University of Glasgow

The Judgment of Forseti

Economic Utility for Dynamic Heap Sizing of Multiple Runtimes

Jeremy.Singer@glasgow.ac.uk

David.Vengerov @ oracle.com

Callum.Cameron @ glasgow.ac.uk


image: wikipedia.org

Motivation

dynamic memory resource allocation in *datacenters*

image: wikipedia.org

dynamic memory resource allocation in *smartphones*

image: wikipedia.org

Requirements

- Satisfy users
- Be economical

Characteristics of VM tasks

- elastic memory usage
- phased behavior

Automatic Mem Mgt

- a.k.a. *Garbage Collection (GC)*
- Automatically deallocate a block of memory when it is no longer reachable
- *Runtime Heap* grows/shrinks on a demand basis

Key Heap Metrics

- *Live size* – current amount of *live* data
- *Current heap size* – current amount of *allocated* data (live and dead)
- *Max heap size* – max permitted value for current heap size

What is the
optimal
max heap size?

Lots of possibilities

How do you find the best settings for your system? … for your application?

1. domain expertise
2. exhaustive search
3. mathematical model

State-of-the-art: Domain Expertise

```
Java -Xmx12g -XX:MaxPermSize=64M -XX:PermSize=32M-XX:MaxNewSize=2g
     -XX:NewSize=1g -XX:SurvivorRatio=128 -XX:+UseParNewGC
     -XX:+UseConcMarkSweepGC -XX:MaxTenuringThreshold=0
     -XX:CMSInitiatingOccupancyFraction=60 -XX:+CMSParallelRemarkEnabled
     -XX:+UseCMSInitatingOccupancyOnly -XX:ParallelGCThreads=12
     -XX:LargePageSizeInBytes=256m ...
```

slide: azulsystems.com

State-of-the-art: Exhaustive Search

The Taming of the Shrew: Increasing Performance by Automatic Parameter Tuning for Java Garbage Collectors

Philipp Lengauer
Christian Doppler Laboratory MEVSS
Johannes Kepler University Linz, Austria
philipp.lengauer@jku.at

Hanspeter Mössenböck
Institute for System Software
Johannes Kepler University Linz, Austria
hanspeter.moessenboeck@jku.at

ABSTRACT

Garbage collection, if not tuned properly, can considerably impact application performance. Unfortunately, configur-

However, while object allocations produce a direct and easy to understand performance impact, the costs of garbage collections are easily overlooked. Programmers are often un-

