IBM

Applications Know Best:

Performance-Driven Memory Overcommit with Ginkgo

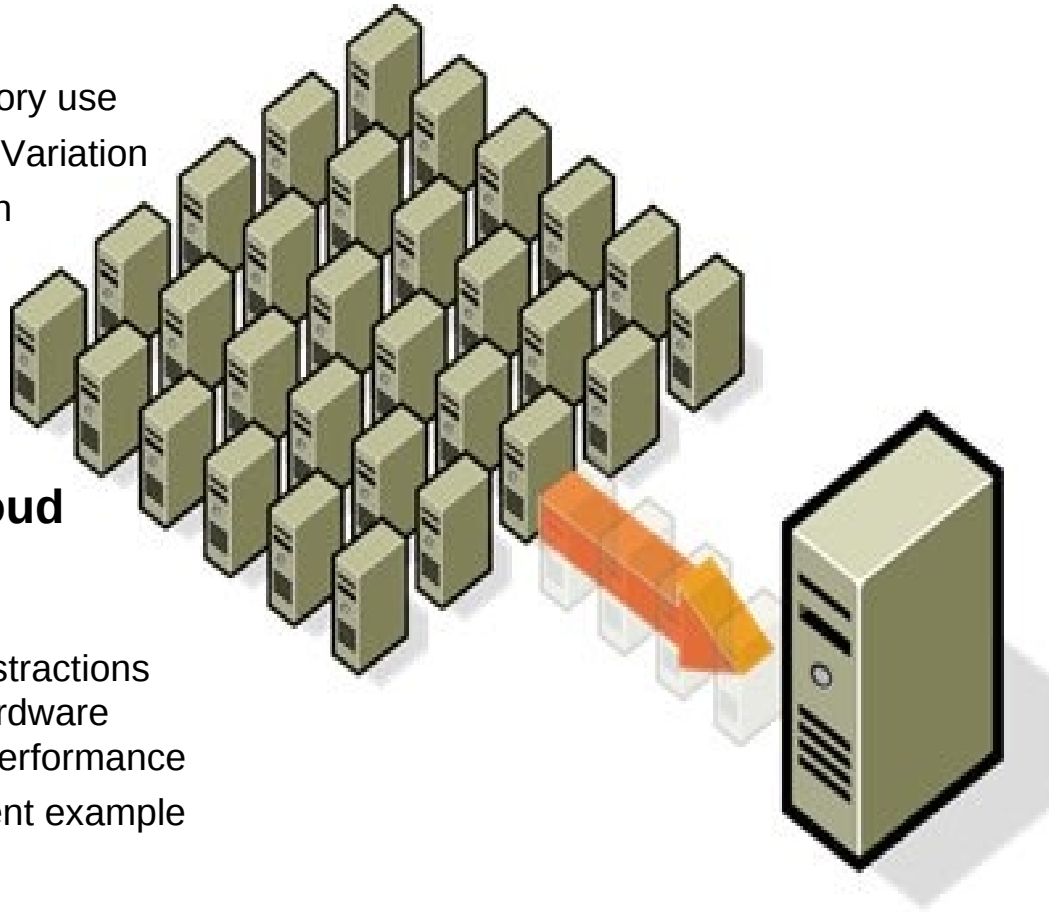
Michael R. Hines, Abel Gordon, Marcio Silva, Dilma Da Silva,
Kyung Dong Ryu, Muli Ben-Yehuda

IBM Research

IEEE International Conference on
Cloud Computing Technology and Science
November 2011

Motivation

- **Today's Cloud Landscape:**
  - Bigger machines => Higher memory use
  - More VM Churn => Performance Variation
  - Customers forced to roll-their-own resource management (Or be driven towards PaaS / SaaS solutions)



- **Memory over-commit for the Cloud**
  - Not commonly practiced (as far as we can tell)
  - Not possible without breaking abstractions
    Provider only understands hardware
    Customer only understands performance
  - We use ballooning as a preeminent example

- **Benefits of over-commit are large**
  - Saving money for provider
  - Potential discounts for customer
  - Energy savings is a bonus

Happy cloud provider ?

