- Programmable sensors (points of instrumentation) made available by providers placed all over the Solaris system
- provider:module:function:name
- tcp:ip:tcp_send:entry
- Syscall:::
- Providers: syscall,io,pid,profile, hotspot, tcp, udp, jdtrace...
- Modules: nfs, zfs, cpc, …
- Names: entry,return

## DTrace – a quick overview

### Predicates, Actions, Predefined Variables

- /cpu == 0/
  - /execname == “date”/
  - /ppid != 0 && arg0 != 0/
- Actions
  - Commands separated by “;”
  - trace(execname)
  - printf(“%s %s %s”, execname, probefunc, copyinstr(arg0));
- Predefined Variables
  - **execname, probefunc, pid, ppid, cpu, timestamp, arg0, arg1, ...**

## DTrace – an Example

### Who wrote a string to any file?

```
# cat method_wrote_this.d
syscall::write:entry
{
    str = copyinstr(arg1, arg2);
}

syscall::write:entry
/strstr(str, $$1) != NULL/
{
    printf("It's me, %s, pid %d, str=%s\n", execname, pid, str);
    jstack();
    exit(0);
}
```

## DTrace – an Example (cont.)

### Who wrote a string to any file?

```
# ./method_wrote_this.d foo
It's me, java, pid 672, str=foo

              libc.so.1`__write+0x8
              libjava.so`handleWrite+0x10
              libjava.so`writeBytes+0x1b8
              libjava.so`Java_java_io_FileOutputStream_writeBytes+0x48
              java/io/FileOutputStream.writeBytes([BIIZ)V*
              java/io/BufferedOutputStream.flush()V*
              java/io/PrintStream.write([BII)V*
              sun/nio/cs/StreamEncoder.writeBytes()V*
              sun/nio/cs/StreamEncoder.flushBuffer()V*
              java/io/PrintStream.write(Ljava/lang/String;)V*
              simpleloop/SimpleLoop.doo()V*
              simpleloop/SimpleLoop.coo()V*
              simpleloop/SimpleLoop.boo()V*
              simpleloop/SimpleLoop.aoo()V*
              simpleloop/SimpleLoop.main([Ljava/lang/String;)V
              StubRoutines (1)

libjvm.so`__1cJJavaCallsLcall_helper6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnG
Thread__v_+0xa50
              libjvm.so`jni_CallStaticVoidMethod+0x908
              libjli.so`JavaMain+0x770
              libc.so.1`_lwp_start
```

## DEMO

**Basic DTrace**

## DTrace Providers

| | | | | |
|---|---|---|---|---|
| **cpc** HW counters | **fbt** HW counters | **pfuinfo** HW counters | **io** disk operation | **lockstat** Lock statistics |
| mib MIB counters | profile Time-base interuppt firing | sched CPU scheduing | syscall System calls | sysinfo Kstat sys field statistics |
| vminfo Kstat sys field statistics | pid User processes | plockstat User-level synchronization | proc Process life cycle | perl Perl scripts tracing |
| ip Ip provider | iscsi Iscsi provider | nfsv3 Nfsv3 provider | nfsv4 Nfsv3 provider | srp Srp provider |
| tcp Tcp provider | udp Udp provider | dtrace Begin/end/error | sdt User probes | hotspot JVM probes |

- And You can create your own with SDT/JSDT

## Hotspot Built-In Probes

- vm-init-begin, vm-init-end, vm-shutdown
- thread-start, thread-end
- class-loaded, class-unloaded
- gc-begin, gc-end, mem-pool-gc-begin, mem-pool-gc-end
- method-compile-begin, method-compile-end
- compiled-method-load, compiled-method-unload

monitor-contended-enter, monitor-contended-entered, monitor-contended-exit, monitor-wait, monitor-waited, monitor-notify, monitor-notifyAll
method-entry, method-return
object-alloc

+ExtendedDTraceProbes

## Hotspot Built-In Probes

- Probes are in JVM
- So very useful for looking at JVM
- Not very useful for looking into Java code
- Overhead can be non-trivial
- Method invocation tracing has high overhead
- Solutions
  - Use Byte Code instrumentation – tools like BTrace
  - Java 7 and later has ways to instrument Java Apps to add probes

