J2EE Instrumentation for software aging root cause application component determination with AspectJ

Javier Alonso
Josep Ll. Berral
Ricard Gavaldà
Jordi Torres

Technical University of Catalonia, Spain

Contents

- Motivation
- Our Contribution
  - Preliminary concepts
  - Architecture
  - Root cause determination strategy
- Experimental Case Study
- Conclusion & Future work

Motivation

- New challenges are demanded by the society.
  - Availability of the information
    - At any time
    - From everywhere

- Becoming in growing complexity day by day.

Motivation

- The growing complexity causes:
  - Necessity for brilliant IT professionals.
  - Increment of the Total owner Cost of the IT infrastructures.
  - Increment of the faults/outages due to (directly or indirectly) the complexity.

Motivation

- These faults/outages have an important impact of the revenue of the companies:
  - Around US$125,000 per hour, direct impact
  - A part of the indirect impact
- Several studies show that the current system outages are more often due to software faults.

Motivation: Software Aging

- One of the most important reasons for software failures is the software aging phenomenon.

- The software aging
  - Accumulation of errors, usually provoking resource contention during long running application
  - Gradual performance degradation could also accompany software aging phenomena.

Motivation: Software Aging

- Software aging related with:
  - Memory bloating/leaks
  - Unterminated threads
  - Data corruption
  - Unreleased file-locks
  - Overruns
  - Potentially some of them together

Motivation: Software Aging

- The applications have to deal with software aging in production stage.
  - The unaffordable and hardly cost task to avoid all software bugs.

- What is it the solution?
  - Software rejuvenation

Motivation: Software Rejuvenation

- Software rejuvenation
  - Basically, reboot the system, although there are most sophisticated techniques like micro-rebooting.
  - There are two main strategies:
    - Time based strategy.
    - Proactive based strategy.

Motivation: Software Rejuvenation

- Time based strategies:
  - Rejuvenation is applied regularly and periodically.
  - Well-known used in web servers.

- Proactive based strategies:
  - System metrics are monitored continuously
  - The rejuvenation action is triggered when the system is near to the crash according to the system metrics.

Motivation: Software Rejuvenation

- The proactive approach is better because:
  - We can reduce the rejuvenation actions

- The effectiveness of the proactive approach depends on the accuracy of the monitoring metrics.

Motivation: Root cause rejuvenation

- However, traditional monitoring tools understand the applications as "*black boxes*".

- This fact makes impossible to know what the *root cause* of the software aging is.
  - We understand as "*root cause*" the system component/s causing of the software aging.

Motivation: Root cause rejuvenation

- Monitoring tools do not offer enough clues about the root cause of failure.
  - The most used rejuvenation mechanisms are based on rebooting or application restarting.
- Rebooting implies also a reduction of availability
  - New more accurate techniques are proposed to reduce the Mean Time to Repair (MTTR), increasing the Availability.

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

Motivation: Root cause rejuvenation

- Micro Rebooting
  - Apply the recovery technique only over the component of the application that causes the failure.
  - However, this technique needs a monitoring tool or detection mechanism that allow us to determinate the root cause of the failure.

Our Contribution

- We present a monitoring framework to help to determine the *"root cause"* of the software aging phenomena.

- Using technologies:
  - Aspect Oriented Programming (AOP)
  - Java Management Extensions (JMX)

- For J2EE infrastructures.

Our Contribution

- The idea:
  - Monitoring the resources consumed by every software component of a J2EE application
  - Monitoring the trend of the consumption
  - Allowing to build a resource-component consumption map.
- All of all:
  - Without modify the source code.
  - With low overhead.

Our Contribution: Preliminary concepts

- Aspect Oriented Programming:
  - Allows to isolate the main business logic of the application from secondary functions like logs or authentication.
  - The core of AOP: *Aspects.*
    - *Aspects are composed by: Advices and Join Points.*
  - AOP allows to inject code in compile, load or runtime without to know the source code.

