

Performance analysis of Java web applications

OKTECH Profiler



István Soós
istvan.soos@oktech.hu

Agenda

At the end of the presentation, you will be aware of ...

- ... the importance of application profiling.
- ... various performance measurement techniques.
- ... the impact of profiling and accepting the limits.

- ... the actual status of OKTECH Profiler.
- ... planned features and developments.

Measuring... why?

We are just fine, because...

- "Memory is cheap..."
- "A little bigger machine will manage the load"...
- "Hosting is cheap..."
- "Just a new machine in the cloud..."

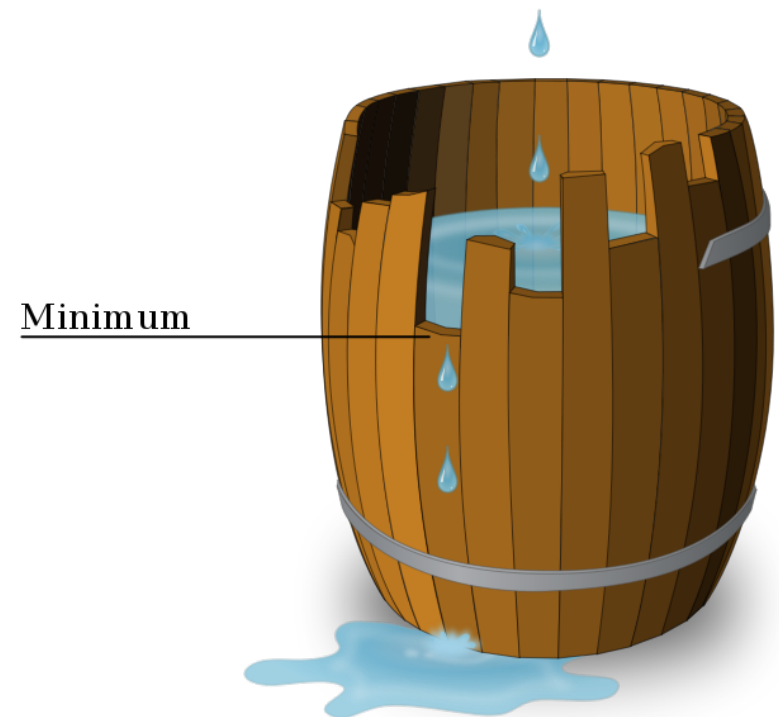
Is it really that cheap?

- There are direct costs (or savings) on the performance.
- If there is competition, someone will benchmark to it.
- The client's time is always expensive, don't make him wait.

Liebig's law of the minimum

growth is controlled not by the total of resources available,
but by the scarcest resource (limiting factor)

(Carl Sprengel (1828) and Justus von Liebig)



Measuring... what?

You can always ask questions about the application:

- How much percentage does the ... processing take?
- Why is this page so slow for the users?
- Do we have an infrastructural bottleneck here?
- Are our synchronization routines perfect?
- Where is the query that involves the most post-processing while reading the results from the database?
- We do have a cache, is it used correctly?
- ...
- What is happening ... just right now?

Profiling methods

Instrumentation

- Manual (coding, coding and coding...)
- Compile-time (pre-binary)
- Binary translation (post-compile)
- Runtime instrumentation (pre-run)
- Runtime injection (on-the-fly)

Sampling

- Execution trace (where are we now?)
- Monitor values (memory, cpu times...)

Hypervisor

- "Virtualization" or "Simulator" (step the clock)

Instrumentation

```
protected void someMethod() {  
    long start = System.nanoTime();  
    // ...  
    long end = System.nanoTime();  
    Trace.report("someMethod()", (end-start) );  
}
```

```
protected void otherMethod() {  
    Trace.reportStart("otherMethod");  
    try {  
        // ...  
    } finally {  
        Trace.reportEnd("otherMethod");  
    }  
}
```

Systematic error of instrumentation

```
public void a() { b(); }  
public void b() { c(); }  
public void c() { for (int i =0; i<100; i++) d(i); }  
...
```

The systematic error:

- measurement times are added to the executions
- on multiple level, this accumulates
- there is no way to eliminate all measurement time
- will be repeatedly the same on each measurements
- distorts on lower timing values impact the overall result more