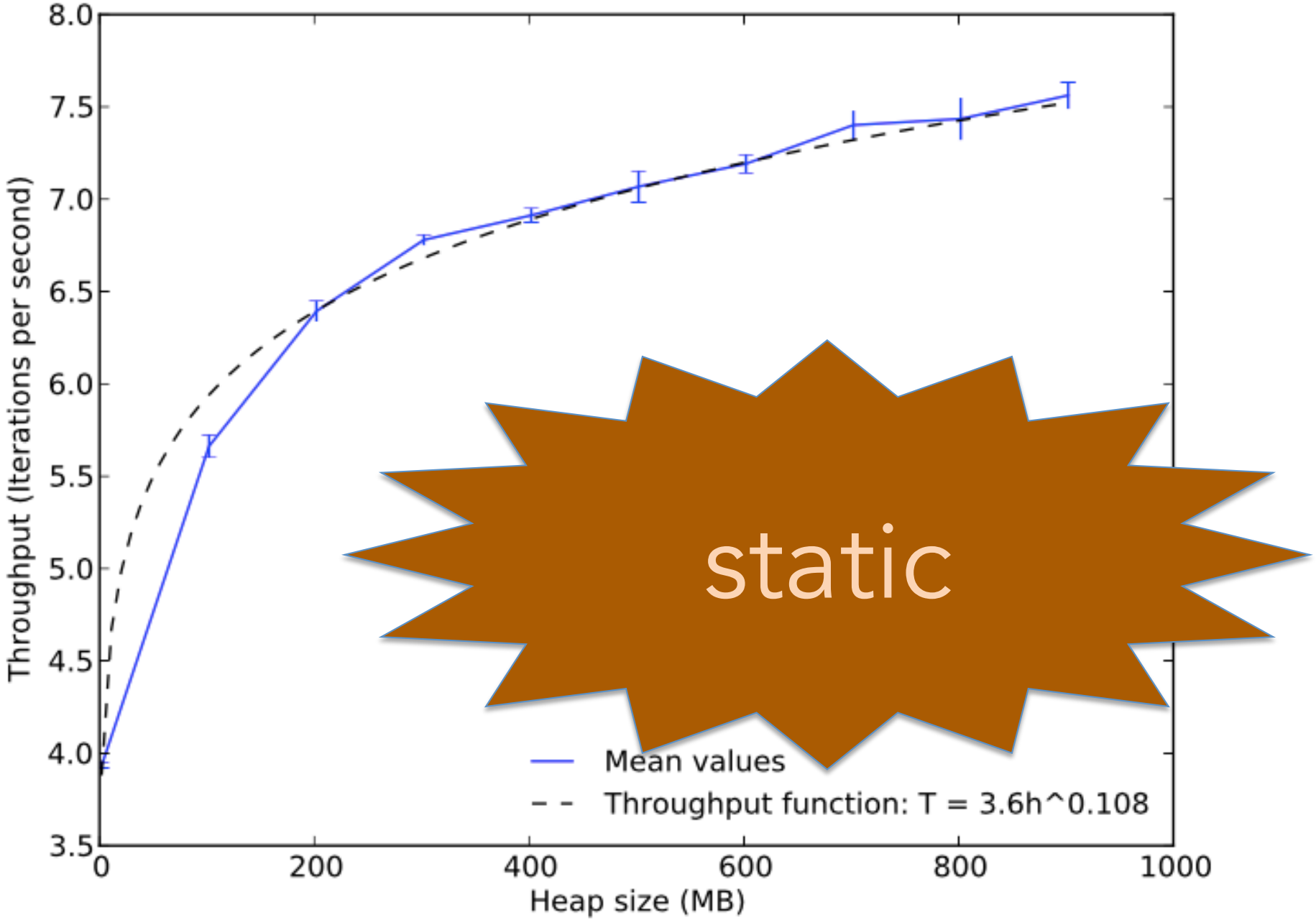
- around 300 GC parameters
- search parameter space for 4 hours
- select *best* configuration

[ICPE 2014]