- We are injecting our observers using AOP

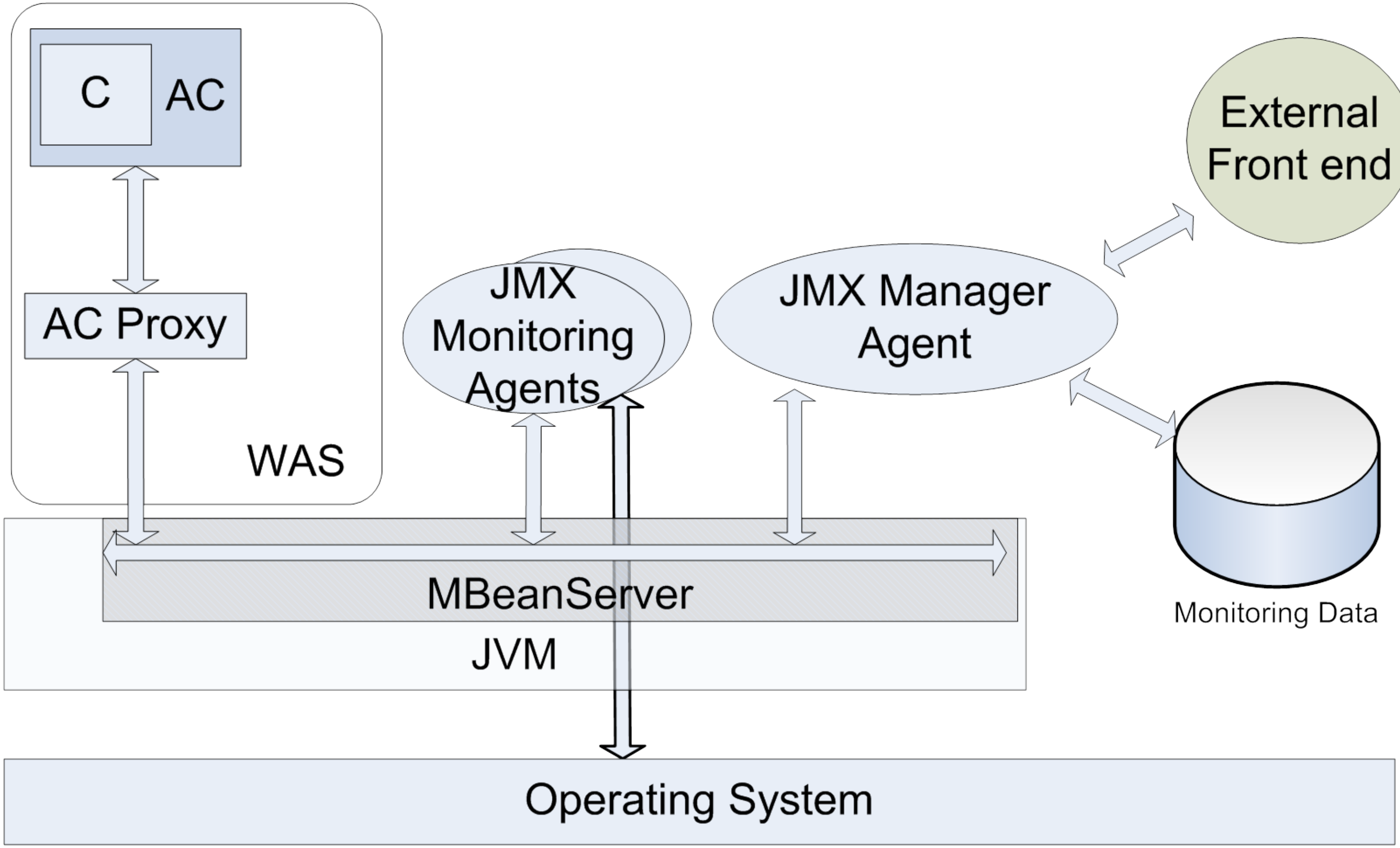
Our Contribution: Preliminary concepts

- Java Management Extensions:
  - a set of capabilities to manage and monitor any system component:
    - from devices to Java objects
  - is based on a 3-level architecture:
    - Probe level, Agent level and Remote Management Level.

Our Contribution: Architecture

- Aspect Component (AC)
  - Associated to every application component.
  - Manage the measurements of resource consumed and the trend.
- Aspect Component Proxy (AC-Proxy).
  - creates a communication channel between the AC and the JMX Manager Agent
- JMX Monitoring Agents
  - Access to the OS and collect system metrics for every component.
- JMX Manager Agent
  - has the responsibility to collect the metrics of each component and build the resource-component map.
  - Activate and deactivate ACs on demand.
- External Front-end
  - allow administrators to communicate with the JMX Manager Agent in real time or activate new ACs or new JMX Monitor Agents.