Sampling

1. Do not specify what to measure, just give me one actual state
 - `StackTraceElement[] Thread.getStackTrace()`
 2. Repeat this couple of times.
 3. Explore the depth of mathematical statistics :)
- Adjustable, typically smaller overhead, but
 - Natural uncertainty
 - Different kind of statistics

Sampling: the stack trace

```
com.sun.jndi.rmi.registry.RegistryContext.lookup(RegistryContext.java:101)
com.sun.jndi.toolkit.url.GenericURLContext.lookup(GenericURLContext.java:185)
javax.naming.InitialContext.lookup(InitialContext.java:392)
javax.management.remote.rmi.RMIConnector.findRMIServerJNDI(RMIConnector.java:1886)
javax.management.remote.rmi.RMIConnector.findRMIServer(RMIConnector.java:1856)
javax.management.remote.rmi.RMIConnector.connect(RMIConnector.java:257)
javax.management.remote.rmi.RMIConnector.connect(RMIConnector.java:338)
javax.management.remote.JMXConnectorFactory.connect(JMXConnectorFactory.java:248)
hu.oktech.profiler.runtime.remote.RemoteJmxRuntime.start(RemoteJmxRuntime.java:65)
hu.oktech.profiler.runtime.remote.RemoteJmxProfiler.main(RemoteJmxProfiler.java:42)
```

Further observer effects

JVM effects:

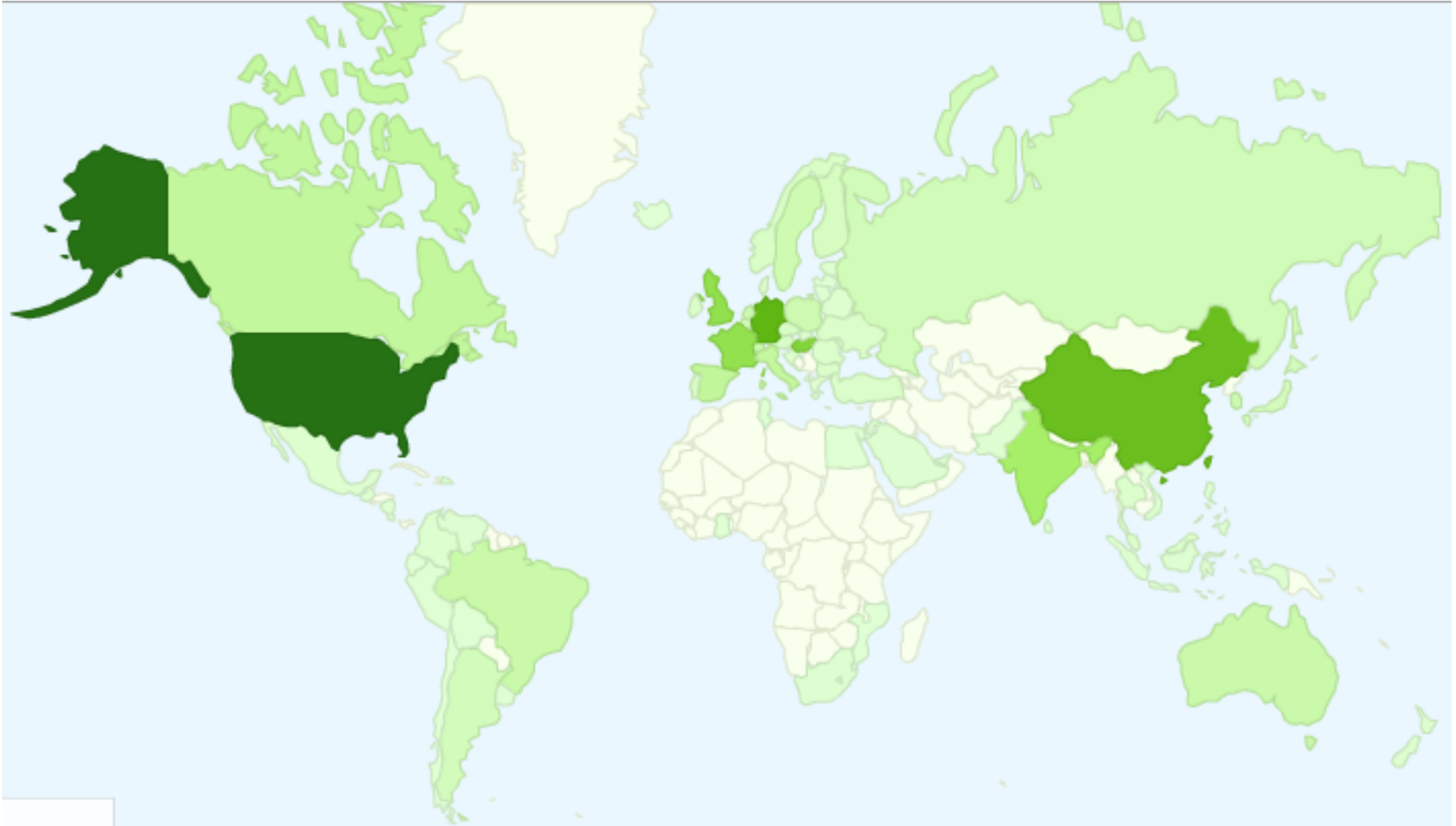
- Running anything attached to a JVM will affect GC times, threads, IO stats, sometimes synchronization
- Instrumentation will make HotSpot different (less optimized)

