Learning Objectives

As a result of this session, you will:

- Know about the architecture of SAP AS Java for scalability
- Understand which KPIs are important for measuring and analyzing performance
- Be able to perform single-user tests
- Know the SAP Java Memory Analyzer
- Be able to measure and analyze the performance of database accesses
- Understand the benefits and success factors of multi-user load tests
- Know how to perform scalability tests and interpret the test results
- Know how to analyze the Java thread contention problem
- Be able to do CPU sizing and prediction
Performance & Scalability of SAP AS Java

KPIs for Measuring & Analyzing Performance

Single-User Performance Tests

Multi-User Load Tests
Performance & Scalability of SAP AS Java

KPIs for Measuring & Analyzing Performance

Single-User Performance Tests

Multi-User Load Tests
Dimensions of Scalability

- Hardware adaptivity
  - Local/Wide Area Network
  - Concurrent users
  - Number of servers
  - Number of processed objects
  - Size of data
  - Number of objects in the DB
  - Number of Clients
  - Data volume

Concurrency
Architecture of SAP AS Java: Scaling in CPU and Memory

JVMs:
- D - dispatcher node
- S - server node

DB: database
E: Enqueue server
M: Message server

Central Instance

Dialog Instance

Load balancer
Architecture of SAP AS Java: Scaling in CPU and Memory

**External scalability**
Scaling over multiple machines
Any load balancer can be used to distribute the load between Java Instances

**Internal scalability**
Scaling with adding CPU and Memory to one machine

**CPU scalability**
- Multi-threading of each server node (Java VM) utilizes the CPU resource

**Memory scalability**
- Memory utilization is controlled by the heap configuration of the Java VM
- Use more server nodes to utilize more physical memory on a server machine
- For detailed Java VM Heap configuration, please refer SAP Note 723909
Performance & Scalability of SAP AS Java

KPIs for Measuring & Analyzing Performance

Single-User Performance Tests

Multi-User Load Tests
KPIs for Java Performance Measurement and Analysis

**Good performance KPIs**
- Reflect the real performance requirements
- Are accurate and reproducible
- Give indications to possible optimizations

**User KPI**
Response time of a User Interaction Step (UIS)
- The elapsed clock time from user input till the next screen is displayed

**System KPI**
Throughput
- Number of transactions per time unit

CPU consumption
- High CPU resource requirement mean high TCO

Memory consumption
- High memory resource requirements impact TCO
- Appropriate heap configuration reduces the Garbage collection activities and maximizes the throughput
KPIs for CPU Consumption

**What can be measured?**
CPU utilization of a physical server (in multi-user tests)
CPU time of a user interaction step (or total CPU time of a business scenario)

**How to measure the CPU utilization?**
Use operating system utilities
Use SAP Management Console (MC)

**How to measure the CPU time of a user interaction step/scenario?**
Use Java Distributed Statistical Records (JDSR) in NW Web Administrator
Memory Utilization vs. Memory Consumption

**Memory utilization is the allocation of physical memory by a Java VM**

Is controlled by the java heap configuration
Avoid paging for Java applications (the max. heap size + permanent space + stack must completely fit into the available physical memory)

**Java memory KPIs**

- Framework Space
- User Session Space
- Processing Space
Java Memory KPIs and Measurement Procedures

Framework space (MB)
Memory footprint of a JVM after warm up (e.g. deployed, started and used Java applications)

User session space (MB)
Memory allocated by active or inactive user, which is not shared by other users
Average memory per user = \((M - \text{Framework space}) / N\)

Processing space (MB per user interaction step)
Total garbage collected bytes of K user interaction steps / K
Impact of Java Garbage Collection on Performance

The automatic Java garbage collection
Productivity enhancement of Java software development
Java memory usage is documented in GC log
Several collectors with different algorithms (for young/old generation, concurrent/parallel, ...) are available