Our Contribution: Architecture

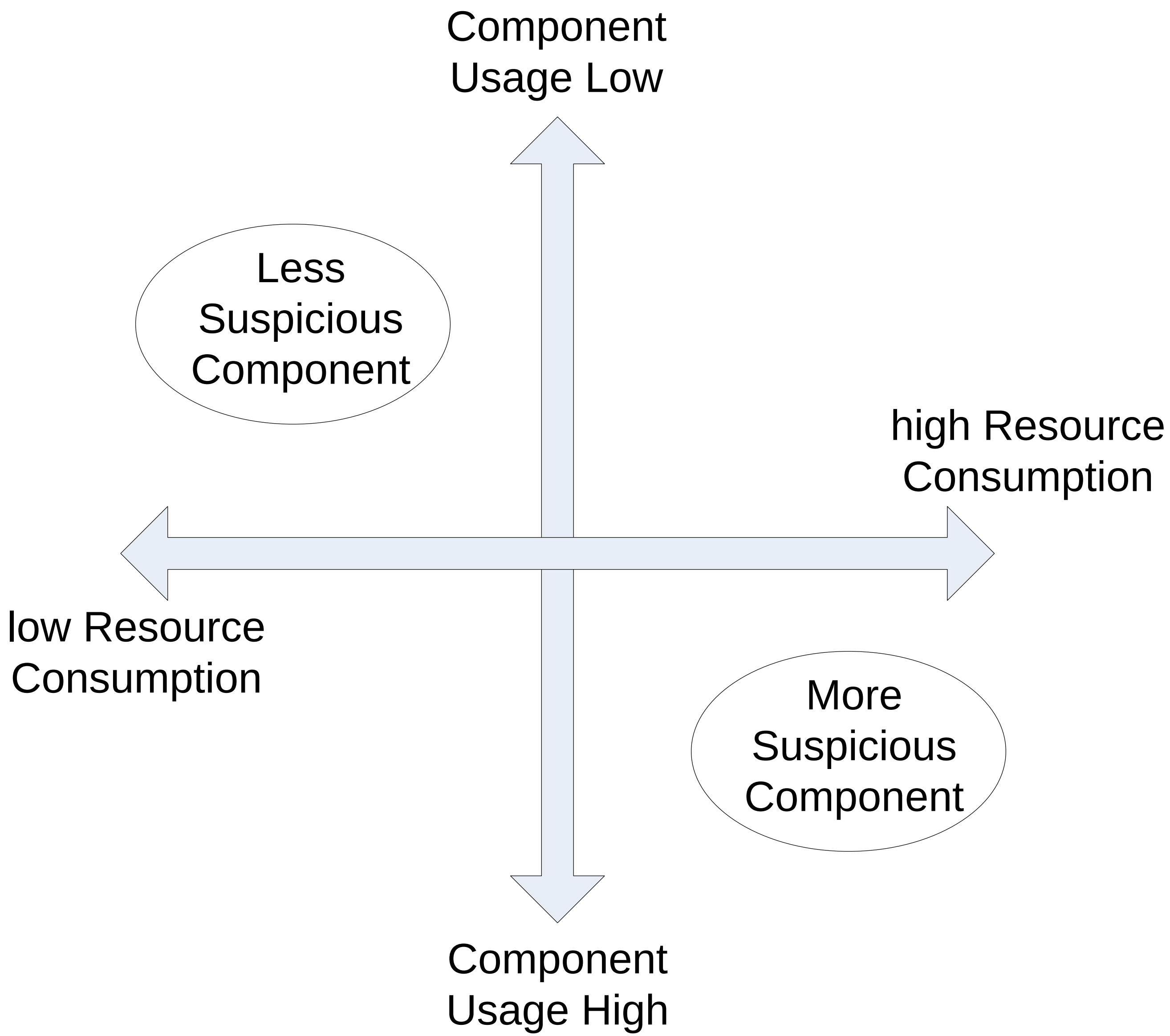


Our Contribution: Root cause determination strategy

- The JMX Manager Agent has a responsibility to build resource-component map:
  - The map is based on two axis:
    - Component usage
    - Resource consumption

- The map helps the engineers to priorize component “repair”