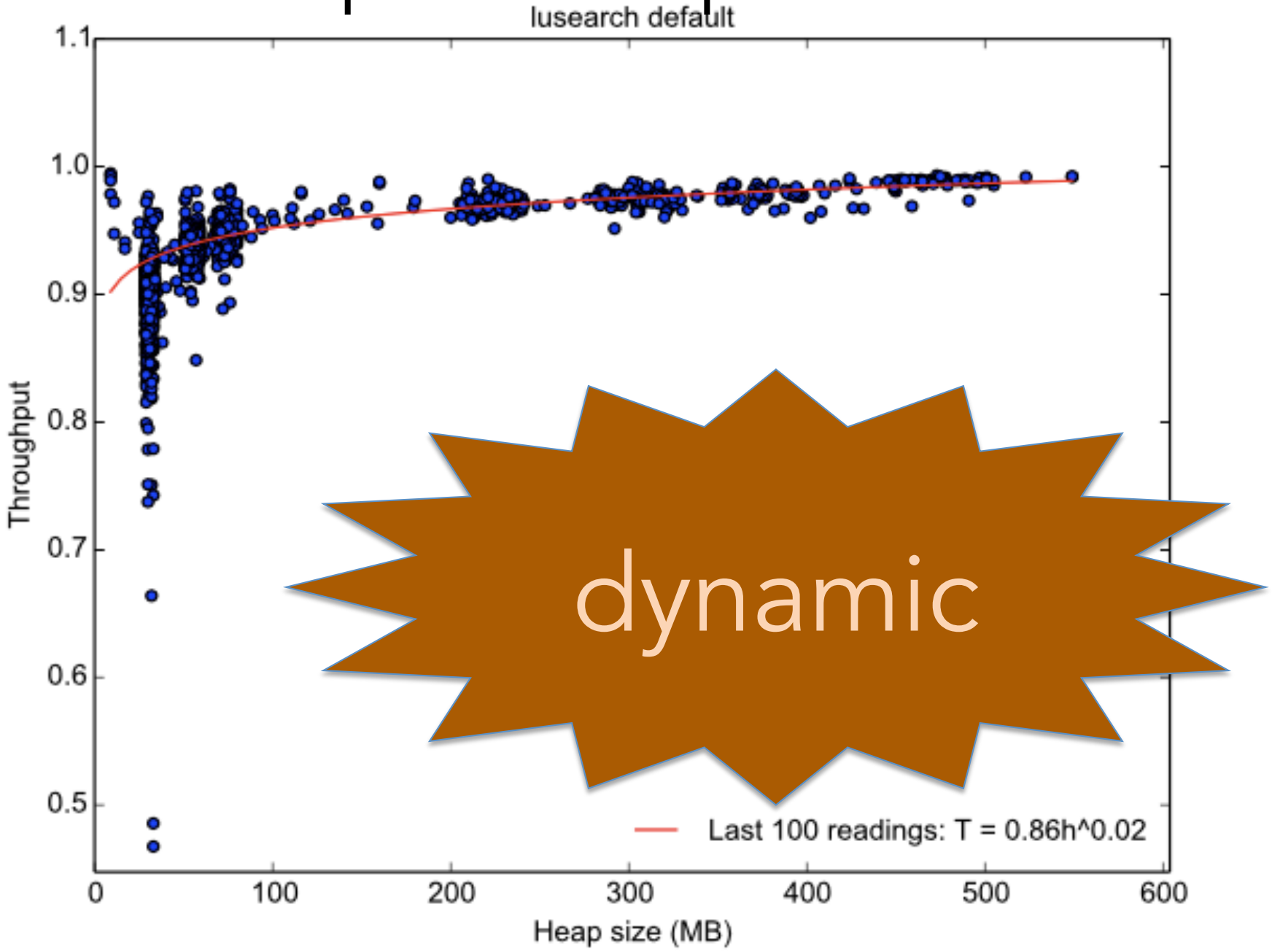
State-of-the-art: Mathematical Model

- decision tree
  - machine learning [ISMM 2007]
- supply/demand curve
  - economics [ISMM 2010]
- differential equations
  - control theory [ISMM 2013]