## Program Agenda with Highlight

1. Dtrace – a quick overview
2. **Java 8 Parallel Streams**
3. JDTrace
4. Demo
5. Future Plans

# Java 8 Parallel Streams

```
Long get occurencesNumber() {
        List<Document>dlist = AllDocuments.getList();
        dlist.parallelStream()
                .mapToLong(doc -> occurrencesCount(doc, searchedWord))
                .sum();
}
```

Generate a parallel stream

Call occrencesCount() for each element element

Map each result to long

Generate a parallel stream

## Program Agenda with Highlight

1. Dtrace – a quick overview
2. Watch Java 8 Parallel Streams
3. **JDTrace**
4. Demo
5. Future Plans

## JDTrace

### Introduction

- Java Method entry/return probes, line number probes, call probes
  - jdtrace:*className*:*methodName*:entry {}
  - jdtrace:*className*:*methodName*:*lineNumber* {}
  - jdtrace:*className*:*methodName*:call*MethodName* {}
- Zero impact when not activated
- Minimal impact when activated
  - Probe firing is instrumented in the target process when activated, deinstrumented when deactivated

## JDTrace

### Introduction

- Implemented by:
  - Java Statically Defined Probes (JSDT)
    - Java API for firing DTrace probes
  - Java Byte Code Instrumentation and Attach API
    - Uses ASM for byte code manipulations
  - No Prior Setting to the Target Application is Required
    - Just Java 7 or higher
  - Wraps Around dtrace
    - Takes same flags
    - Just instrument probe calls and invokes dtrace
    - Cleans instrumentation at the end

## JDTrace – an Example

### Get method execution time

```
# cat method_time.d
jsdt$target:simpleloop.SimpleLoop:foo:entry
/pid == $target/
{
        self->starttime = timestamp;
}

jsdt$target:simpleloop.SimpleLoop:foo:return
/self->starttime/
{
        @total[“total time(ns):”] = sum(timestamp - self->starttime);
        self->starttime = 0;
}

# jdtrace –s method_time.d –p process-id
 total time(ns):                                              6001590233
```

## JDTrace – an Example

### Consider this foo() method:

```
static int counter == 0;

private static void foo() {
        System.out.println("foo");
        try {
            Thread.sleep((counter++ % 10) == 0 ? 1000 : 0);
        } catch (InterruptedException ex) {
            Logger.getLogger(SimpleLoop.class.getName()).log(Level.SEVERE, null, ex);
        }
        goo();
}
```

**1/10 of the calls should sleep for a second**

# JDTrace - example

## Get method execution time distribution

```
# cat method_time.d
 …
          /* @total[“total time:”] = sum(timestamp - self->starttime); */
           @total["total time:"] = lquantize((timestamp - self->starttime) / 1000000, 0, 10000, 100);
…


# jdtrace –s method_time.d –p process-id
distribution time:
value  ------------- Distribution ------------- count
             < 0 |                                        0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@     31
             100 |                                        0
             200 |                                        0
             300 |                                        0
             400 |                                        0
             500 |                                        0
             600 |                                        0
             700 |                                        0
             800 |                                        0
             900 |                                        0
            1000 |@@@@                                    3
            1100 |                                        0
```

**(time is in milliseconds)**

## JDTrace - example

### Real Parallelism Level

```
String[] wordsIn(String line) {
              return line.trim().split("(\\s|\\p{Punct})+");
}


Long occurrencesCount(Document document, String searchedWord) {
        long count = 0;
        for (String line : document.getLines()) {
            for (String word : wordsIn(line)) {
                if (searchedWord.equals(word)) {
                    count = count + 1;
                }
            }
        }
        return count;
}
```

## JDTrace - example

### Real Parallelism Level – How much of my large machine does my app use?

```
# cat jmethod_on_cpu.d
BEGIN
{
        starttime = timestamp;
}

jsdt$target:wordcount.*.WordCounter:occurrencesCount:entry
{
        inmethod[tid] = 1;
        starttime = starttime ? starttime : timestamp;
}

jsdt$target:wordcount.*.WordCounter:occurrencesCount:return
{
        inmethod[tid] = 0;
}

profile-97
{
        @oncpu[pid == $target ? (inmethod[tid] ? "in method" : "other method") : "other process"] = count();
        @oncpu[“total polls”] = count();
}

END
{
        n_of_polls = (timestamp - starttime) * 97 / 1000000000;
        normalize(@oncpu, n_of_polls);
}
```