Our Contribution: Root cause determination strategy

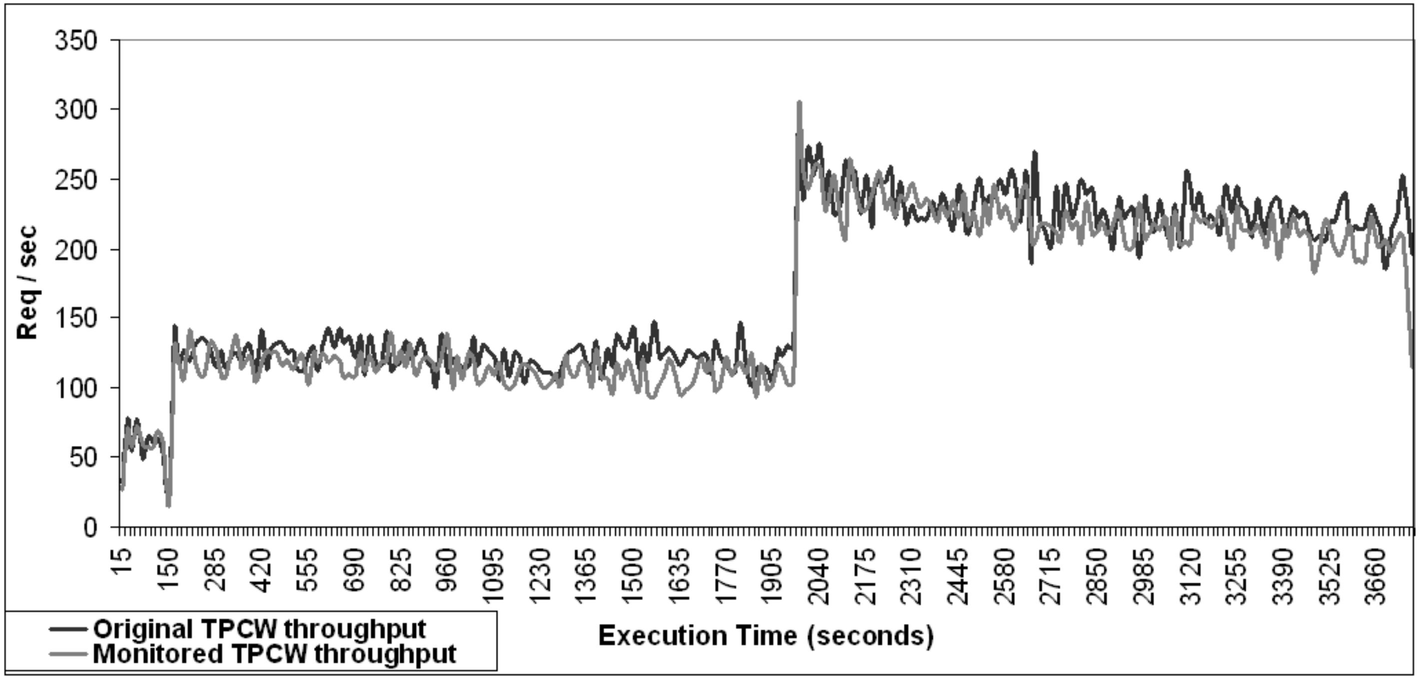


Experimental Case Study

- We have used TPC-W J2EE application to evaluate of our approach.
- TPC-W simulates a on-line book store and uses Emulated Browsers (EBs) to simulate clients.
- The EBs calculate a thinking time to simulate the time used by a human to decide what will be his next step in the web.
- We have modified a set of TPC-W servlets to inject memory leaks at different ratios.

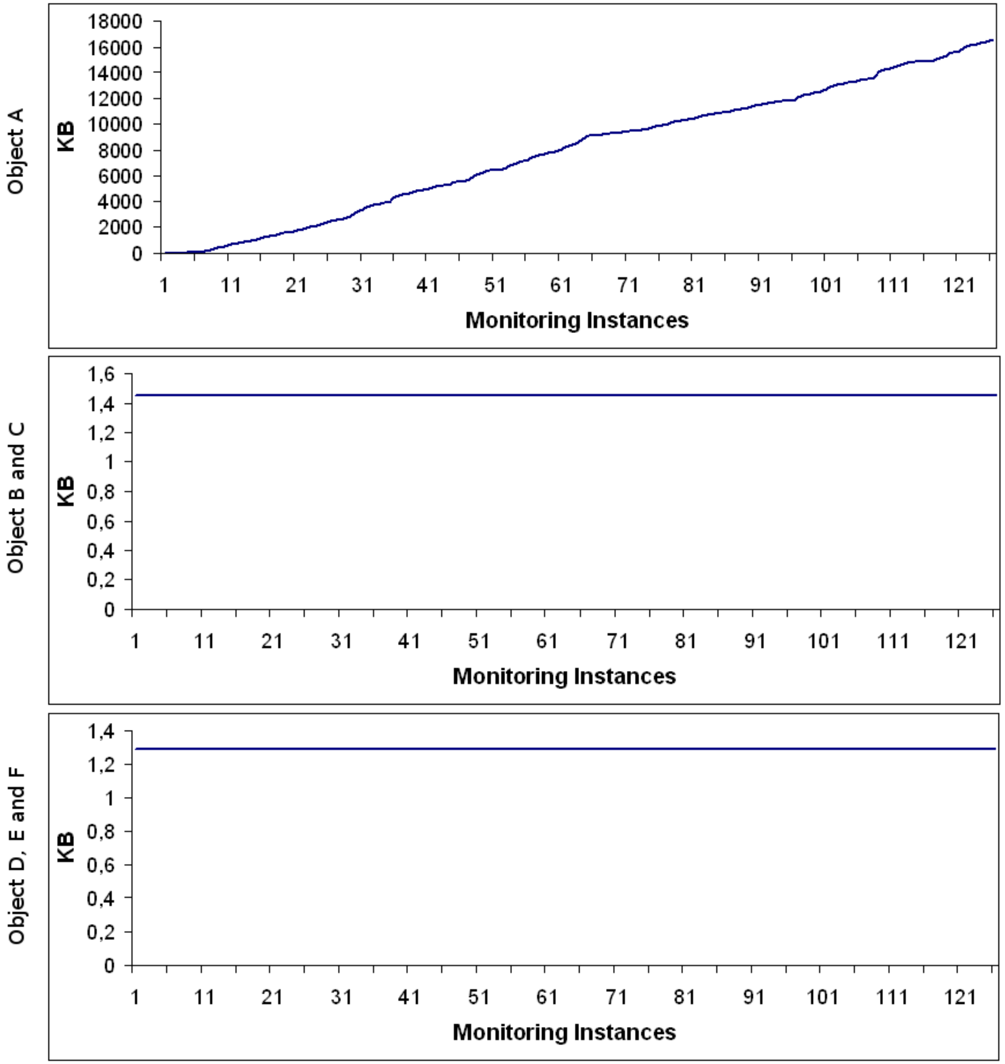
Experimental Case Study

- Overhead measurement:
  - Around 5% of overhead.