Reduce VMs memory footprint

Happy cloud customer ?

Maintain application performance

Happy Everyone?

Balanced Approach

Limited memory space

	No memory over-commit	Aggressive memory over-commit	Ginkgo
Provider			
Customer			

Assumptions

- **Assumption #1**: *Customer willing to expose application performance monitoring*
  - Most major server software products have such capabilities already

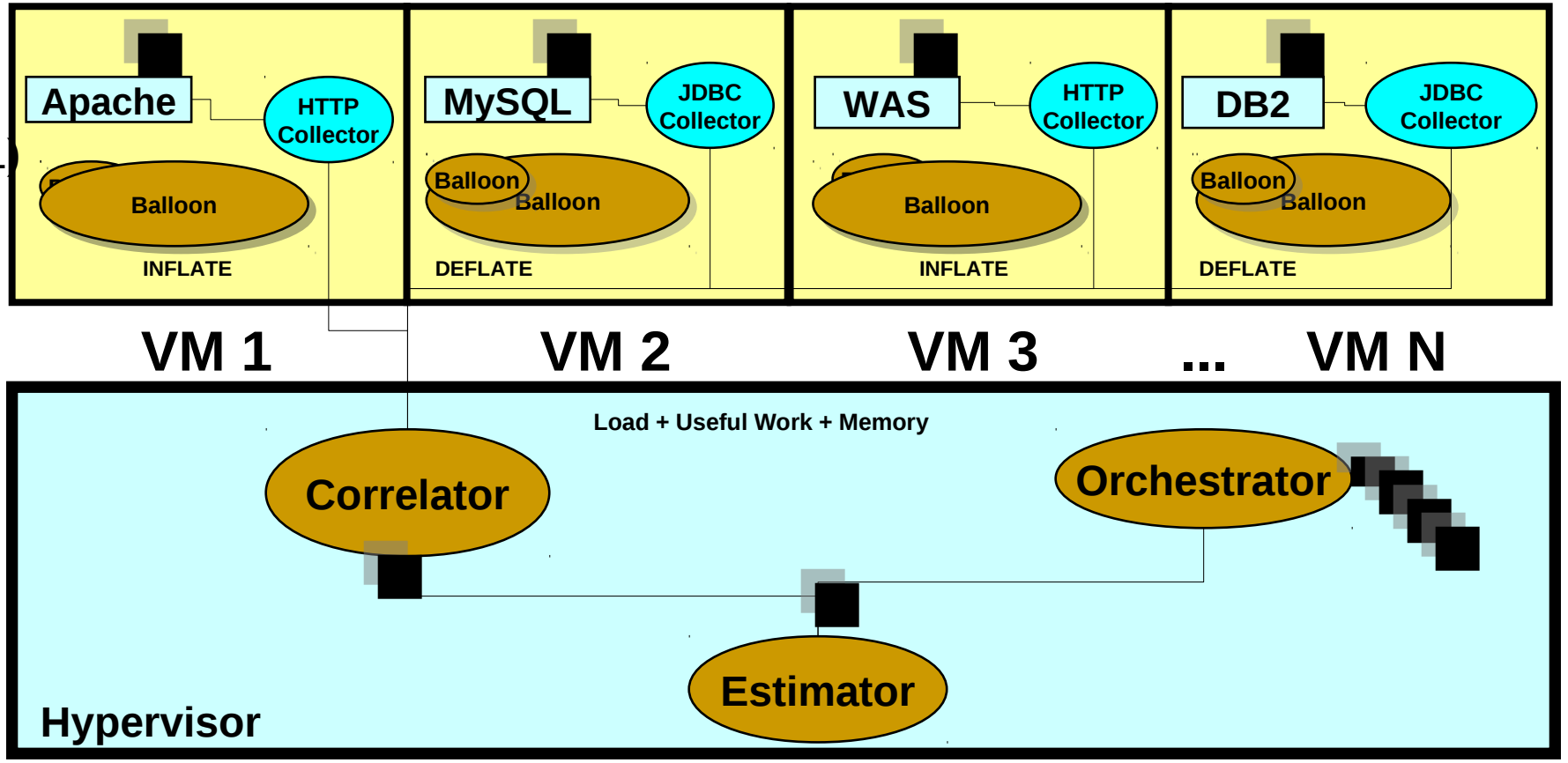
  (Note: This is on top of system-level monitoring.)