## JDTrace - example

### Real Parallelism Level – How Parallelized is the Core Method

Concurrent Running Statistic on the **256 HW Threads** of an Oracle T5-2 Server

```
# ./jdtrace_3 -x jstackstrsize=2048 -s jmethod_on_cpu.d –p pid
  other method                                                      8
  other process                                                    74
  in method                                                       161
  total polls                                                     243
```

# JDTrace - example

## Real Parallelism Level – Now let's add a scaling problem and run again

```
String[] wordsIn(String line) {
         String [] result;
         synchronized(this) {
                result = line.trim().split("(\\s|\\p{Punct})+");
         }
          return result;
}


Long occurrencesCount(Document document, String searchedWord) {
        long count = 0;
        for (String line : document.getLines()) {
            for (String word : wordsIn(line)) {
                if (searchedWord.equals(word)) {
                    count = count + 1;
                }
            }
        }
        return count;
}
```

## JDTrace - example

### Real Parallelism Level – Run the jdtrace script again

```
# ./jdtrace_3 -x jstackstrsize=2048 -s jmethod_on_cpu.d -p pid
  other method                                                     0
  in method                                                        1
  other process                                                  250
  total polls                                                    252


(previous good results:
  other method                                                     8
  other process                                                   74
  in method                                                      161
  total polls                                                    243
}
```

## DTrace - example

### Another Way to Watch Locking/Waiting – Off CPU tracing

```
# cat off-cpu.d
BEGIN
{
    start_timestamp = timestamp;
}

sched:::off-cpu
/pid == $target/
{
    self->ts = timestamp;
}

sched:::on-cpu
/self->ts && timestamp - self->ts < 5000000000/ /* 5 seconds wait threshold */
{
    @[jstack()] = sum(timestamp - self->ts);
}

END
{
    printf("elapsed time: %d\n", timestamp - start_timestamp);
    printa(@);
}
```