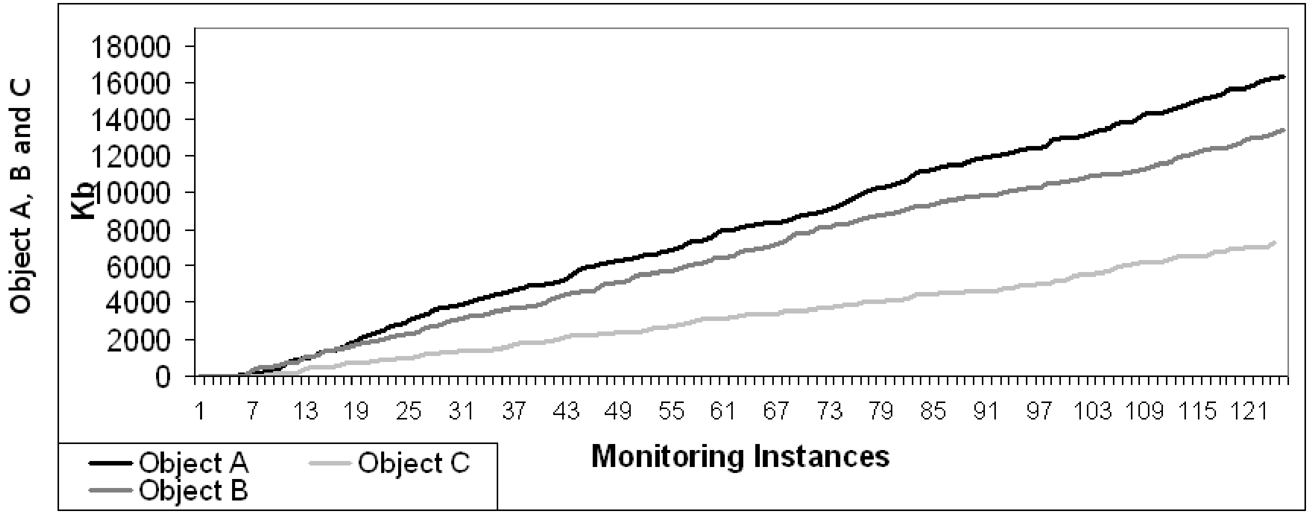
Experimental Case Study

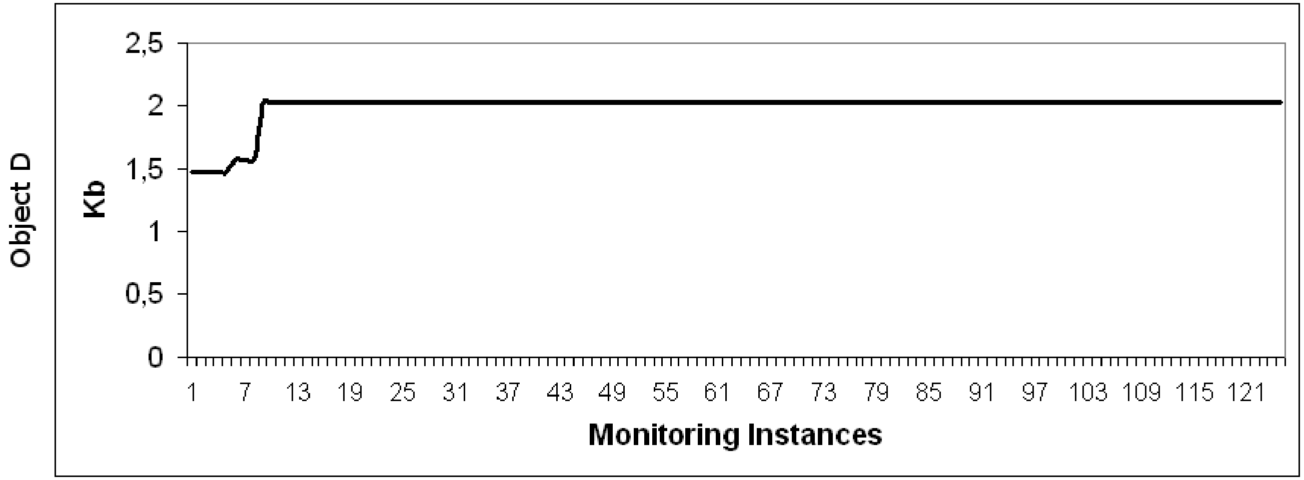
- Effectiveness to determine a memory leak:
  - Only one component injects a memory leak (100Kb every injection):