CPU effects:

- Modern CPUs do advanced caching and parallel processing of instructions
- Sometimes low-level instructions are not processed in the designated order

OKTECH Profiler

- <http://code.google.com/p/oktech-profiler/>



OKTECH Profiler

- <http://code.google.com/p/oktech-profiler/>
- Local Java-agent, remote JMX connections
- Sampling profiler
- Instrumentation profiler
- Simple tree statistics
 - Text and XML output

Planned features:

- Monitoring, alerting
- More statistics, more output formats
- More precise instrumentation and sampling, probes
- Commercial support: <http://oktech.hu/>

Demo snippets

```
java -jar hu.oktech.profiler-runtime.jar \  
  remote.jmx.url=service:jmx:rmi:///jndi/rmi://localhost:8686/jmxrmi \  
  remote.jmx.user=admin \  
  remote.jmx.password=adminadmin
```

```
java -jar hu.oktech.profiler-runtime.jar prop=remote.properties
```

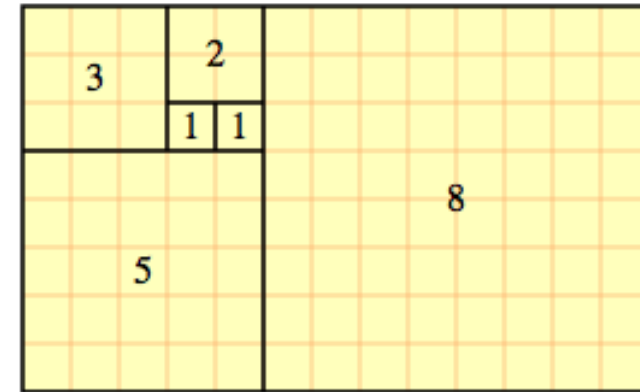
```
java -javaagent:hu.oktech.profiler-runtime.jar=prop=local.properties \  
  -cp ... some.Main arg1 arg2
```

```
instr.class=my.company.*!,org.apache.*,-org.apache.catalina.*  
#instr.methods=
```

```
java -jar hu.oktech.profiler-report.jar \  
  input=tmp/profiler/2009-09-19-09-02-20.dump
```

Case study: fibonacci numbers

```
public BigInteger fibonacci(int n) {  
    if (n == 0) return BigInteger.ZERO;  
    if (n == 1) return BigInteger.ONE;  
    return fibonacci(n - 1).add(fibonacci(n - 2));  
}
```



This extreme scenarion can not be profiled in any sane way:

- Instrumentation
 - Exponential number of events - no go
 - Systematic error would be obvious
- Sampling
 - Not consistent stack depth
- Probe: Time depends on parameter

Workaround:

- Profile the caller method (or create a wrapper)