## DTrace - example

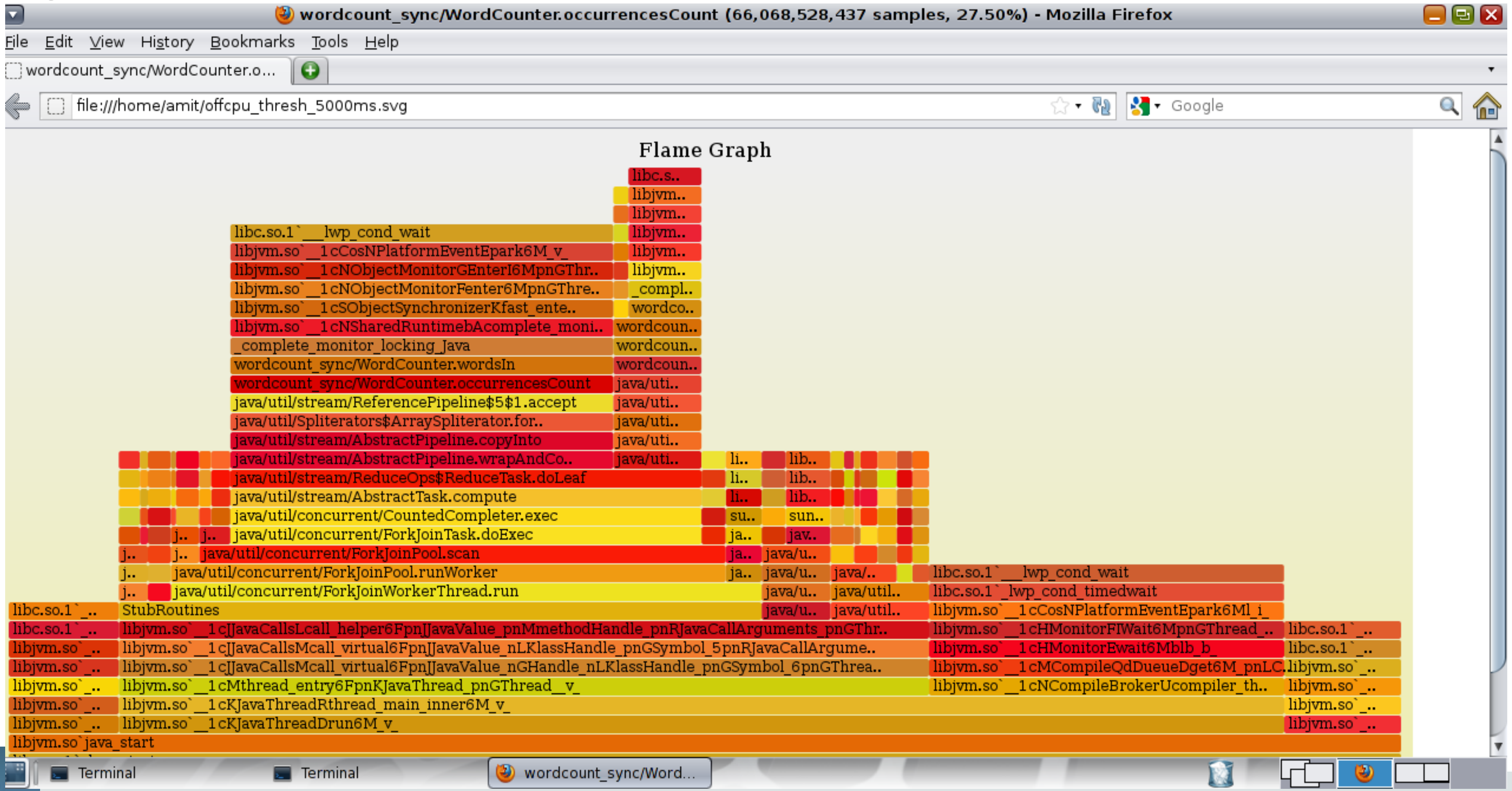
### A snippet from the output

```
…
wordcount_sync/WordCounter.wordsIn(Ljava/lang/String;)[Ljava/lang/String;*

wordcount_sync/WordCounter.occurrencesCount(Lwordcount_sync/Document;Ljava/lang/String;)Ljava/lang/Long;*
              java/util/stream/ReferencePipeline$5$1.accept(Ljava/lang/Object;)V*
              java/util/Spliterators$ArraySpliterator.forEachRemaining(Ljava/util/function/Consumer;)V*
              java/util/stream/AbstractPipeline.copyInto(Ljava/util/stream/Sink;Ljava/util/Spliterator;)V*
              java/util/stream/AbstractPipeline.wrapAndCopyInto(Ljava/util/stream/Sink;Ljava/util/Spliterator;)Ljava/util/stream/Sink;*
              java/util/stream/ReduceOps$ReduceTask.doLeaf()Ljava/lang/Object;*
              java/util/stream/AbstractTask.compute()V*
              java/util/concurrent/CountedCompleter.exec()Z*
              java/util/concurrent/ForkJoinTask.doExec()I*
              java/util/concurrent/ForkJoinPool.scan(Ljava/util/concurrent/ForkJoinPool$WorkQueue;I)I*
              java/util/concurrent/ForkJoinPool.runWorker(Ljava/util/concurrent/ForkJoinPool$WorkQueue;)V
…
        35908605737
```