Experimental Case Study

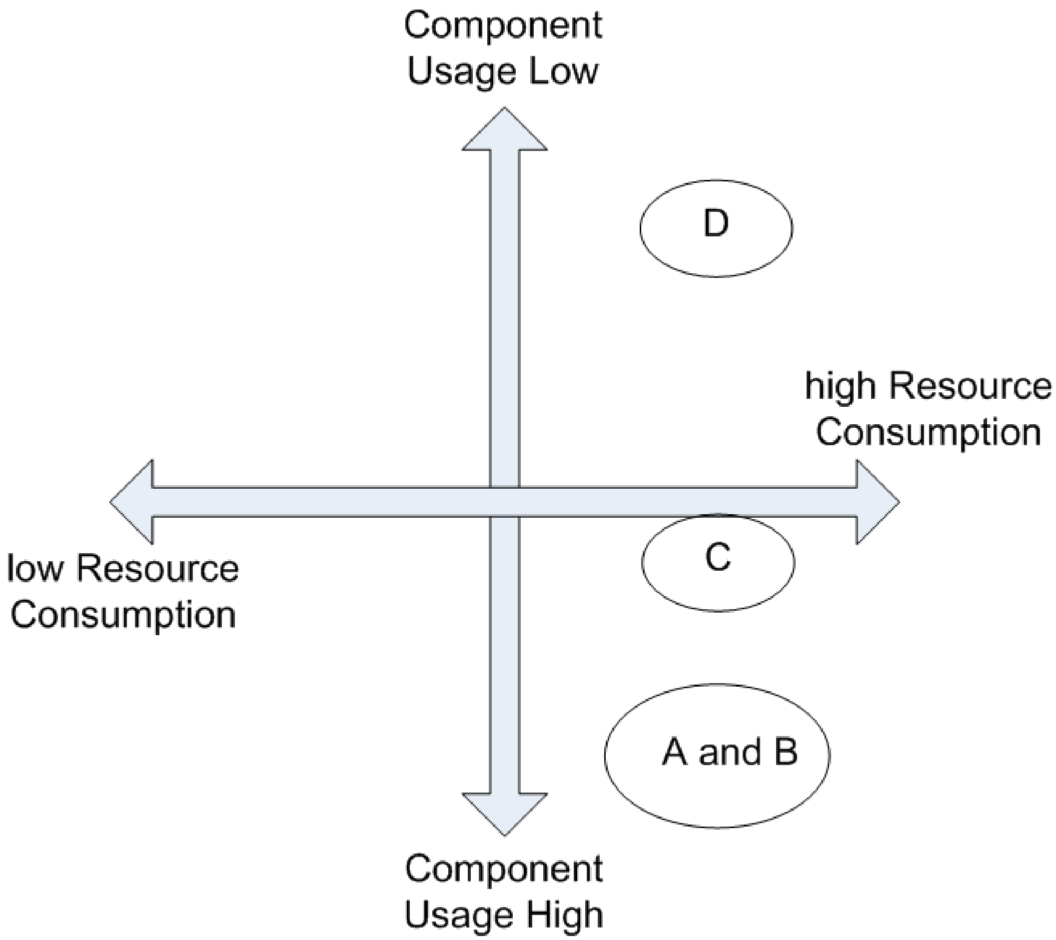
- Effectiveness to determine a memory leak:
  - Four components inject a memory leak (100Kb every injection):