Case study: fibonacci numbers

```
fibonacci.FibonacciUtils.fibonacci() cnt=2720  
|-fibonacci.FibonacciUtils.fibonacci() cnt=2579/cntP=94.816%  
|-java.math.BigInteger.add() cnt=15/cntP=0.551%
```

However:

- total sampling of root fibonacci(): 141
- BigInteger.add(): $15/141 = 10.6\%$

Case study: fibonacci numbers

```

-fibonacci.FibonacciUtils.fibonacci() cnt=811/cntT=19.618%/cntP=100.0% sys=419020.305ms/sysr=0.0%
|-fibonacci.FibonacciUtils.fibonacci() cnt=811/cntT=19.618%/cntP=100.0% sys=419020.305ms/sysr=0.0%
| |-fibonacci.FibonacciUtils.fibonacci() cnt=807/cntT=19.521%/cntP=99.507% sys=415972.563ms/sysr=0.0%
| | |-fibonacci.FibonacciUtils.fibonacci() cnt=805/cntT=19.473%/cntP=99.752% sys=414455.05ms/sysr=0.0%
| | | |-fibonacci.FibonacciUtils.fibonacci() cnt=799/cntT=19.328%/cntP=99.255% sys=40993.5ms/sysr=0.0%
| | | | |-fibonacci.FibonacciUtils.fibonacci() cnt=797/cntT=19.279%/cntP=99.75% sys=40993.5ms/sysr=0.0%
| | | | | |-fibonacci.FibonacciUtils.fibonacci() cnt=783/cntT=18.94%/cntP=98.243% sys=40993.5ms/sysr=0.0%
| | | | | | |-fibonacci.FibonacciUtils.fibonacci() cnt=753/cntT=18.215%/cntP=96.169% sys=40993.5ms/sysr=0.0%
| | | | | | | |-fibonacci.FibonacciUtils.fibonacci() cnt=683/cntT=16.522%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | |-fibonacci.FibonacciUtils.fibonacci() cnt=584/cntT=14.127%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | |-fibonacci.FibonacciUtils.fibonacci() cnt=451/cntT=10.91%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | |-fibonacci.FibonacciUtils.fibonacci() cnt=326/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | |-fibonacci.FibonacciUtils.fibonacci() cnt=137/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | |-fibonacci.FibonacciUtils.fibonacci() cnt=71/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | |-fibonacci.FibonacciUtils.fibonacci() cnt=4/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=5/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=1/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | |-fibonacci.FibonacciUtils.fibonacci() cnt=3/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=3/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=28/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=1/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=66/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=76/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=4/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=125/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=133/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=2/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=70/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=4/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=4/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=70/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=3/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=30/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=8/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%
| | | | | | | | | | | | | | | | | | | | | | | |-java.math.BigInteger.add() cnt=2/cntT=7.88%/cntP=90.0% sys=40993.5ms/sysr=0.0%

```



Case study: fibonacci numbers

```
fibonacci.FibonacciUtils.fibonacci() cnt=15963  
|-fibonacci.FibonacciUtils.fibonacci() cnt=15152/cntP=94.92%  
|-java.math.BigInteger.add() cnt=628/cntP=3.934%
```

However:

- total sampling of root fibonacci(): 811
- BigInteger.add(): $628/811 = 77.43\%$

Increased sampling numbers do increase precision, although the numbers are not really reliable in this case.

Case study: Webspaces portal



Webspaces portal = Sun's Liferay bundle

Complex infrastructure:

- Servlets, Spring, Hibernate, Ehcache, MySQL...
- Performance benchmark with JMeter, database profiling tools, JConsole and OKTECH Profiler

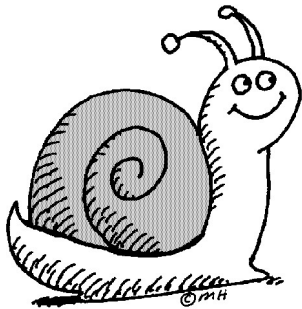
Use cases:

- Support given theories (where is the time spent?)
- Check cache configuration changes