The impact of frequent/long GC on performance
Increased response time
Low CPU utilization
No locality of memory accesses, paging becomes "dangerous"
Java “resonance” effect (in worst cases)
GC KPIs: GC Duration and GC Interval

**GC duration**
The average elapsed time for completion of a GC cycle. Information is available in the GC log.

**GC interval**
The average time between two successive GC occurrences. Information is available in GC log.

**The GC KPIs**
Relative GC time = GC duration / GC interval

Recommendation regarding minor GCs
- GC duration < 0.2 s and GC interval > 1.0 s

Recommendation regarding full GCs
- GC duration < 10 s, GC interval > several minutes
Example of Memory KPIs

A certified EP-ESS Benchmark with SAP NetWeaver Portal 7.0, SP06, on a 2 processors / 4 cores / 8 threads machine (AIX on Power, ca. 10.000 SAPS)

<table>
<thead>
<tr>
<th>KPI</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of concurrent users with 10 seconds think time</td>
<td>2600</td>
</tr>
<tr>
<td>CPU utilization</td>
<td>99%</td>
</tr>
<tr>
<td>Average response time</td>
<td>1.811 s</td>
</tr>
<tr>
<td>CPU time per user interaction step</td>
<td>0.018 s</td>
</tr>
<tr>
<td>Framework space</td>
<td>325 MB</td>
</tr>
<tr>
<td>User session space</td>
<td>&lt; 1 MB</td>
</tr>
<tr>
<td>Processing space</td>
<td>8.714 MB</td>
</tr>
<tr>
<td>GC interval</td>
<td>2.390 s</td>
</tr>
<tr>
<td>GC duration</td>
<td>0.424 s</td>
</tr>
</tbody>
</table>
Performance & Scalability of SAP AS Java

KPIs for Measuring & Analyzing Performance

Single-User Performance Tests

Multi-User Load Tests
Single-User Performance Testing

**Benefits and advantages of single-user tests**

JDSR of Java Web AS provides easy way to measure important business processes for single user

Providing baseline performance figures, early focus on response time

Tests can be easily repeated for accurate, reproducible results

Sharing the test system with users for other implementation and testing activities

**Be aware of caching effects and Java JIT compiler behavior**

Perform warm-up test runs before measurements
Performance Measurements with JDSR (1/2)

Java Distributed Statistical Records (JDSR)

Performance Measurements with JDSR (2/2)

**Performance KPIs measured with JDSR**
Response time (server side)
CPU time
DB time
External calls time
Database calls (read, insert, update, delete)
Bytes send/received to database
Processing memory space
RFC calls to the backend system (function modules details)
Analysis of Database Accesses (1/2)

Database access analysis with Open SQL Trace

http://<host>:<port>/nwa -> Problem Management → Database → Open SQL Monitors
Analysis of Database Accesses (2/2)

**KPIs per user interaction step**
Numbers of executed SQL statements (select, insert, modify, delete)
Transferred data volume

**Performance Guidelines**
No identical selects within a user interaction step
Major DB accesses should be supported by appropriate indexes
Consider caching of DB data to be reused and shared in different scopes of context (user interaction step, user session, user, JVM)

…
Memory Consumption Analysis

**Memory analysis**
User session space analysis
Framework space analysis

**The Eclipse Memory Analyzer**
Visit our SDN Wiki page and blogs
- The tool is available for download

Supported platforms (HPROF)
- SUN, SAP, and HP JVM
- From version 1.4.2_12, 5.0_7, and 6.0
Features of Eclipse Memory Analyzer (1/3)

The information in the heap dump is difficult to analyze

- Huge object graphs
- Long reference chains
- Reference cycles
- No object size accumulation

Interprets and displays object dependencies, and adds two very important pieces of information for memory analysis

- Retained set/size of a single object (or a set of objects)
- A Dominator Tree
Features of Eclipse Memory Analyzer (2/3)

The Retained Set
The *retained set* of a single object $x$ (or a set of objects $x_1, x_2, \ldots, x_n$) is the set of objects which will be garbage collected if $x$ (respectively the set $x_1, x_2, \ldots, x_n$) is garbage collected.