## DTrace - example

### Off CPU tracing – What is my Application Thread Waiting for?

### Use Flame Graph to Show Results



wordcount_sync/WordCounter.occurrencesCount (66,068,528,437 samples, 27.50%) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
wordcount_sync/WordCounter.o...
file:///home/amit/offcpu_thresh_5000ms.svg
Google
Flame Graph
libc.s..
libjvm..
libc.so.1`__lwp_cond_wait
libjvm.so`__1cCosNPlatformEventEpark6M_v_
libjvm.so`__1cNObjectMonitorGEnterI6MpnGThr..
libjvm.so`__1cNObjectMonitorFenter6MpnGThre..
_compl..
libjvm.so`__1cSObjectSynchronizerKfast_ente..
wordco..
libjvm.so`__1cNSharedRuntimebAcomplete_moni..
wordcoun..
_complete_monitor_locking_Java
wordcount_sync/WordCounter.wordsIn
wordcount_sync/WordCounter.occurrencesCount
java/uti..
java/util/stream/ReferencePipeline$5$1.accept
java/util/Spliterators$ArraySpliterator.for..
java/util/stream/AbstractPipeline.copyInto
java/util/stream/AbstractPipeline.wrapAndCo..
java/util/stream/ReduceOps$ReduceTask.doLeaf
java/util/stream/AbstractTask.compute
java/util/concurrent/CountedCompleter.exec
java/util/concurrent/ForkJoinTask.doExec
java/util/concurrent/ForkJoinPool.scan
java/util/concurrent/ForkJoinPool.runWorker
java/util/concurrent/ForkJoinWorkerThread.run
li..
lib..
su..
sun..
ja..
jav..
java/u..
java/..
java/util..
libc.so.1`__lwp_cond_wait
libc.so.1`_lwp_cond_timedwait
libjvm.so`__1cCosNPlatformEventEpark6Ml_i_
libjvm.so`__1cHMonitorFIWait6MpnGThread_..
libjvm.so`__1cHMonitorEwait6Mblb_b_
libjvm.so`__1cMCompileQdDueueDget6M_pnLC..
libjvm.so`__1cNCompileBrokerUcompiler_th..
libc.so.1`_..
libjvm.so`_..
StubRoutines
libjvm.so`__1cJJavaCallsLcall_helper6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThr..
libjvm.so`__1cJJavaCallsMcall_virtual6FpnJJavaValue_nLKlassHandle_pnGSymbol_5pnRJavaCallArgume..
libjvm.so`__1cJJavaCallsMcall_virtual6FpnJJavaValue_nGHandle_nLKlassHandle_pnGSymbol_6pnGThrea..
libjvm.so`__1cMthread_entry6FpnKJavaThread_pnGThread__v_
libjvm.so`__1cKJavaThreadRthread_main_inner6M_v_
libjvm.so`__1cKJavaThreadDrun6M_v_
libjvm.so`java_start
Terminal
Terminal
wordcount_sync/Word...

