Performance Profiling (focused on Java SE)

Topics

- Profiling tools
- CPU profiling tips
- Lock contention profiling tips
- When to use what tool?
- Identifying problem pattens (Anti-patterns)

Performance Profiling Tools

Tools For Profiling Java applications

- Free tools
  - NetBeans Profiler
    - http://www.netbeans.org
  - VisualVM
    - New kid in the block
  - Sun Studio Collector / Analyzer (Solaris & Linux only)
    - http://developers.sun.com/sunstudio/downloads
- Commercial Profilers
  - JProbe
  - OptimizeIt
  - YourKit

Free Profilers : NetBeans Profiler

- CPU performance profiling using byte code instrumentation
- Low overhead profiling through selective profiling
  - Select specific method(s) for profiling
- Supported platforms; Solaris (SPARC & x86), Linux, Windows and Mac OS X
- Requires HotSpot JDK 5 or later
- Included out-of-the-box in NetBeans IDE 6.0

Free Profilers : Sun Studio

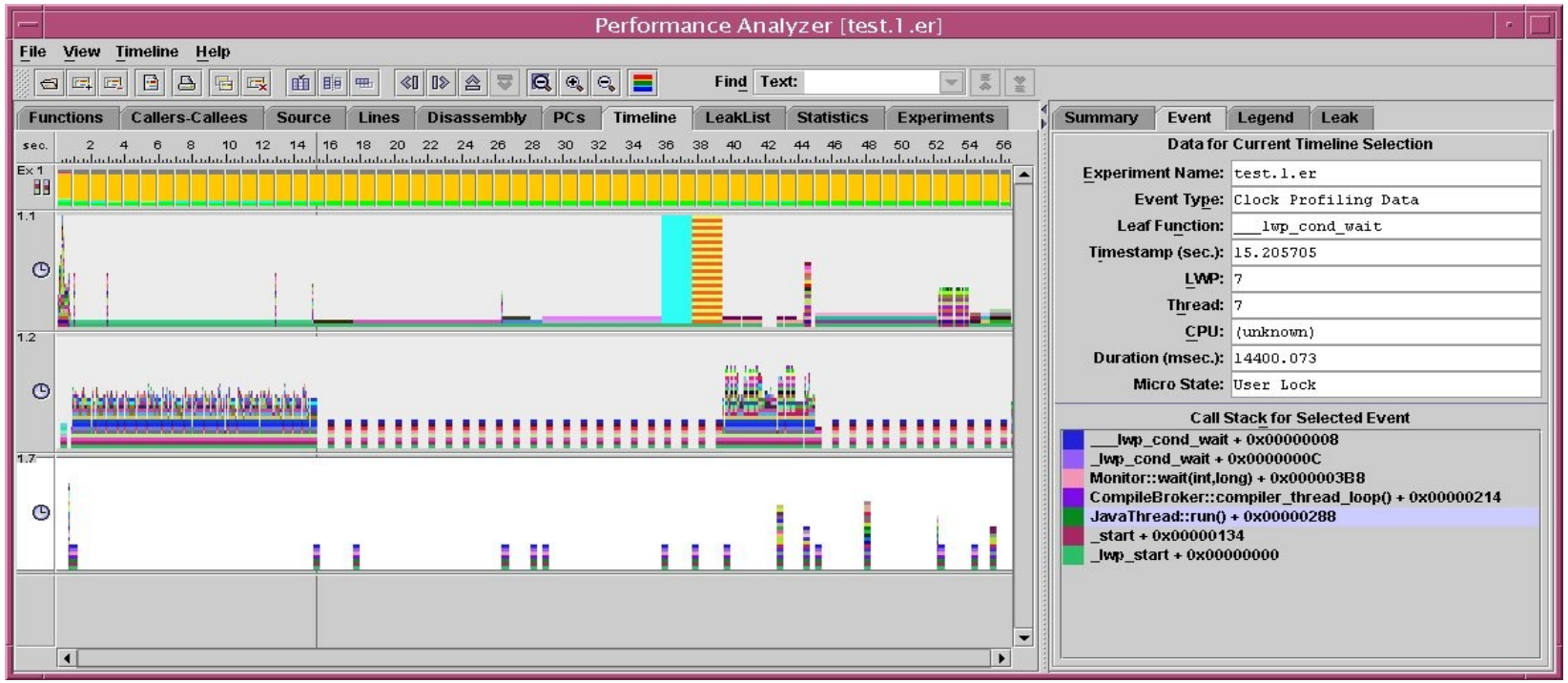
- Sun Studio Collector / Analyzer
  - Statistical CPU profiling using JVMTI
    - Can specify sampling interval, default 1 sec
    - User and sys cpu time
    - Inclusive or exclusive method times
  - Time spent in locks
  - View Java byte code in User Mode & Machine Mode
  - View JIT compiler generated assembly code in Expert Mode
  - Supports specific CPU counter collection
  - Supported platforms; Solaris (SPARC & x86) and Linux

Free Profilers : Sun Studio

- Sun Studio Collector / Analyzer
  - Easily invoked with 'collect -j on' prefixed to Java command line.
  - Requires HotSpot JDK 5 or later
  - Additional Info:
    - http://developers.sun.com/solaris/articles/perftools.html
    - http://developers.sun.com/solaris/articles/javapps.html
    - http://developers.sun.com/solaris/articles/profiling_websphere.html

Free Profilers : Sun Studio

- View 'collected' data, in Analyzer GUI



- Or, command line 'er_print'

CPU Profiling Tips

Profiling Tips : CPU Profiling

- CPU profiling provides information about where an application is spending most of its time.
- When is CPU profiling needed or beneficial?
  - Poor application throughput measured against a predetermined target
  - Saturated cpu utilization
  - High sys or kernel cpu utilization
  - High lock contention
  - To a lesser extent, idle cpu or poor application scalability

Profiling Tips : Strategies

- Approaches which work best for CPU profiling
  - Start with holistic approach to isolate major cpu consumers or hot methods.
    - Look at methods with high usr and/or sys cpu usage.
    - Look at both inclusive and exclusive method times.
    - Looking at inclusive times may help identify a change in implementation or design could be a good corrective approach.
    - Looking at exclusive times focus on specific implementation details within a method.

