Enabling Java-based VoIP backend platforms through JVM performance tuning

(Bruno Van Den Bossche, Filip De Turck, April 3rd 2006)
Outline

✓ Introduction
✓ Java 4 Telecom
✓ Evaluation Setup
  ➢ Hardware
  ➢ Software
✓ Java Virtual Machine
  ➢ Default behavior
  ➢ Possible Optimizations
  ➢ Optimized Behavior
✓ Results
✓ Conclusions
Introduction

✓ Software backend platforms are increasingly popular in VoIP offerings.
✓ Java is currently one of the most popular programming languages for implementing business logic.
✓ Is Java suitable for implementing VoIP and telecom related applications?
✓ Java Application Servers for Telecom related applications are emerging:
  ➢ SIP Servlet
  ➢ JAIN SLEE
Telecom Applications: VoIP

✓ Telecom related applications have very specific requirements
  ➢ Low Latency
  ➢ High Throughput

✓ Example: Softswitch
  ➢ How fast can it set up a call?
  ➢ How many calls can in set up per second?

✓ Java 4 Telecom?
Java Virtual Machine: Features

- Java Byte Code is executed by a virtual Machine
  - This makes it platform independent

- Java features automatic Memory Management and a Garbage Collector
  - This simplifies the task of the developer
Java Virtual Machine: Problems

✓ Java Byte Code is executed by a virtual Machine
  ➢ This can be a performance penalty

✓ Java features automatic Memory Management and a Garbage Collector
  ➢ This can cause a performance penalty due to unpredictable behavior of the system.
Evaluation: Proxy 200 benchmark

Alice Proxy Bob

Invite
Trying [100]
Ringing [180]
OK [200]
ACK
BYE
Invite
Ringing [180]
OK [200]
ACK
BYE

Media transfer

Measured Time Interval

No Media Session started when benchmarking
Evaluation Setup: Hardware

Alice

Bob

Internet

Perform routing

Proxy

Dual Opteron
1.6Ghz 2GB RAM
Linux kernel 2.6
Evaluation Setup: Software

- SIP Servlet
  - BEA Weblogic SIP Server
- JAIN SLEE
  - Open Cloud Rhino
- Benchmark software
  - SIPp (test tool / traffic generator for the SIP protocol)
SIP Servlet: Architecture

- Container based
  - Life cycle management
  - Manage network listen points
- Very similar to HTTP-Servlet
- SIP Protocol Specific
- Request Response Model

Diagram:
- Application client
- SIP Servlet Server
- JAIN SIP Servlet API
- Container
- SIP Stack
JAIN SLEE: Architecture

- Container based
  - Life cycle management
- Protocol Agnostic
- Resource Adaptors allow protocols to be "plugged in"
  - Manage network Listen Points
- Event driven and asynchronous

Application client

JAIN SLEE SBB API

JAIN SLEE SBB

Activity ctx

Application

JAIN SLEE Container

JAIN SLEE Server

JAIN SIP

SIP RA

JAIN JCC

JCC RA

JAIN JCC RA

JAIN SIP RA
# Message Handling

<table>
<thead>
<tr>
<th>SIP Request</th>
<th>SIP SERVLET</th>
<th>JAIN SLEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVITE</td>
<td>doInvite()</td>
<td>onInviteEvent()</td>
</tr>
<tr>
<td>ACK</td>
<td>doAck()</td>
<td>onAckEvent()</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>doOptions()</td>
<td>onOptionsEvent()</td>
</tr>
<tr>
<td>BYE</td>
<td>doBye()</td>
<td>onByeEvent()</td>
</tr>
<tr>
<td>CANCEL</td>
<td>doCancel()</td>
<td>onCancelEvent()</td>
</tr>
</tbody>
</table>

Methods defined in specification:
```
javax.servlet.sip.SipServlet
```

Methods defined by event names specified by the Resource Adapter:
Results: SIP Servlet (default)

Response time as a function of call rate and system load

- Average response time
- 50th percentile
- 95th percentile

System load

Call Rate (caps)

Response time (ms)

System Load (%)

50 ms

25 ms

Results: SIP Servlet (default)

25 ms

50 ms
Results: JAIN SLEE (untuned)

Response time as a function of call rate and system load

- Average response time
- 50th percentile
- 95th percentile

System load

Call Rate (caps)

Response time (ms)

- 50 ms
- 25 ms

0 50 100 150 200
0 20 40 60 80 100
0 10 20 30 40 50 60 70 80 90 100

3 April, 2006
Cause of the problems?

✓ Memory Management and Garbage Collector

Legend
- Heap Usage
- GC-Time
- GC-occurrence
- Tenured Gen.
- Young Gen.

Image created with gcviewer (http://www.tagtraum.com/gcviewer.html)
Solution: Virtual Machine Tuning

✓ Memory Organization

✓ Garbage Collection
Memory Tuning

- Young generation
  - Eden
  - Survivor
  - Virtual
  - Tenured

- Perm generation
  - Perm
  - Virtual

Tiny Survivor Spaces

Total Heap Size large enough

Fixed Young Generation Size

Other Generation Sizes NOT Fixed

Other Generation Sizes NOT Fixed
Garbage Collector Tuning

**Default**
- Long GC pauses
  - All generations

**Parallel**
- Multi-threaded GC
  - Only for young generation

**Concurrent**
- GC partially parallel with Application Execution
  - Only for tenured generation

**Parallel & Concurrent**
- All generations

**Performance implications:** More resources are spent on Garbage Collection
Tuning Options

- `-Xmx512m`
- `-XX:MaxNewSize=32m`
- `-XX:NewSize=32m`

- `-XX:+UseParNewGC`
- `-XX:+UseConcMarkSweepGC`
- `-XX:+CMSIncrementalMode`
- `-XX:+CMSIncrementalPacing`
- `-XX:CMSIncrementalDutyCycleMin=0`
- `-XX:CMSIncrementalDutyCycle=10`

- `-XX:+UseTLAB`
- `-XX:MaxTenuringThreshold=0`
- `-XX:SurvivorRatio=128`

Sizing of the memory generations
Selection of the Garbage Collector
Allow multi-threaded memory allocation
Move long living objects to Tenured Generation
Problems solved?

Legend
- **Heap Usage**
- **GC-Time**
- **GC-occurrence**
- **Tenured Gen.**
- **Young Gen.**

No long lasting GC
No heap resizing
Low response time

Image created with gcviewer (http://www.tagtraum.com/gcviewer.html)
Results: SIP Servlet

Response time as a function of call rate and system load

- Average response time
- 50th percentile
- 95th percentile

- Higher CPU load

- Improved Latencies

Call Rate (caps) vs. Response time (ms)

- 50 ms
- 25 ms
Results: JAIN SLEE

Response time as a function of call rate and system load

- Average response time
- 50th percentile
- 95th percentile

System load

Call Rate (caps)

Response time (ms)

System Load (%)
Conclusions

- Java Technologies can simplify the Design and Management of Telecom and VoIP related applications

- Java Virtual Machine Tuning can improve the Java Garbage Collection significantly

- Java Application Servers combined with appropriate tuning can meet strict Low Latency Requirements