The Retained Size
The retained size of an object $x$ (or a set of objects $x_1, x_2, \ldots, x_n$) is the sum of the net heap sizes of all objects in the retained set of $x$ (respectively of $x_1, x_2, \ldots, x_n$).

<table>
<thead>
<tr>
<th>Set of elements</th>
<th>Retained Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>$C, F, G, J$</td>
</tr>
<tr>
<td>$K$</td>
<td>$K$</td>
</tr>
</tbody>
</table>
Features of Eclipse Memory Analyzer (3/3)

**Dominators and Dominator Tree**

Definition: An object \( x \) *dominates* an object \( y \) if every path in the object graph from the start (or the root) node to \( y \) must go through \( x \).

Definition: The *immediate dominator* \( x \) of some object \( y \) is the dominator closest to the object \( y \).

We build a *dominator tree* out of the object graph. In the dominator tree each object is the immediate dominator of it’s children. The dominator tree allows us to easily identify the dependencies among the objects.
Performance & Scalability of SAP AS Java

KPIs for Measuring & Analyzing Performance

Single-User Performance Tests

Multi-User Load Tests
Multi-User Load Testing

**Multi-user load testing**
Generate system load by simulation of concurrent users
Execute defined critical business scenarios
Monitor and analyze system behavior under load

**Benefits**
Ensure that functional and performance requirements are fulfilled under system load
Analyze and optimize system/software components and configurations if requirements are not met at first

**Costs**
Exclusive allocation of system landscape
Availability and stability of business applications
Expertise of project team in all areas (business process, development, IT operations, performance experts, and so on)
Success Factors of a Load Test Project (1/2)

Define clear goals of the load test project
Which business scenarios are mission critical and performance relevant?
- Derived from the business process
What system load to be simulated?
- Number of concurrent users and think time
- Volume of business data
Hardware sizing of the test system
- Execute single-user tests before load tests
- Predict load test result based on single-user test result (sizing)
Time line of the load test project
- Check readiness to start load tests (availability, stability, …)
- Plan time for iterative analysis and optimization

Ensure the correct simulation at technical level
The system configuration reflects the production environment
- Application servers, DB server, …
- Network connections, load balancing, …
Simulated business scenarios can be executed concurrently and repetitively
Ensure the correct simulation at technical level

Correct simulation of user behavior

- Execute business scenarios in loops and ensure a dominating „high-load phase“ where all simulated users are running in parallel (concurrent users)
- Choose between simulation or same user of different (new) user in each loop
- Make sure the parameterization (input parameters values chosen from valid ranges) and the correlation (substitution of dynamic session state relevant parameters) is correct
- Verify system responses in load test scripts
- Simulate correct browser caching behavior
Scalability Tests (1/2)

How to conduct scalability tests

Carry out a series of load tests with an increasing number of concurrent users (and constant think time)

Monitor and evaluate

- Average response time
- Average CPU time per user interaction step
- CPU utilization (on machine total and java, db, etc. processes)
- Memory utilization (overall on machine and in java heap)
- Network I/O (throughput, latency)
Scalability Tests (2/2)

The prove of CPU scalability

![Graph showing CPU scalability](image)

- **CPU utilization** increases linearly with the number of users.
- **CPU time per user interaction step** remains constant.
- **Average response time** and **CPU time per user interaction step** remain constant as expected from queueing models.