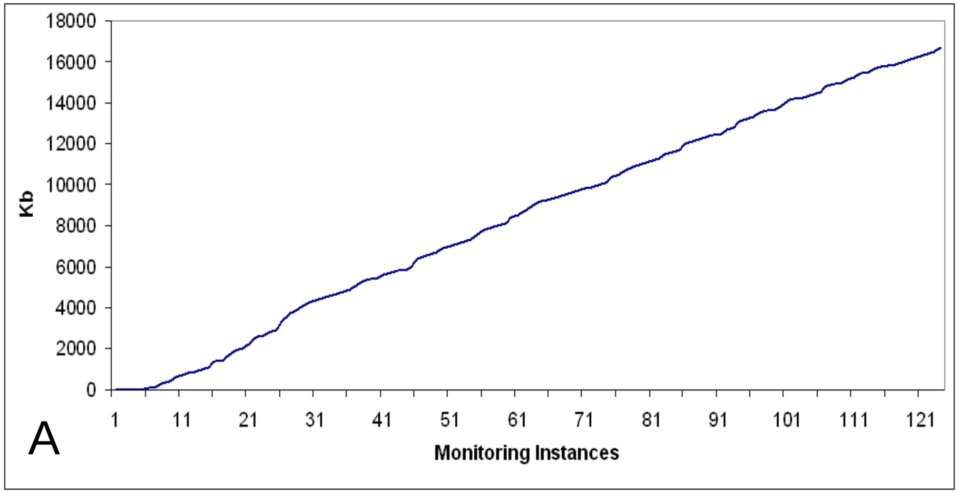
Experimental Case Study

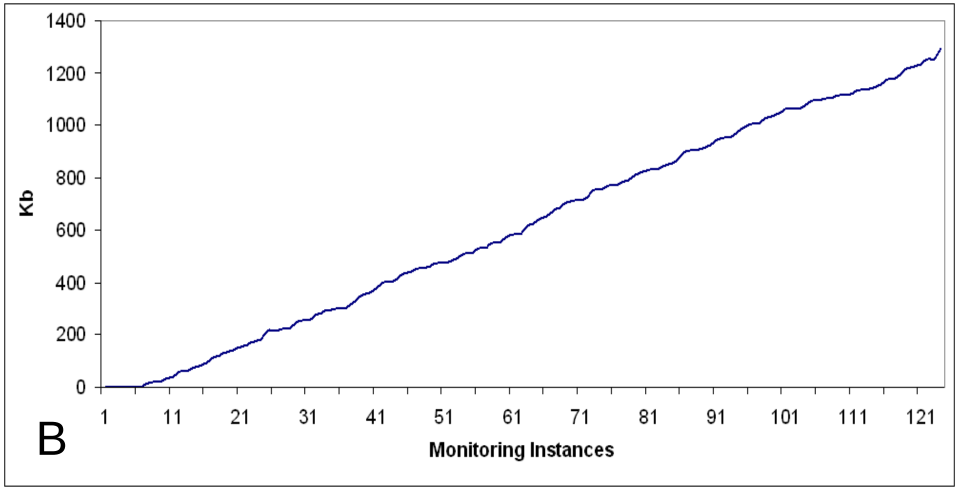
- The map built in the last experiment was:

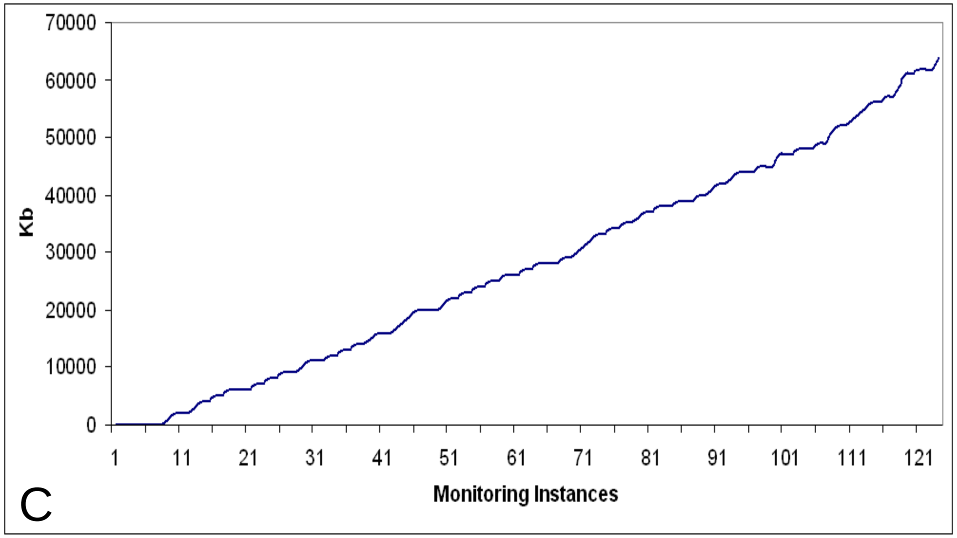


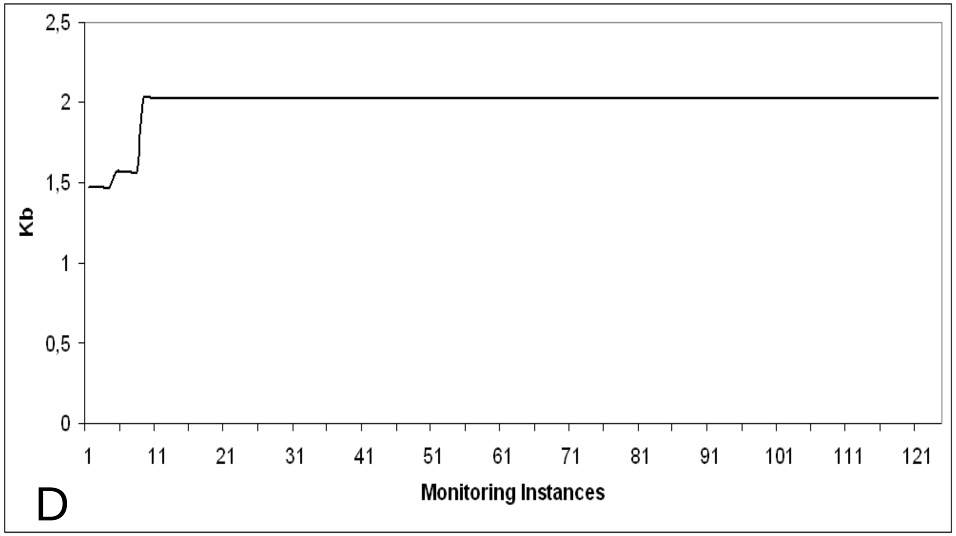
Experimental Case Study

- Effectiveness to determine a memory leak:
  – Four components inject a memory leak (A = 100Kb every injection, B = 10KB, C and D = 1MB):



