Active Control of Memory for Java Virtual Machines

Peter Westerink, Norm Bobroff, Liana Fong

IBM T.J. Watson Research Center

Yorktown Heights, NY

Issues with JVM memory management

- **A single JVM**
  - may take all allowed memory (heap size limit) while there is no performance benefit
  - may take memory but may not need it later when the workload changes
- **Multiple collocated JVMs**
  - may take or reserve much more total memory than is needed
  - Memory overcommit may result in swapping

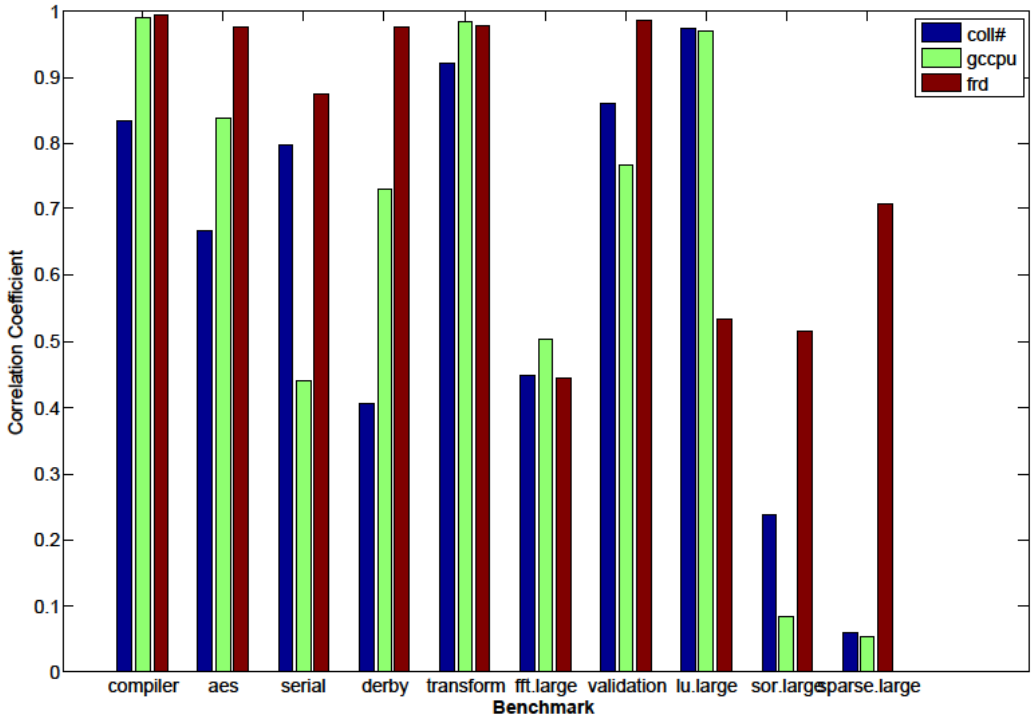
Solution

- **Overcommit JVM memory**
  - The sum of max heap size limit over all JVM's exceeds OS memory
- **Plus: move memory between JVM's**
  - Give more memory to JVM's that need it most
  - Remove memory from JVM's that do not need it or do not benefit