Profiling Tips : Strategies

- Some profilers such as NetBeans Profiler allows you profile a subset of an application.
  - Approach can be useful when or if profiling the entire application is very intrusive or severely disturbs application's performance.
  - If holistic approach is not possible or painful, then profiling suspected subsets of an application is good approach.
- DTrace scripts can also be effective

Profiling Tips : Which Product

- If you need to profile the entire application (instead of portion of the application)
  - Sun Studio Collector works well
    - 1 second default sampling rate
    - Easy to setup, just prepend 'collect -j on' to java command line.
    - Can fine tune sampling rate.
    - Can direct output to specified file name.
  - DTrace scripting
    - Can customize to target specific areas.
    - May require DTrace scripting expertise to author the script.

Profiling Tips : Which Product

- Profiling portions of applications
  - NetBeans Profiler works very well
    - Can easily configure which classes or packages to profile, (include or !include).
    - Easy to setup if application is setup as a NetBeans Project.
    - Remote or local profiling
    - Can view profiling as application is running.
    - Can compare profile against another profile.
  - DTrace scripting
    - Customize to target specific portions.
    - May require DTrace scripting expertise to author the script.

Lock Contention Profiling Tips

Profiling Tips : Lock Contention

- Use of Java synchronization can lead to highly contended locks.
- High sys cpu utilization and high *smtx* (spin on mutex) counts on Solaris *mpstat* can be an indication highly contented locks.
- Sun Studio Collector / Analyzer is very good with identifying Java objects experiencing lock contention.

When To Use What Tool?

Profiling Tips : Good Use Cases

- Sun Studio Collector / Analyzer
  - CPU profiling entire application
  - Sys cpu profiling or distinct usr vs sys profiling
  - Lock contention profiling
  - Integration with scripts, command files or batch files
  - Also view performance of JVM internals including methods
  - Want to see machine level assembly instructions
  - Narrow to specific window of sampling

Profiling Tips : Good Use Cases

- NetBeans Profiler
  - Profiling subset of application, for CPU profiling or heap profiling
  - Heap profiling
  - Finding memory leaks
  - Profiling an application using NetBeans IDE and/or NetBeans project
  - Remote profiling
  - Attach to running application
  - View profiling as application is running

Profiling Tips : Good Use Cases

- DTrace and DTrace scripts
  - Non-intrusive snapshots of running application
  - Command line utility
  - Can leverage existing public scripts
    - Heap profiling
    - Finding memory leaks
    - Monitor contention
    - JIT Compilation
    - Garbage collection activity
    - Method entry / exit
    - Java Native Interface entry / exit

Profiling Tips : Good Use Cases

- jmap / jhat
  - Heap profiling
  - Finding memory leaks
  - Simple command line utilities
  - Quick & easy snapshots of running application

Profiling Tips : Inlining effect

- If observing mis-leading or confusing results in cpu profiles, disable inlining.
- It is possible methods of particular interest are being inlined and leading to misleading observations.
- To disable inlining, add the following JVM command line switch to the JVM command line args: -XX:-Inline
  - Note: disabling inlining may distort “actual” performance profile

Identifying Problem Patterns

How to reduce lock contention

- Approaches to reduce lock contention
  - Identify ways to partition the “guarded” data such that multiple locks can be integrated at a finer grained level as a result of partitioning.
  - Use a concurrent data structure found in Java 5's java.util.concurrent package.
  - Separate read lock from write lock by using a Java 5 ReentrantReadWriteLock if writes are much less frequent than reads.

Concurrent data structures

- A note on using concurrent data structures versus synchronized collections.
  - Concurrent data structures may introduce additional cpu utilization overhead and may in some cases not provide as good of performance as a synchronized Collection.
  - Compare the approaches with meaningful workloads.

Concurrent data structures, cont.

- Concurrent data structures versus synchronized collections, continued...
  - HotSpot JVM biased locking may also improve synchronized Collection performance. -XX: +UseBiasedLocking introduced in JDK 5.0_06
  - Improved in JDK 5.0_08
  - Must be explicitly enabled in JDK 5 versions.
  - Enabled by default in JDK 6 versions.

Anti-patterns in method profiles

- Observing *Map.containsKey(key)* in profile.
  - Look at stack traces for unnecessary call flows which look like

```
if (!map.containsKey(key))
    value = map.get(key);
```

  - value will be null if a key is not found via map.get(key)
  - Other use cases using Map methods such as put(key, value) or remove(key) may potentially be eliminated depending too.

Anti-patterns in method profiles

- Observing high sys cpu times.
  - Look for monitor contention
    - Monitor contention and high sys cpu time have a strong correlation.
    - Consider alternatives to minimize monitor contention.
  - Look for opportunities to minimize number of system calls.
    - Example: read as much data as is ready to be read using non-blocking SocketChannels.
  - Reduction in sys cpu time will likely lead to better application throughput and response time.

Performance Profiling (focused on Java SE)