- **Assumption #2:** *Application is (or can be made to be) trustworthy.*
  - **Otherwise:** overcommitment will be disabled
    - Along with any incentives given to the customer
    - Overcommitment benefit for hypervisors will also be lost

- **Assumption #3**: *Application's memory footprint is resizable*
- **Assumption #4:** *Only one application per virtual machine*

  (Not a strict requirement, but simplifies the discussion)
- **Assumption #5**:
  - In this work, we assume load-injection arrival rate is known.
    - We understand that in practice, this is not possible.
  - In such cases, large deviations in our performance model will require that overcommitment in a production system ***be disabled***

Ginkgo System Design



1) Monitor application performance
2) Calibrate Model: F(Memory, Load) = Performance
3) Calculate memory assignment (**For example**: VM1 += 100MB, VM2 -= 100MB)
4) Deliver memory assignments to virtual machines
5) **Preview: ~73% Memory savings (50% not including free spaces)**

Calculating Memory Assignments

- **Simple LP Objective Function:**
  - Maximize application performance
  - LP is easy to program, even with small matrix
- **Inputs:**
  - Minimum / Maximum VM Memory Sizes
  - Application performance for every memory size
  - Load injection level for every memory size
- **Constraints:**
  - Hypervisor Memory should not exceed capacity
  - VMs receive only one assignment <= Configured Maximum