Case study: Webspaces portal

- Symptom: document library seemed to be slow
- Profiler result: 51 of 69 (73.9%) samples in DB read
- Solution: database profiling was required

```
com.liferay.portlet.documentlibrary.service.impl.DLFileEntryLocalServiceImpl.getFileAsStream() cnt=69/cntT=9.223372036854776E15%/cntP=100.0% sys=316237.8
com.liferay.portlet.documentlibrary.service.persistence.DLFileEntryPersistenceImpl.update() cnt=52/cntT=9.223372036854776E15%/cntP=75.362% sys=209432
com.liferay.portlet.documentlibrary.service.persistence.DLFileEntryPersistenceImpl.updateImpl() cnt=52/cntT=9.223372036854776E15%/cntP=100.0% sys
-com.liferay.portal.service.persistence.BatchSessionUtil.update() cnt=52/cntT=9.223372036854776E15%/cntP=100.0% sys=209432.264ms/sysr=0.0ms/sy
|-com.liferay.portal.service.persistence.BatchSessionImpl.update() cnt=52/cntT=9.223372036854776E15%/cntP=100.0% sys=209432.264ms/sysr=5483
|-com.liferay.portal.dao.orm.hibernate.SessionImpl.flush() cnt=51/cntT=9.223372036854776E15%/cntP=98.077% sys=203385.436ms/sysr=0.0ms/s
|-com.liferay.portal.dao.orm.hibernate.LiferaySession.flush() cnt=51/cntT=9.223372036854776E15%/cntP=100.0% sys=203385.436ms/sysr=0
|-org.hibernate.impl.SessionImpl.flush() cnt=51/cntT=9.223372036854776E15%/cntP=100.0% sys=203385.436ms/sysr=0.0ms/sysP=100.0%
|-org.hibernate.event.def.DefaultFlushEventListener.onFlush() cnt=51/cntT=9.223372036854776E15%/cntP=100.0% sys=203385.436m
|-org.hibernate.event.def.AbstractFlushingEventListener.performExecutions() cnt=51/cntT=9.223372036854776E15%/cntP=100.0%
|-org.hibernate.engine.ActionQueue.executeActions() cnt=51/cntT=9.223372036854776E15%/cntP=100.0% sys=203385.436ms/
|-org.hibernate.engine.ActionQueue.executeActions() cnt=51/cntT=9.223372036854776E15%/cntP=100.0% sys=203385.43
|-org.hibernate.jdbc.AbstractBatcher.executeBatch() cnt=51/cntT=9.223372036854776E15%/cntP=100.0% sys=20338
|-org.hibernate.jdbc.BatchingBatcher.doExecuteBatch() cnt=51/cntT=9.223372036854776E15%/cntP=100.0% sys
|-com.mchange.v2.c3p0.impl.NewProxyPreparedStatement.executeBatch() cnt=51/cntT=9.223372036854776E15%
|-com.mysql.jdbc.PreparedStatement.executeBatch() cnt=51/cntT=9.223372036854776E15%/cntP=100.0%
|-com.mysql.jdbc.PreparedStatement.executeBatchSerially() cnt=51/cntT=9.223372036854776E15%/c
|-com.mysql.jdbc.PreparedStatement.executeUpdate() cnt=51/cntT=9.223372036854776E15%/cntP
|-com.mysql.jdbc.PreparedStatement.executeInternal() cnt=51/cntT=9.223372036854776E15
|-com.mysql.jdbc.Connection.execSQL() cnt=51/cntT=9.223372036854776E15%/cntP=100.
|-com.mysql.jdbc.MySQLIO.sqlQueryDirect() cnt=51/cntT=9.223372036854776E15%/c
|-com.mysql.jdbc.MySQLIO.sendCommand() cnt=51/cntT=9.223372036854776E15%/
|-com.mysql.jdbc.MySQLIO.checkErrorPacket() cnt=51/cntT=9.22337203685
|-com.mysql.jdbc.MySQLIO.reuseAndReadPacket() cnt=51/cntT=9.22337
|-com.mysql.jdbc.MySQLIO.readFully() cnt=51/cntT=9.2233720368
|-com.mysql.jdbc.util.ReadAheadInputStream.read() cnt=51/
|-com.mysql.jdbc.util.ReadAheadInputStream.readFromUnd
|-com.mysql.jdbc.util.ReadAheadInputStream.fill()
|-java.net.SocketInputStream.read() cnt=51/cn
|-java.net.SocketInputStream.socketRead0()
```