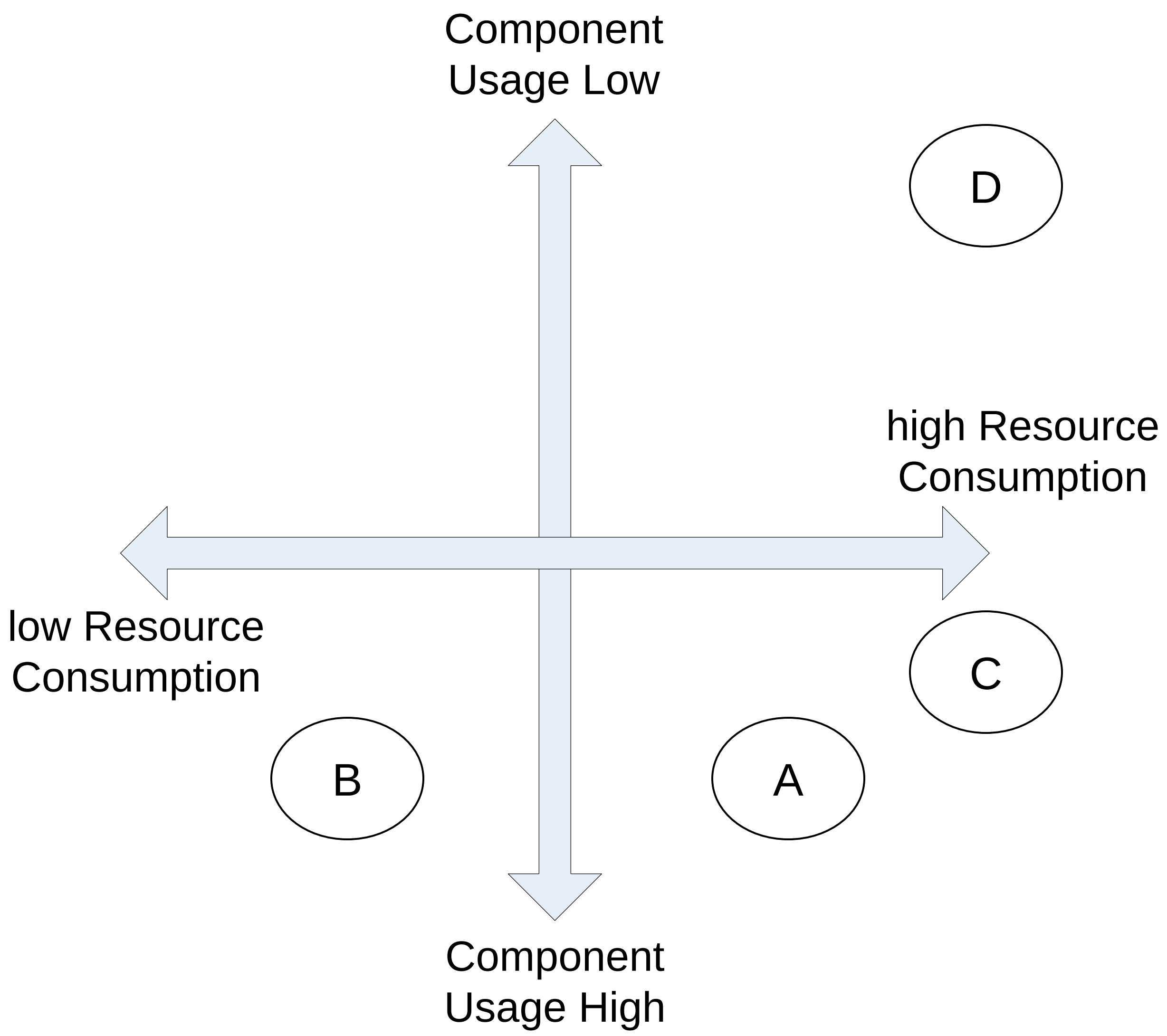






Experimental Case Study

- The map built in the last experiment was:



Conclusion & Future work

- We have presented our framework and its utility and effectiveness to help to determine the root cause failure.
- We have focused on one type of software aging: memory leaks.
- The resource-component consumption could be an useful tool to help to determine the riskiest component
- We have to evaluate the effectiveness of that approach to determine other type of software aging due to different resources or even an interaction of more than one resource.

J2EE Instrumentation for software aging root cause application component determination with AspectJ

Javier Alonso
Josep Ll. Berral
Ricard Gavaldà
Jordi Torres

Technical University of Catalonia, Spain