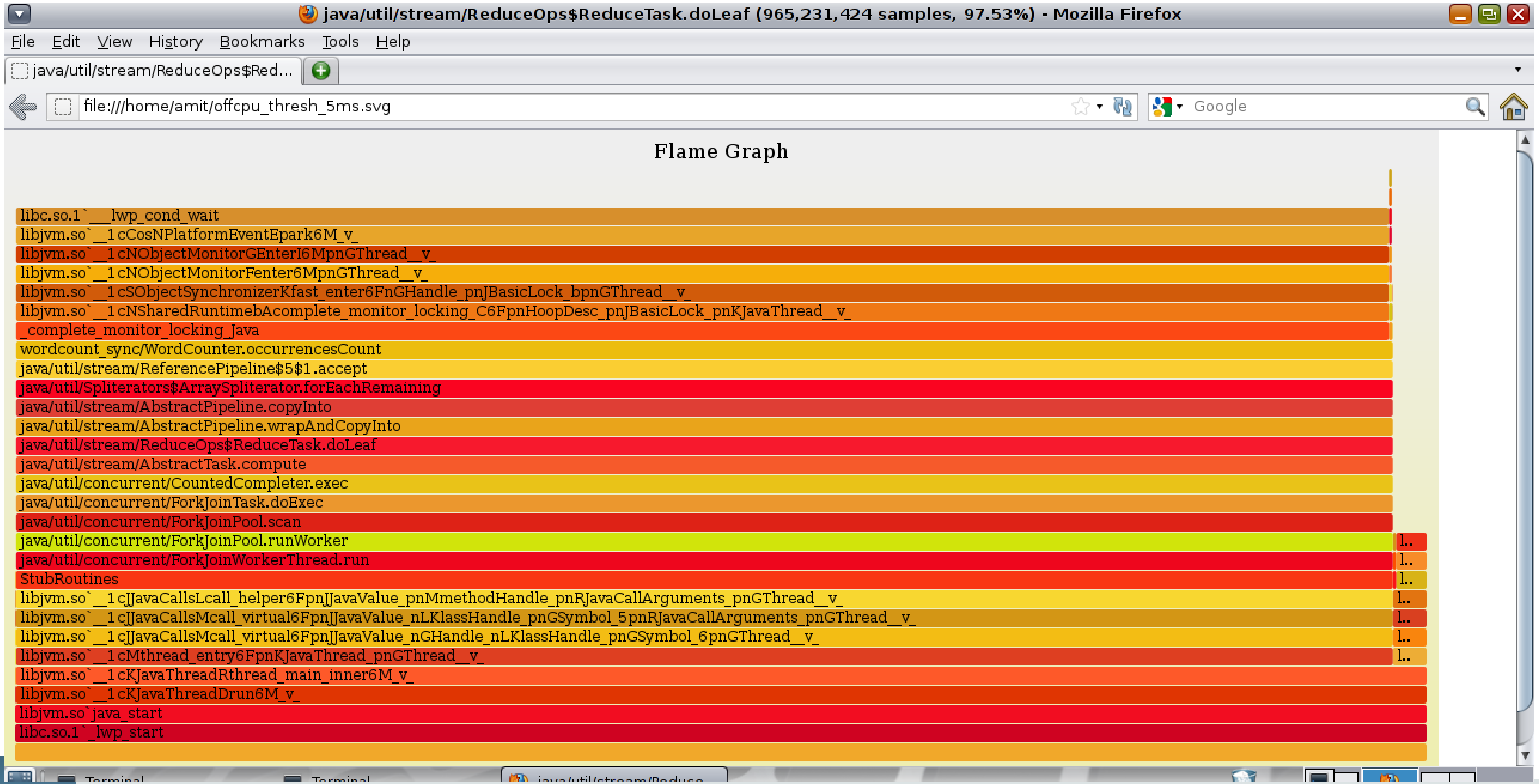
## DTrace - example

### If we Reduce Wait Threshold to 5 Milliseconds

```
# cat off-cpu.d
BEGIN
{
    start_timestamp = timestamp;
}

sched:::off-cpu
/pid == $target/
{
    self->ts = timestamp;
}

sched:::on-cpu
/self->ts && timestamp - self->ts < 5000000/ /* 5 milliseconds wait threshold */
{
    @[jstack()] = sum(timestamp - self->ts);
}

END
{
    printf("elapsed time: %d\n", timestamp - start_timestamp);
    printa(@);
}
```

# DTrace - example

## We’ll get this



java/util/stream/ReduceOps$ReduceTask.doLeaf (965,231,424 samples, 97.53%) - Mozilla Firefox

File Edit View History Bookmarks Tools Help
java/util/stream/ReduceOps$Red...
Google

file:///home/amit/offcpu_thresh_5ms.svg

Flame Graph

```
libc.so.1`__lwp_cond_wait
libjvm.so`__1cCosNPlatformEventEpark6M_v_
libjvm.so`__1cNObjectMonitorGEnterI6MpnGThread__v_
libjvm.so`__1cNObjectMonitorFenter6MpnGThread__v_
libjvm.so`__1cSObjectSynchronizerKfast_enter6FnGHandle_pnJBasicLock_bpnGThread__v_
libjvm.so`__1cNSharedRuntimebAcomplete_monitor_locking_C6FpnHoopDesc_pnJBasicLock_pnKJavaThread__v_
_complete_monitor_locking_Java
wordcount_sync/WordCounter.occurrencesCount
java/util/stream/ReferencePipeline$5$1.accept
java/util/Spliterators$ArraySpliterator.forEachRemaining
java/util/stream/AbstractPipeline.copyInto
java/util/stream/AbstractPipeline.wrapAndCopyInto
java/util/stream/ReduceOps$ReduceTask.doLeaf
java/util/stream/AbstractTask.compute
java/util/concurrent/CountedCompleter.exec
java/util/concurrent/ForkJoinTask.doExec
java/util/concurrent/ForkJoinPool.scan
java/util/concurrent/ForkJoinPool.runWorker                      l..
java/util/concurrent/ForkJoinWorkerThread.run                    l..
StubRoutines                                                     l..
libjvm.so`__1cJJavaCallsLcall_helper6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v_   l..
libjvm.so`__1cJJavaCallsMcall_virtual6FpnJJavaValue_nLKlassHandle_pnGSymbol_5pnRJavaCallArguments_pnGThread__v_   l..
libjvm.so`__1cJJavaCallsMcall_virtual6FpnJJavaValue_nGHandle_nLKlassHandle_pnGSymbol_6pnGThread__v_   l..
libjvm.so`__1cMthread_entry6FpnKJavaThread_pnGThread__v_         l..
libjvm.so`__1cKJavaThreadRthread_main_inner6M_v_
libjvm.so`__1cKJavaThreadDrun6M_v_
libjvm.so`java_start
libc.so.1`_lwp_start
```

## DTrace - example

### We can also get wait time distribution of each stack trace (a small change to the script)

```
# cat off-cpu.d
BEGIN
{
    start_timestamp = timestamp;
}

sched:::off-cpu
/pid == $target/
{
    self->ts = timestamp;
}

sched:::on-cpu
{
    @[jstack()] = quantize(timestamp - self->ts);
}

END
{
    printf("elapsed time: %d\n", timestamp - start_timestamp);
    printa(@);
}
```