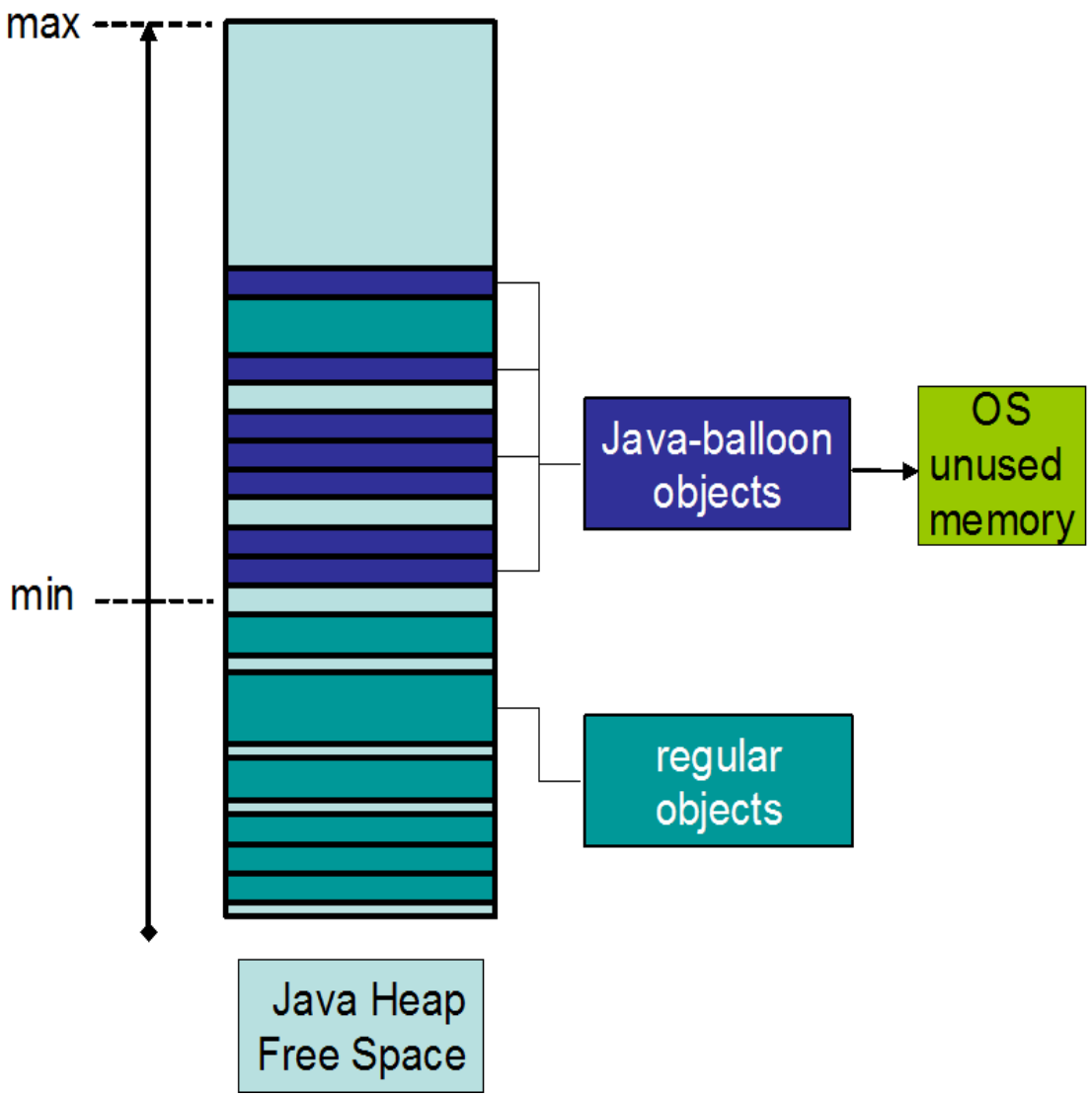
The Java Heap Resizing Problem

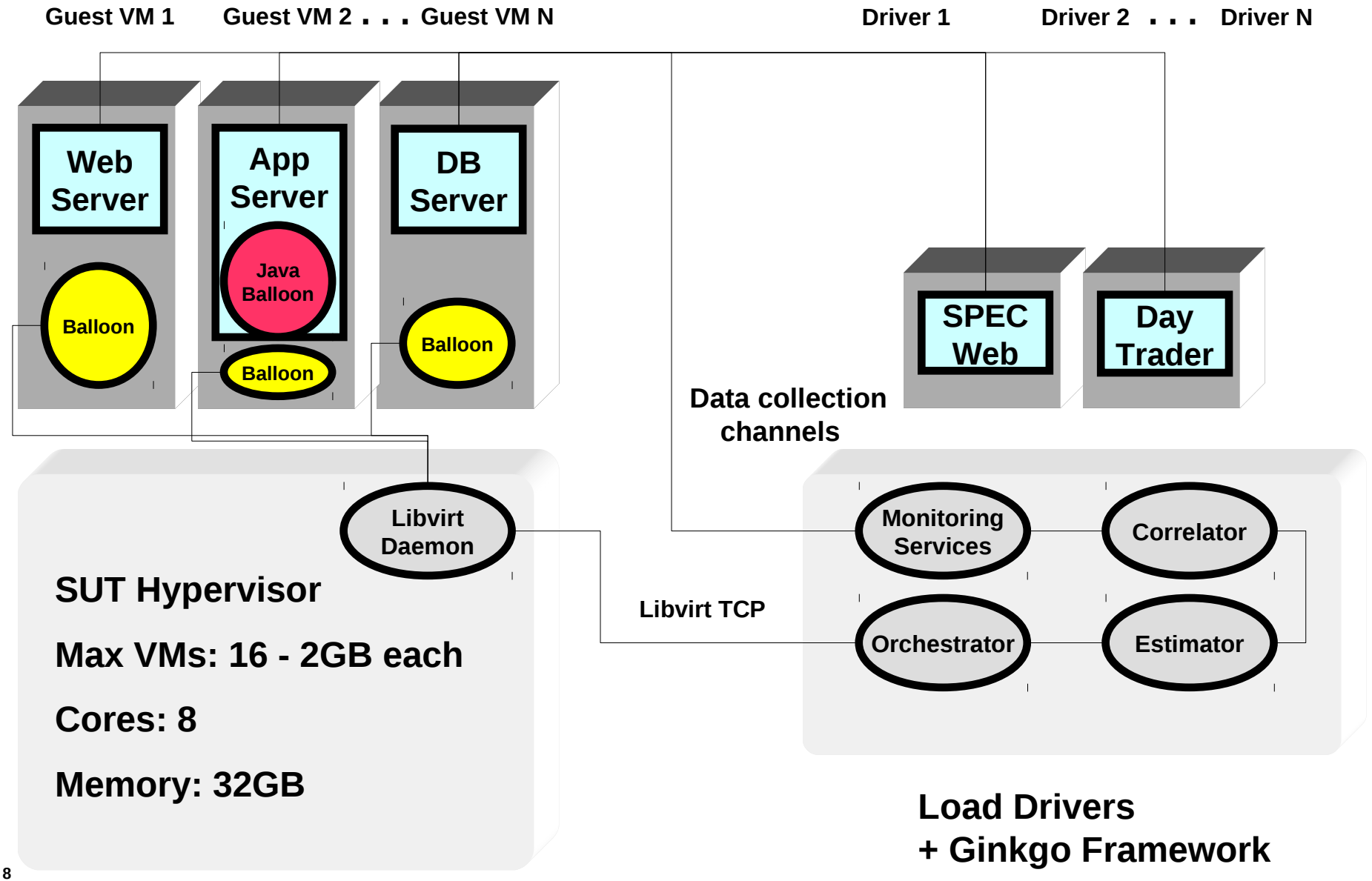
Problem: Java Heap is not resizable
It grows monotonically

