Towards High Performance Processing in Modern Java-based Control Systems

Marek Misiowiec
Wojciech Buczak, Mark Buttner
CERN
ICalepcs 2011
Performance with soft real time

Distributed system - Monitoring & Alarms at CERN

- collect data from over 10’000 devices
- heterogenous environment

Performance in middle-tier

- process with soft real-time constraints
  - lose no data during calculations
  - deliver results within time frame
- build on standard JDK

- tens of consoles
- 100’000 calc/s
- 30’000 data/s
- 10MB/s net
- high-level middleware
- 10’000 monitored devices
Technical focus

- common view on data and devices
- immutability favors parallelism
- decomposition for concurrency
- multithreaded communication
- optimal structures and algorithms
- garbage collectors, 32 vs 64 bit, Java Virtual Machine settings
Memory Management

Garbage Collection (GC)

- introduces **non-deterministic** behaviour
- slows the application with potentially long **stop-the-world pauses**
- makes it hard to achieve real-time

Diagram:
- Stack
- App
- GC
- App threads
- Largest problem for performance
Real-time is not about speed

Translates into requirements for GC

- we expect a degree of determinism
- number of stop-the-world pauses limited for a period

hard real-time: fatal
soft real-time: undesirable
Solution

- steady progress in Garbage Collection techniques

- tuning JVM with over 50 properties
  - memory sizes, number of GC threads,…

JVM GC history

<table>
<thead>
<tr>
<th>90’s</th>
<th>Serial Collector</th>
</tr>
</thead>
<tbody>
<tr>
<td>00’s</td>
<td>Parallel-Compacting, Concurrent Mark-Sweep</td>
</tr>
<tr>
<td>now</td>
<td>GarbageFirst</td>
</tr>
</tbody>
</table>
Garbage Collection concepts

Heap
GC concepts

can work with different collectors

- Young much smaller than Old
- objects tend to live **shortly**
- new objects in Eden
Young much smaller than Old

- Young much smaller than Old
- objects tend to live shortly
- new objects in Eden moderate in Survivors

**GC concepts**

minor collection: stop-the-world

Young generation

Old generation
Young much smaller than Old

objects tend to live **shortly**

new objects in Eden
moderate in Survivors
old in Old

**minor collection: stop-the-world**
GC concepts

major collection: stop-the-world

1st marking live objects
2nd sweeping memory
GC concepts

1st marking live objects
2nd sweeping memory
defragmentation: compacting

major collection: stop-the-world
Key improvements to collections:

- **parallel** – multiple GC threads
- **concurrent** – GC along with application

1st **marking** live objects

2nd **sweeping** memory

defragmentation: **compact**
Concurrent Mark-Sweep (CMS)

- generational, incremental, parallel
- partially concurrent: marking & sweeping in stages
- no compacting

Well tuned, most effective in our tests
GarbageFirst (G1)

Meets soft real time goal with high probability

- default in JDK7, succeeds Concurrent Mark-Sweep

- targeted for multi-processors with large memories
  - heavy use of multithreading
  - heap with many equal regions, no generations
  - compaction

- algorithmically complex

enabling in Java 6:
-XX:+UnlockExperimentalVMOptions
-XX:+UseG1GC
Outcome

Performance analysis with Java Standard Edition 6

- fine-tuned CMS most effective, G1 close second

Observations

- 64 bit architecture
  - 4GB limit per JVM crossed
  - too much memory used - performance penalty
- repetitive nature of processing diminishes effects of dynamic class loading
- long startup time is negligible
- short lived objects, locality
Conclusions

- High Performance Computing with *soft real time* requirements can be achieved with modern JVMs

- JVM tuning is indispensable
  - select most fitting garbage collector
  - set JVM options
  - approach 64 bit boost with restraint

- constant improvement in memory management
  - G1 (Java 7) more efficient than CMS (Java 6)