## DTrace - example

### We can also get wait time distribution of each stack trace (a small change to the script)

```
…
wordcount_sync/WordCounter.wordsIn(Ljava/lang/String;)[Ljava/lang/String;*
              wordcount_sync/WordCounter.occurrencesCount(Lwordcount_sync/Document;Ljava/lang/String;)Ljava/lang/Long;
              wordcount_sync/WordCounter.lambda$countOccurrencesByPStreams$2(Ljava/lang/String;Lwordcount_sync/Document;)J
              wordcount_sync/WordCounter$$Lambda$2.applyAsLong(Ljava/lang/Object;)J
              java/util/stream/ReferencePipeline$5$1.accept(Ljava/lang/Object;)V
…
value  ------------- Distribution ------------- count
          262144 |                                         0
          524288 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1
         1048576 |                                         0

libc.so.1`___lwp_cond_wait+0x4
              libc.so.1`_lwp_cond_timedwait+0x20
              libjvm.so`__1cCosNPlatformEventEpark6Ml_i_+0x254
              libjvm.so`__1cHMonitorFIWait6MpnGThread_l_i_+0x11c
              libjvm.so`__1cHMonitorEwait6Mblb_b_+0x2b4
              libjvm.so`__1cMCompileQdDueueDget6M_pnLCompileTask__+0x15c
              libjvm.so`__1cNCompileBrokerUcompiler_thread_loop6F_v_+0x258
              libjvm.so`__1cKJavaThreadRthread_main_inner6M_v_+0x8c
              libjvm.so`__1cKJavaThreadDrun6M_v_+0x400
              libjvm.so`java_start+0x35c
              libc.so.1`_lwp_start

           value  ------------- Distribution ------------- count
4503599627370496 |                                         0
9007199254740992 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 144
18014398509481984 |                                         0
```

## Program Agenda with Highlight

## Parallel Streams Tracing – an Example

### Actual Parallelism on a Massively Scaled Platform – Oracle T5-2

- Count Word Occurrences in Files (again):
  - Parallel read
    - `listOfDocs.parallelStream().forEach(doc -> doc.readLines());`
  - Now assume we want to read look for word occurrences only on the end (last 10%) of the files
    - Looks like reading the files should be much quicker
      - Using RandomAccessFile to start reading from the 9/10*fileLength position
    - But…

Demo

## Parallel Streams Tracing – an Example

### Actual Parallelism on a Massively Scaled Platform – Oracle T5-2

– Before:

```
listOfDocs.parallelStream().forEach(doc -> doc.readLines());
public void readLines() {
   …
   lines = new LinkedList<>();
   RandomAccessFile raf = new RandomAccessFile(file, "rw");
   raf.seek(raf.length() * 9 / 10);
   String line = line = raf.readLine();
   while (line != null) {
       lines.add(line);
       line = raf.readLine();
   }
    …
 }
```

# Parallel Streams Tracing – an Example

## Actual Parallelism on a Massively Scaled Platform – Oracle T5-2

– Before:

```
    listOfDocs.parallelStream().forEach(doc -> doc.readLines());
public void readLines() {
    …
    lines = new LinkedList<>();
    RandomAccessFile raf = new RandomAccessFile(file, "rw");
    BufferedReader br = new BufferedReader(new FileReader(raf.getFD()));
    raf.seek(raf.length() * 9 / 10);
    String line = br.readLine();
    while (line != null) {
        lines.add(line);
        line = br.readLine();
    }
  …
  }
```

## Program Agenda with Highlight

## JDTrace – Plans

### Want to help?

- JDTrace Project
  - Maybe the name will change…
  - A java.net project to come (soon)
- jdtrace Provider Probes:
  - jdtrace$target:*className*:*methodName*:entry|return {}
    - Already done
  - jdtrace$target:*className*:*methodName*:lineNumber {}
  - jdtrace$target:*className*:*methodName*:call_*className*:*methodName* *{}*
  - Make method arguments visible to jdtrace actions

## JDTrace Questions / Suggestions?

Email:

Amit Hurvitz amit.hurvitz@oracle.com

Angelo Rajadurai angelo.rajadurai@oracle.com

# Thank You!

Hardware and Software

Engineered to Work Together

JavaOne™

ORACLE®

ORACLE®