Additional flexibility requirements
in Java became clear immediately

Ginkgo invokes Java balloon
- JVM allocates raw objects
- Objects are pinned
- Released to OS
- Then OS balloon is invoked

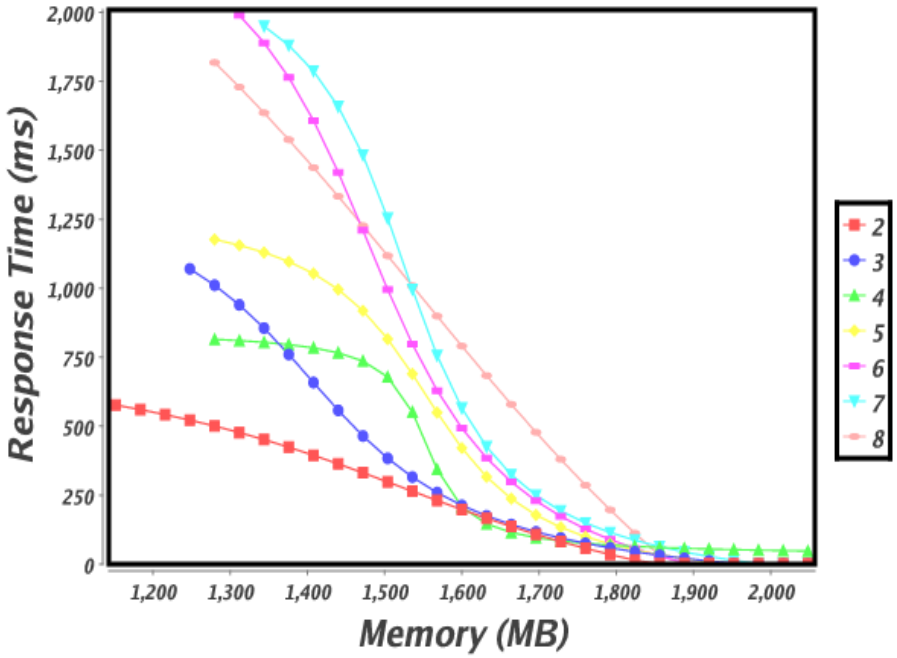
The database and webserver
in our experiments already had such flexibility – no modifications required.