Case study: Webspaces portal

- Symptom: cache change doesn't affect results
- Profiler instrumentation of ehcache in different scenarios

```
| -JVM.instrumentation() cnt=0/cntT=0.0%/cntP=0.0%
|-com.liferay.portlet.documentlibrary.service.impl.DLFileEntryLocalServiceImpl.getFileAsStr
| |-com.liferay.portlet.documentlibrary.service.persistence.DLFileEntryPersistenceImpl.up
| | |-com.liferay.portlet.documentlibrary.service.persistence.DLFileEntryPersistenceImp
| | | |-net.sf.ehcache.hibernate.EhCache.put() cnt=100/cntT=9.223372036854776E15%/cn
| | | | |-net.sf.ehcache.Cache.put() cnt=100/cntT=9.223372036854776E15%/cntP=100.0
| | | | | |-net.sf.ehcache.Cache.put() cnt=100/cntT=9.223372036854776E15%/cntP=1
| | | | | | |-net.sf.ehcache.Cache.applyDefaultsToElementWithoutLifespanSet()
| | | | | | | |-net.sf.ehcache.Element.setEternal() cnt=100/cntT=9.223372036
| | | | | | | |-net.sf.ehcache.config.CacheConfiguration.getTimeToLiveSeconds
| | | | | | | |-net.sf.ehcache.Element.setTimeToLive() cnt=100/cntT=9.223372
| | | | | | | |-net.sf.ehcache.config.CacheConfiguration.getTimeToIdleSeconds
| | | | | | | |-net.sf.ehcache.Element.setTimeToIdle() cnt=100/cntT=9.223372
| | | | | | | |-net.sf.ehcache.config.CacheConfiguration.isEternal() cnt=100
| | | | | | | |-net.sf.ehcache.Element.isLifespanSet() cnt=100/cntT=9.223372
| | | | | | -net.sf.ehcache.event.RegisteredEventListeners.notifyElementUpdate
| | | | | | | |-net.sf.ehcache.event.RegisteredEventListeners.hasCacheEventLi
| | | | | -net.sf.ehcache.Cache.isElementInMemory() cnt=100/cntT=9.22337203
| | | | | | |-net.sf.ehcache.store.MemoryStore.containsKey() cnt=100/cntT=
| | | | | -net.sf.ehcache.store.MemoryStore.put() cnt=100/cntT=9.2233720368
| | | | | | |-net.sf.ehcache.Element.getObjectKey() cnt=100/cntT=9.2233720
| | | | | | | |-net.sf.ehcache.store.LruMemoryStore$SpoolingLinkedHashMap.rem
| | | | | | | | |-net.sf.ehcache.store.LruMemoryStore$SpoolingLinkedHashMap
| | | | | | | | | |-net.sf.ehcache.store.MemoryStore.isFull() cnt=1/cntT
| | | | | | | | | | |-net.sf.ehcache.Cache.getMaxElementsInMemory() cn
| | | | | | | | | | | |-net.sf.ehcache.config.CacheConfiguration.getM
| | | | | | | | | -net.sf.ehcache.Element.isExpired() cnt=1/cntT=9.2233
| | | | | | | | | | |-net.sf.ehcache.Element.getExpirationTime() cnt=1
```



Case study: OKTECH Profiler

Eating our own dogfood...

- 1.0's tree report was very slow
- We have profiled it with sampling
- It turned out it spends most of the time reading the input
- We have missed BufferedInputStream
- After the rewrite, it is much faster

With OKTECH Profiler...

... we can see and measure:

- Method-level performance characteristics (both)
- Possible locking and synchronization bottlenecks (sampling)
- Mostly acceptable method timings (instr.)

... we cannot see and measure:

- Parameter-level timings (e.g. per-parameter times)
- Large memory consumption areas

Unfortunately it doesn't answer the management's questions directly, but we will work on that one too :)

Q & A

At the end of the presentation, you are aware of ...

- ... the importance of application profiling.
- ... various performance measurement techniques.
- ... the impact of profiling and accepting the limits.

- ... the actual status of OKTECH Profiler.
- ... planned features and developments.