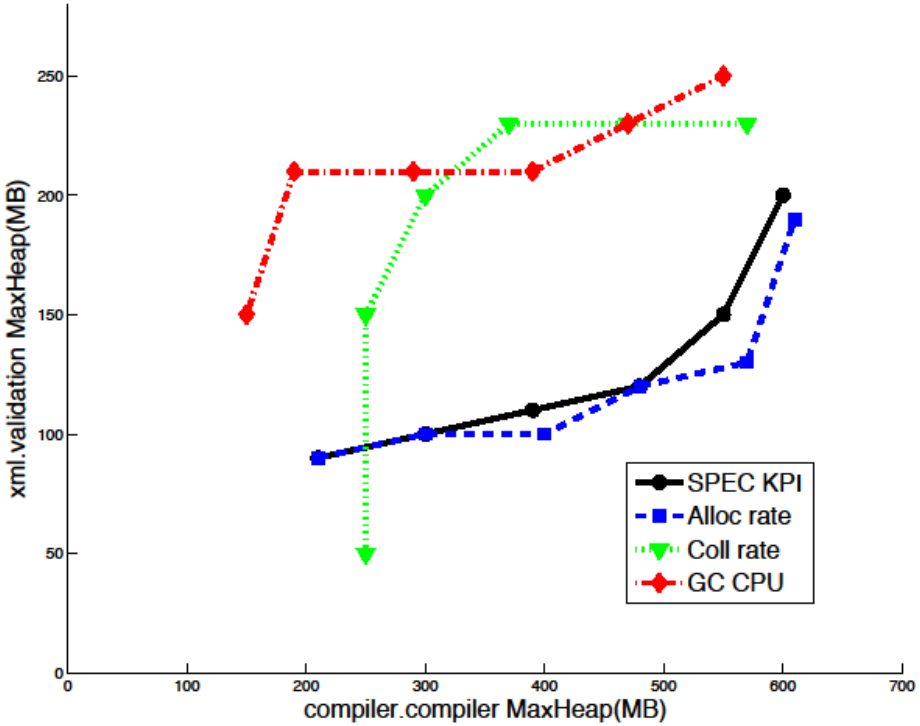
Performance measurement

- **Use JVM workload instrumentation?**
  - Typically not available
  - Even if available, interpretation is domain specific
- **Use available JVM metrics to infer workload performance:**
  - Available via JMX (Java Management Extension)
  - CPU, garbage collection count, memory freed

Correlation between KPI and JVM metrics



Memory balancing using KPI or JVM metrics



Dynamic changing of JVM memory

- **Use JVM ballooning**
  - Requires JVM plug-in
  - Requires hypervisor ballooning support, i.e. pinning/releasing of physical memory
- **Use IBM J9 JMX method to move heap size**
  - Lowering heap max followed by a GC releases memory to OS
  - Raising heap max allows JVM to take more memory
  - Control via JMX

Performance measure definition

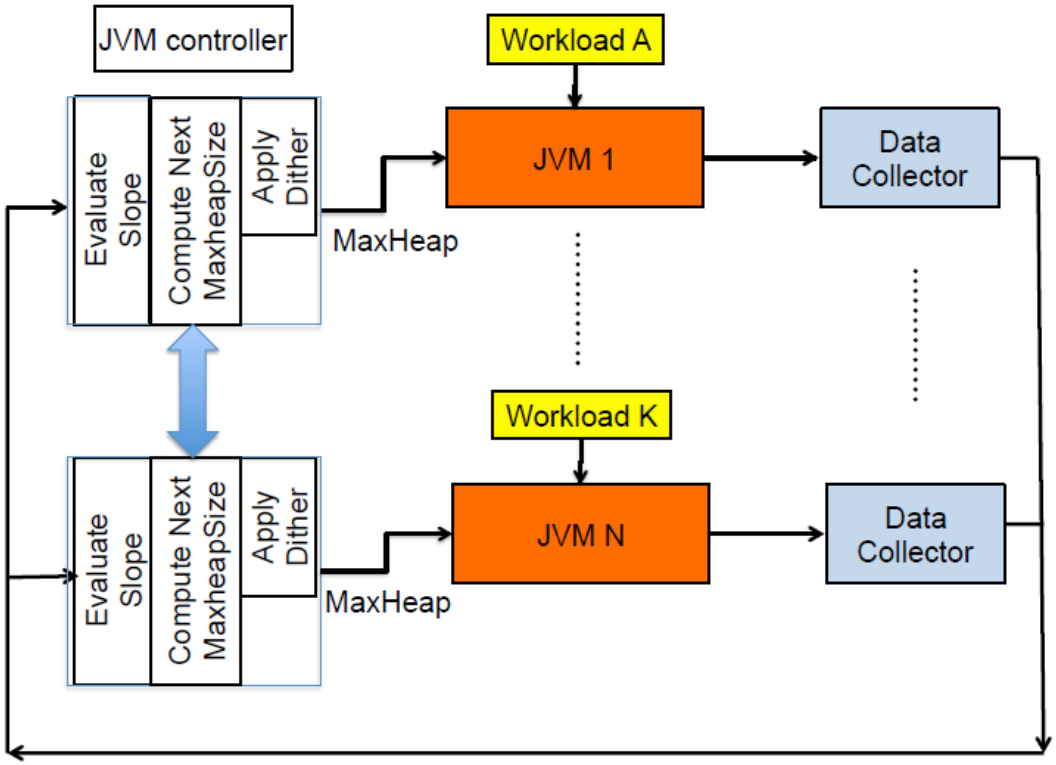
- **Define a *relative differential* performance measure:**
  - Differential: same the performance gain/loss % per fixed memory change
  - Relative: to compare the performance of different JVM workloads
  - The relative performance slope S(j) for each JVM j is defined as the slope of the curve of the application performance P(j) against MaxHeapSize, normalized by the performance value:

$$S_j = \frac{\Delta P_j}{\Delta MaxHeapSize} \times \frac{1}{P_j}$$

Real-time performance measurement

- **Measure slope for each JVM by "dithering" max heap size:**
  - Measure performance at current max heap size setting, then change setting to measure at another level
  - Continuously change the max heap size setting: "dithering"
- **Equate the relative slopes of all JVM's under a total memory constraint**
  - This will yield a new max heap size setting for each JVM