Max heap affects performance



Max heap affects performance



Math Model based on *economic utility*

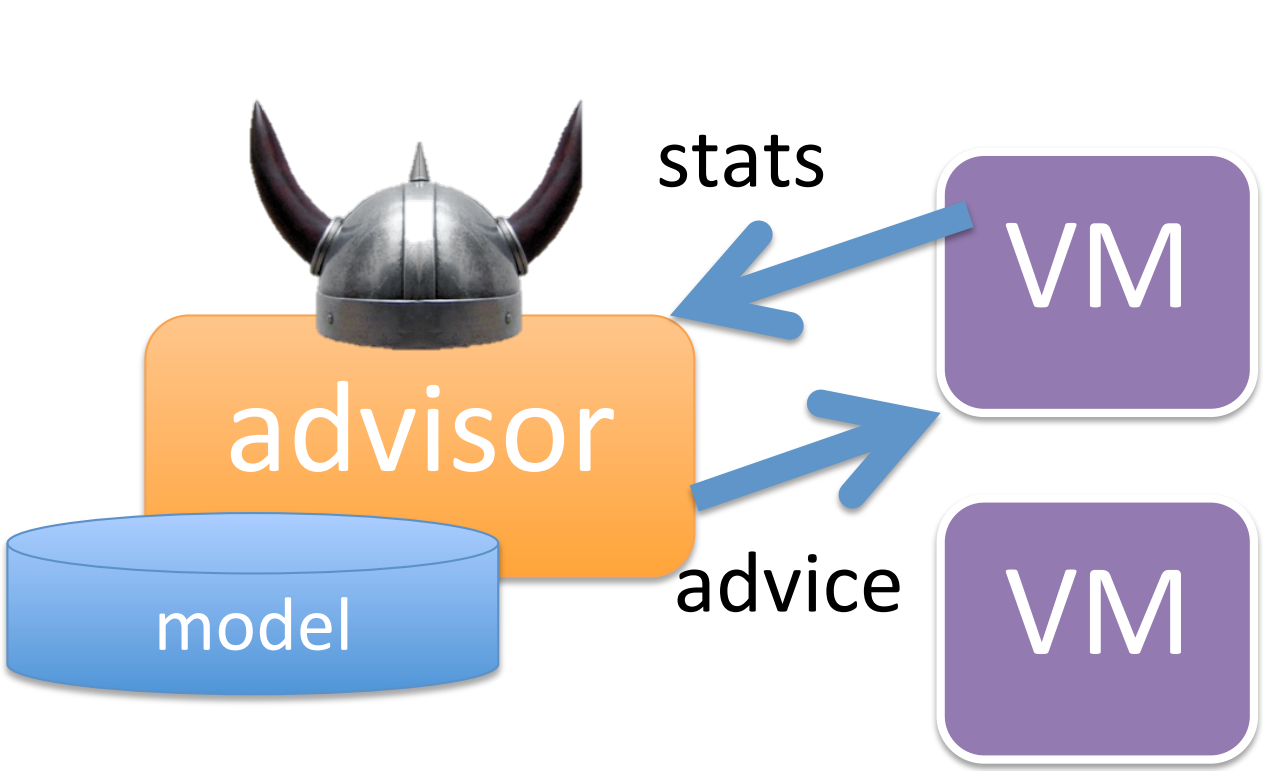
- utility function for individual VM

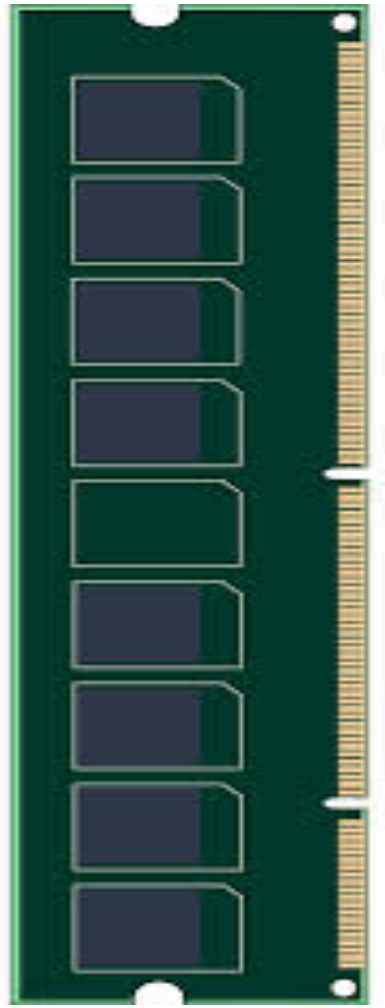
$$\mathrm{U}(h) = \mathrm{a}h^{\mathbf{b}}$$