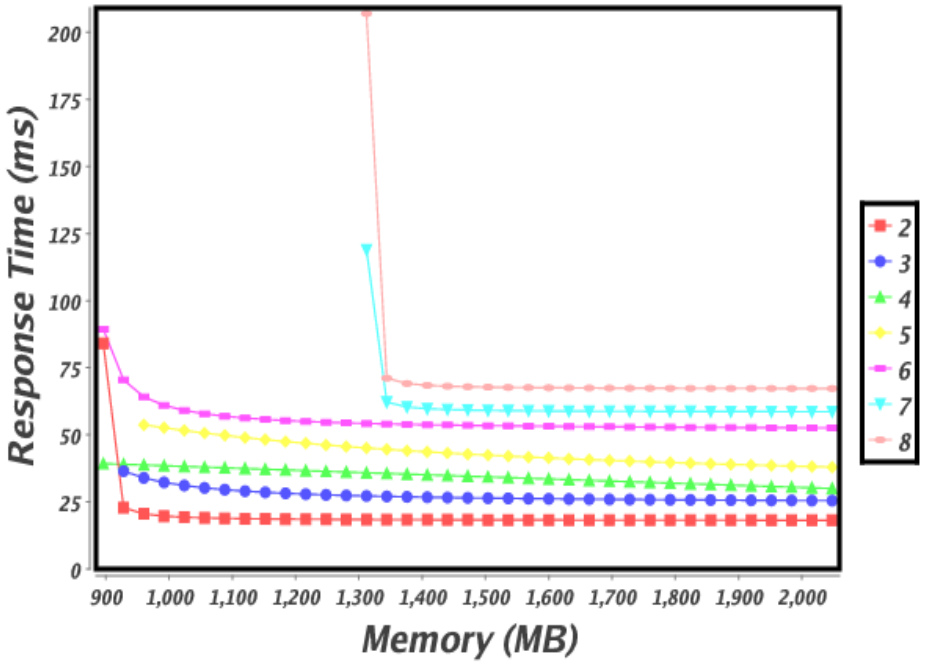

Guest VM 1
Guest VM 2 . . . Guest VM N
Driver 1
Driver 2 . . . Driver N
Web Server
App Server
Java Balloon
DB Server
Balloon
Balloon
Balloon
SPEC Web
Day Trader
Data collection channels
Libvirt Daemon
SUT Hypervisor
Max VMs: 16 - 2GB each
Cores: 8
Memory: 32GB
Libvirt TCP
Monitoring Services
Correlator
Orchestrator
Estimator
Load Drivers
+ Ginkgo Framework

Profiling Results (1)