Response time behavior as expected from queueing models.
CPU Sizing and Prediction

Result of single-user tests
- CPU\textsubscript{UIS}: Average CPU time per user interaction step (UIS)

System load profile
- ThT: Average think time – time between two successive user interaction steps
- N: Number of concurrent users

System behavior under load
- RespT: Average response time per user interaction step
- Util\%: Target Percentage of CPU utilization (usually 65%)

System size (considering CPU resource only)
- #CPU: Number of CPUs in the system

\textbf{Sizing is valid only for Scalable Software}

\[ #\text{CPU} = N \times CPU_{UIS} \div ((\text{RespT} + \text{ThT}) \times \text{Util}\%) \]

![Diagram showing user interaction with response time and think time](image)
Analyze the Java Thread Contention Problem

**Symptoms**

Significantly lower CPU time per user interaction as compared to the high response time for same user interaction (high wait time)

The high response time is reproducible with increase of load even with less than 65% CPU usage

Timeout of requests (failures)

**Example**

Application blocked by two deadlocked threads

HTTP requests return after certain timeouts

**Identify the contention by Stack Trace Dump (or any thread monitoring tool)**

Trigger stack trace dump(s) during the contention (SAP Note 710154)

Find the waiting / deadlocked threads

Identify the components that cause the contention
Analyze GC Duration and GC Interval

GC KPIs can be calculated as average counters from a GC log
Frequent or long GCs can be the reason for scalability issues
apply JVM heap configuration tuning
implement software memory optimizations

4470.800: [GC 4470.800: [ParNew: 153248K->17024K(153344K), 0.1094240 secs] 29013220K->28898090K(33537408K), 0.1098090 secs] [Times: user=0.64 sys=0.02, real=0.10 secs]
4471.113: [GC 4471.113: [ParNew: 153344K->17024K(153344K), 0.1050420 secs] 29034410K->28917798K(33537408K), 0.1054180 secs] [Times: user=0.66 sys=0.00, real=0.11 secs]
4471.573: [GC 4471.573: [ParNew: 153344K->17024K(153344K), 0.1309050 secs] 29054118K->28957299K(33537408K), 0.1312640 secs] [Times: user=0.83 sys=0.01, real=0.13 secs]
4472.001: [CMS-concurrent-preclean: 4.147/6.230 secs] [Times: user=27.36 sys=0.55, real=6.23 secs]
4472.001: [CMS-concurrent-abortable-preclean-start]
4472.036: [GC 4472.036: [ParNew: 153344K->17024K(153344K), 0.1202210 secs] 29093619K->28991097K(33537408K), 0.1205910 secs] [Times: user=0.78 sys=0.00, real=0.12 secs]
4472.327: [GC 4472.327: [ParNew: 153344K->17024K(153344K), 0.1160550 secs] 29127417K->29013738K(33537408K), 0.1164640 secs] [Times: user=0.64 sys=0.00, real=0.12 secs]
4472.527: [CMS-concurrent-abortable-preclean: 0.279/0.526 secs] [Times: user=2.59 sys=0.07, real=0.52 secs]
4472.531: [GC[YG occupancy: 71420 K (153344 K)]4472.531: [Rescan (parallel), 0.0575640 secs]4472.589: [weak refs processing, 1.1027950 secs] [1 CMS-remark: 28996714K(33384064K)] 29068135K(33537408K), 1.1972350 secs] [Times: user=1.51 sys=0.01, real=1.19 secs]
4473.729: [CMS-concurrent-sweep-start]
4473.802: [GC 4473.802: [ParNew: 153344K->17024K(153344K), 0.1219790 secs] 29149892K->29032141K(33537408K), 0.1223570 secs] [Times: user=0.76 sys=0.01, real=0.12 secs]
4474.027: [GC 4474.027: [ParNew: 153344K->17024K(153344K), 0.0858120 secs] 29159111K->29032906K(33537408K), 0.0862140 secs] [Times: user=0.54 sys=0.01, real=0.09 secs]
Summary

Take advantage of the scalable architecture of SAP J2EE Engine

Define well motivated performance KPIs

Single-user tests
Measure performance KPIs with JDSR
Perform advanced memory analysis with Eclipse Memory Analyzer
Analyze database accesses with Open SQL Trace

Multi-user load tests
Select valid business scenario
Define realistic load profile
Align results expectation to HW sizing prediction
Ensure correct simulation at technical level
Resolve thread contentions and memory leaks
Thank you