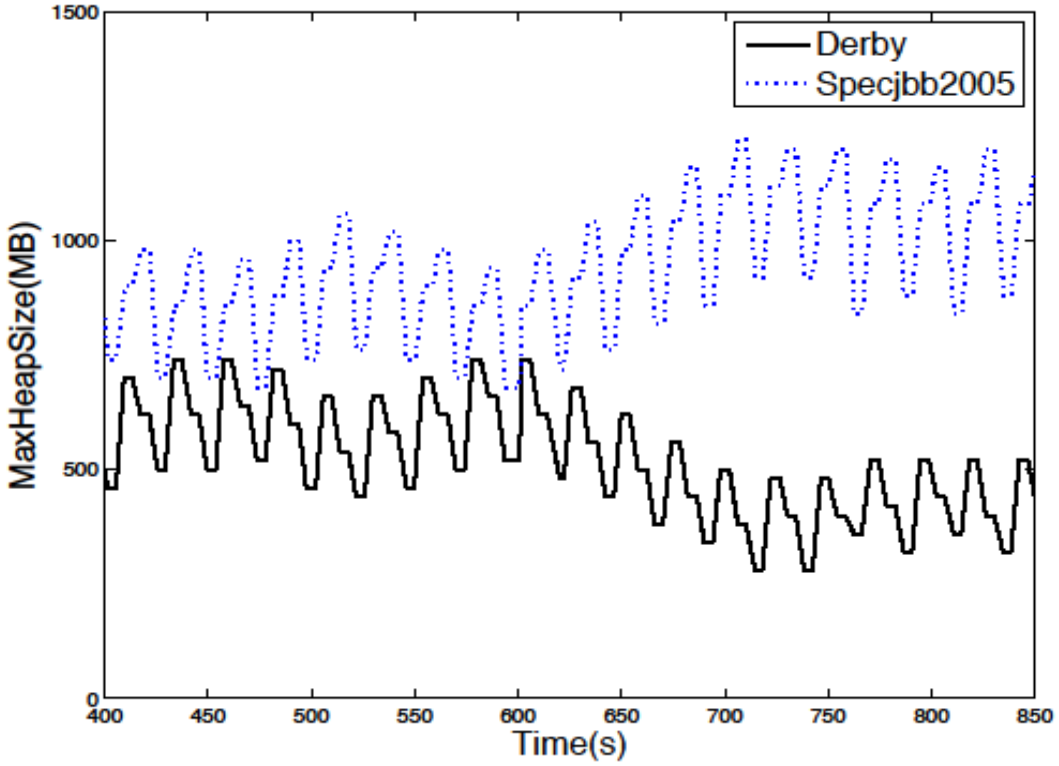
Real-time measure and control



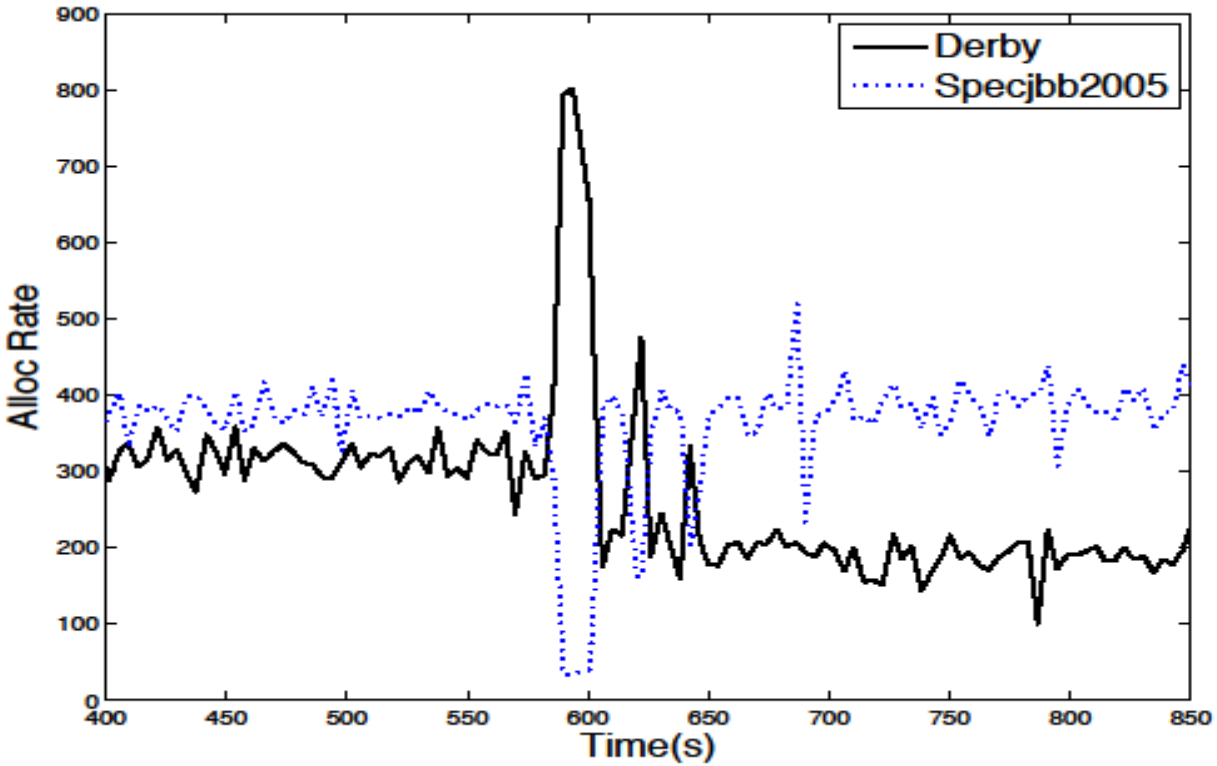
Experiment

- **Run SPECjvm2008 “derby” and SPECjbb2005**
- **Let run for a while**
- **Change the number of SPECjbb2005 warehouses from 10 to 20**

Experiment: max heap size when a workload changes



Experiment: allocation rate when a workload changes



Summary and Conclusions

- **Use JVM supplied metrics instead of relying on workload instrumentation**
- **Define relative differential performance measure**
- **Dither to continuously measure performance**
- **Optimize a global memory distribution**

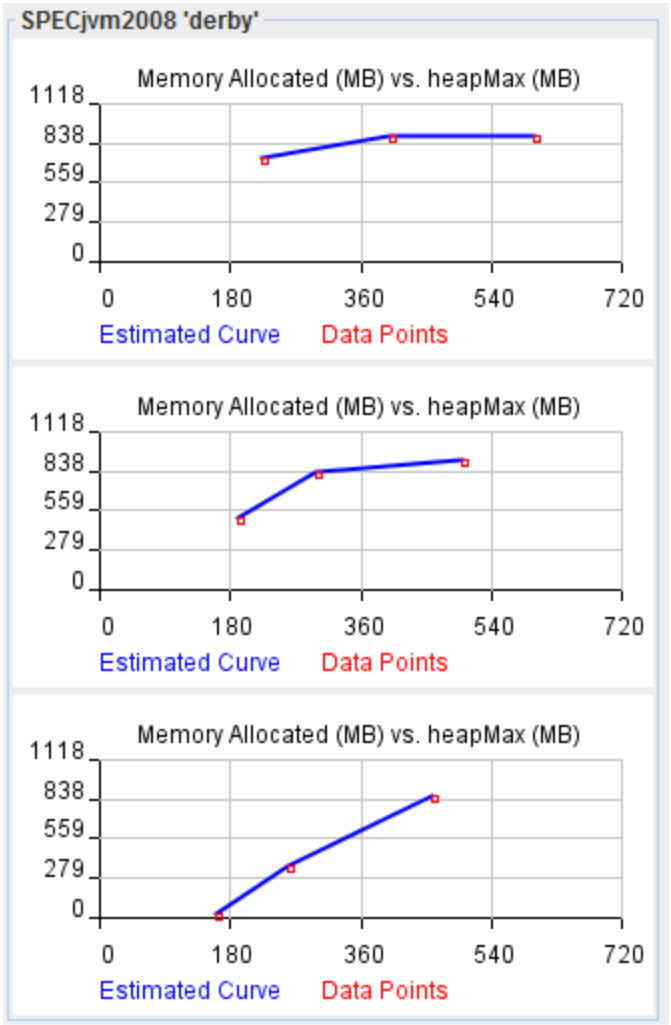
- **Workload changes result in a new memory distribution over the JVM's in the system**

Backup

Existing solutions to manage JVM memory

- **Assume JVM workload is instrumented to measure performance**
  - Typically not available
  - Even if available, interpretation is domain specific
- **Use JVM ballooning**
  - Requires JVM plug-in
  - Requires hypervisor ballooning support, i.e. pinning/releasing of physical memory
- **No global optimization**
  - Each JVM's is given enough memory to avoid performance drop
  - Severe memory shortage still results in swapping
- **Based upon how memory a JVM takes**
  - JVM may not benefit from memory and therefore unnecessarily takes too much memory

Example of measuring the performance curve



Experiment: measured curves when a workload changes