WebSphere Monitored Performance vs. Memory



Java Balloon + WebSphere Performance vs. Memory

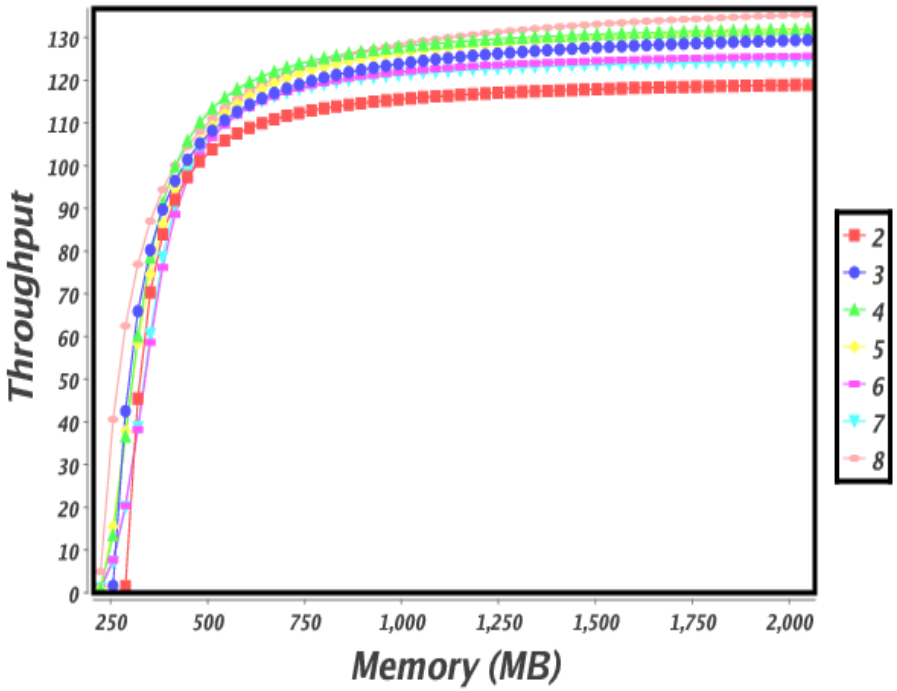


Y-axis: Performance (response time / throughput), X-axis: Memory size (up to 2GB)
Legend: Load injection level (# simultaneous clients)

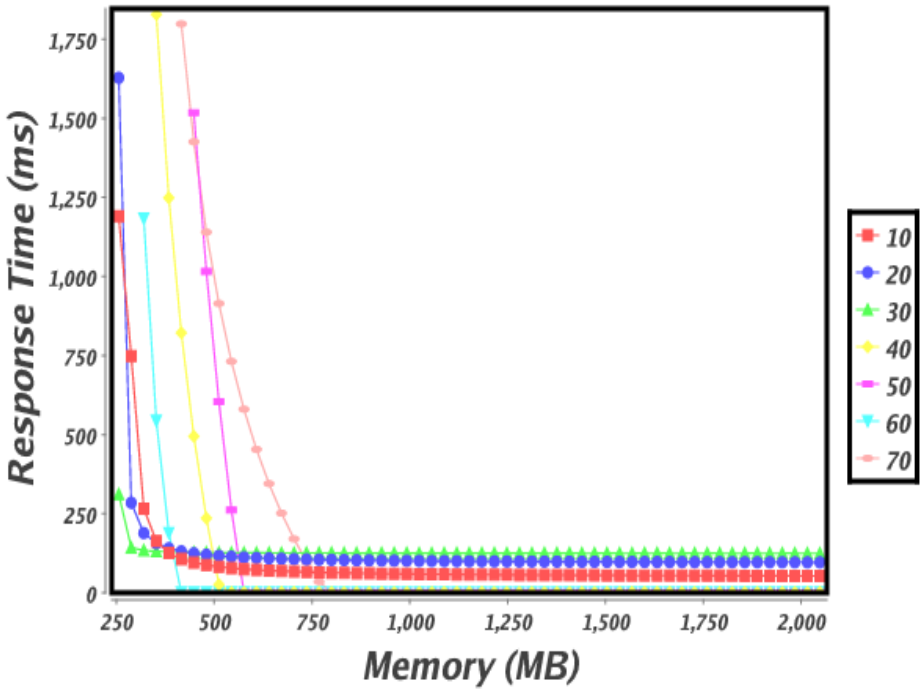
Without a java balloon, application performance is highly unstable.

Profiling Results (2)

DB2 Monitored Performance vs. Memory



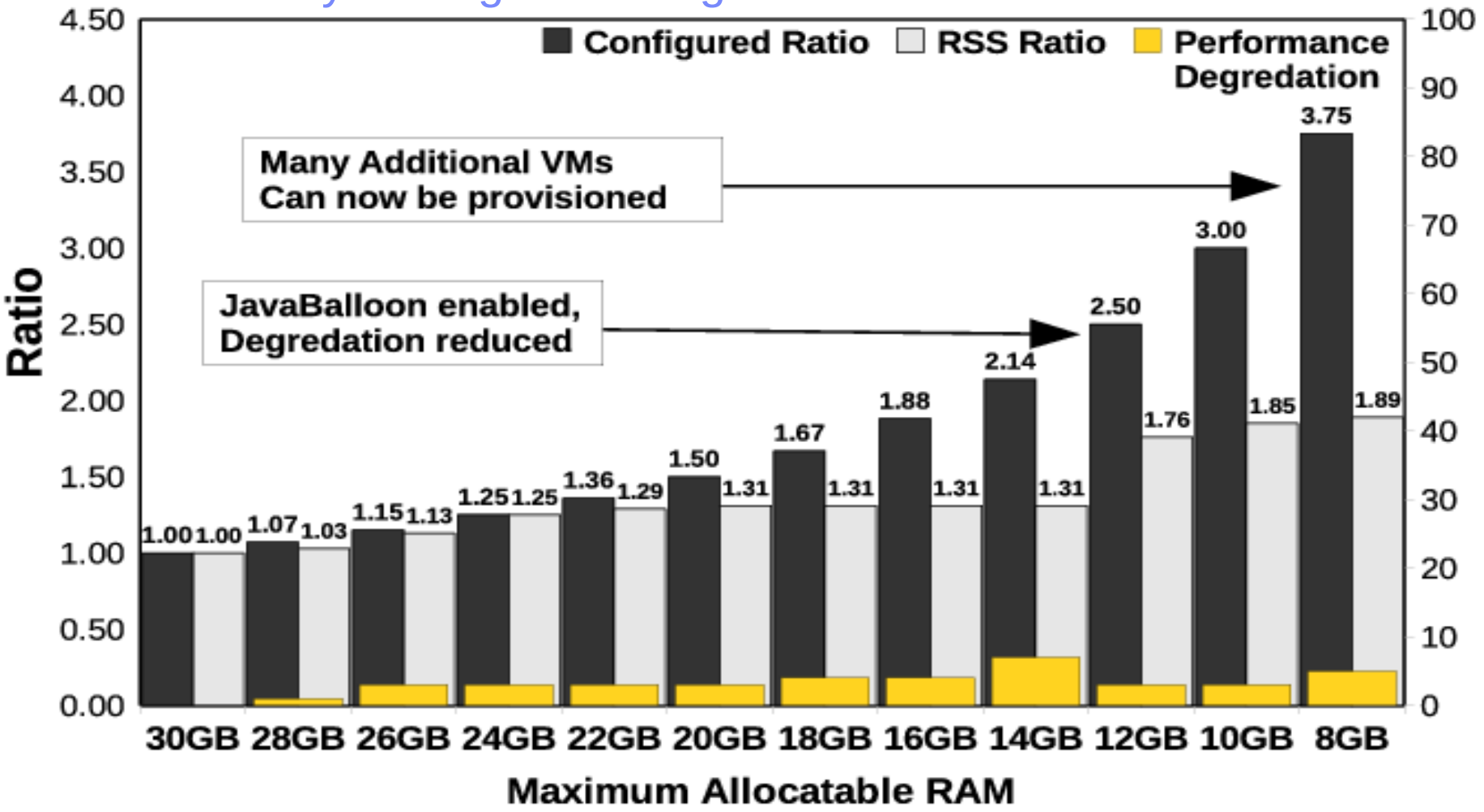
SPEC Web 2009 Monitored Performance vs. Memory



Y-axis: Performance (response time / throughput), X-axis: Memory size (up to 2GB)
Legend: Load injection level (# simultaneous clients)

DB2 / Apache are naturally self-resizable. Performance is stable with less memory.

Overall Memory Savings w/ Ginkgo Activated



Configured Ratio: Memory savings including free space
RSS Ratio: Memory savings ONLY from reclaimed application memory
Performance Degradation: Amount of performance loss calculated over the profiled baseline

Conclusions & Challenges

- Managing memory overcommitment mechanisms is a complex task and must be done carefully to avoid performance degradation
  - Can we bill customers around this model? Transparently? Is that feasible?
  - Can we assist cloud provisioning with this model?
- Application performance metrics can be used to infer memory needs and calculate efficient memory assignments
  - How do we isolate non-memory related factors that might affect the application performance ?
- Under certain loads, common workloads are not affected when running with less memory
  - Can we react fast enough to load variation during runtime ?

Questions?

Thanks!