- overall utility function for whole-system
  - product of individual utilities

- maximise overall utility function
  - use numeric optimization

Forseti concept

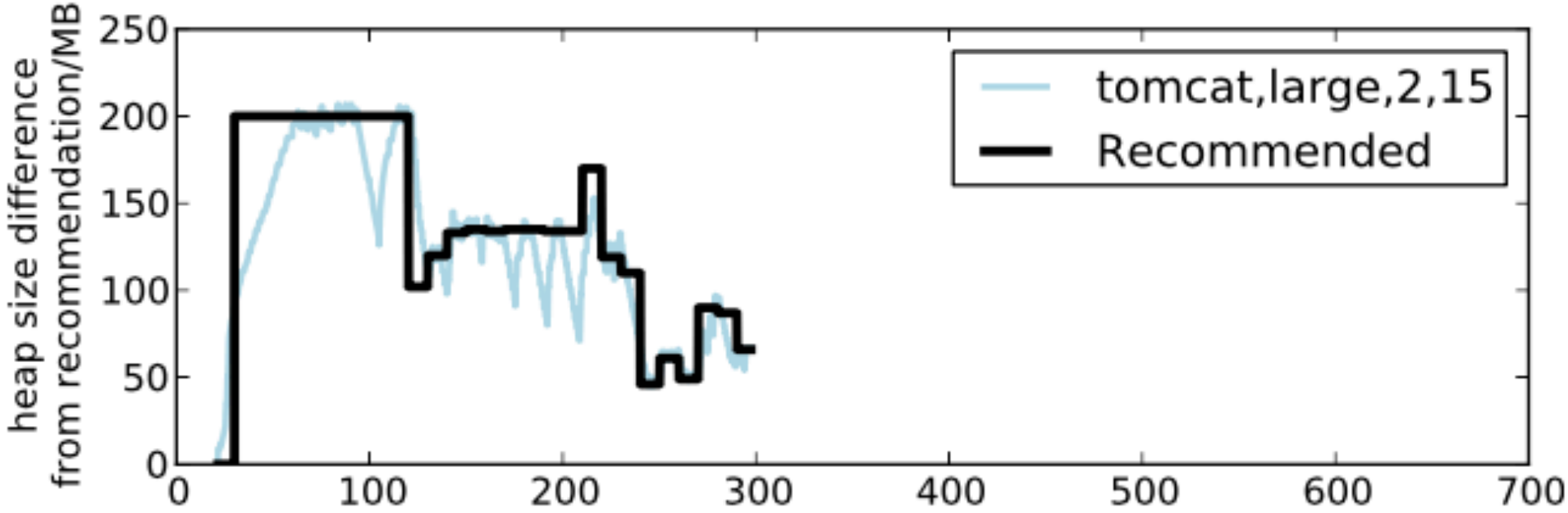


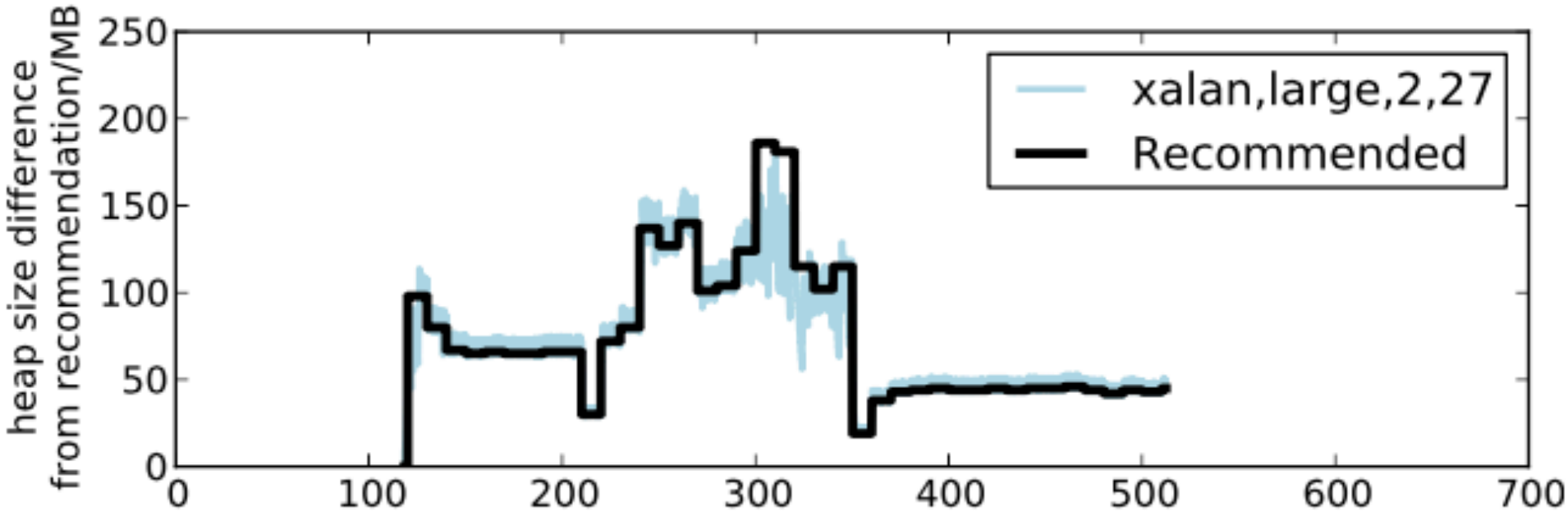


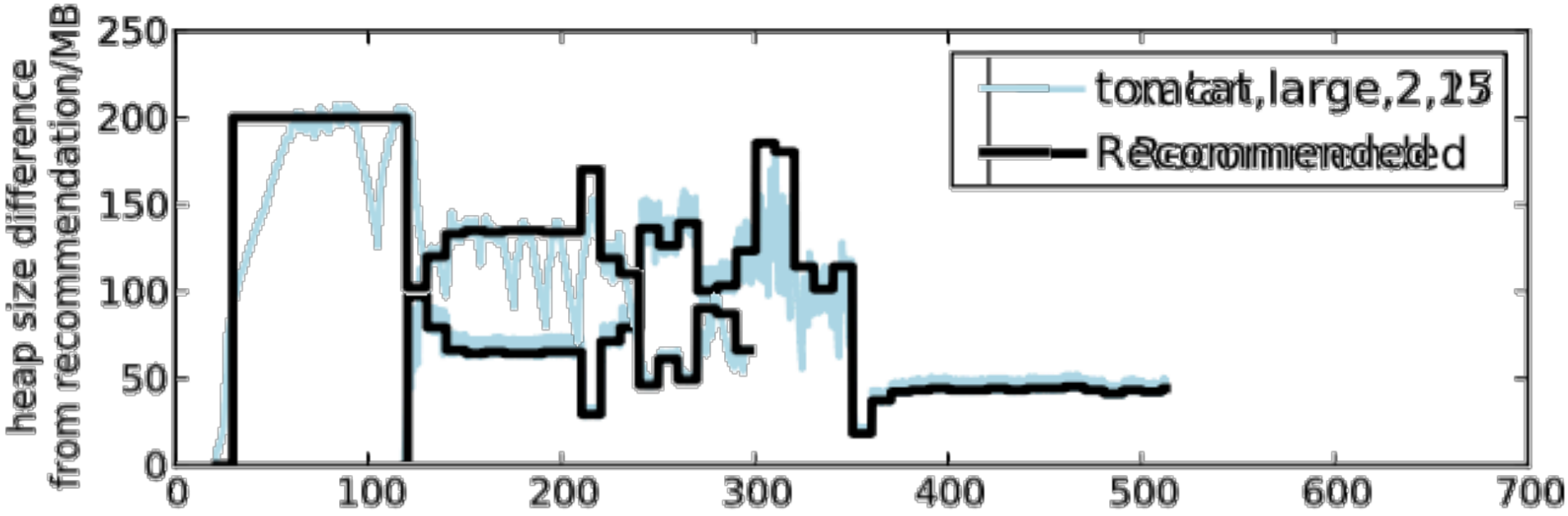
Evaluation

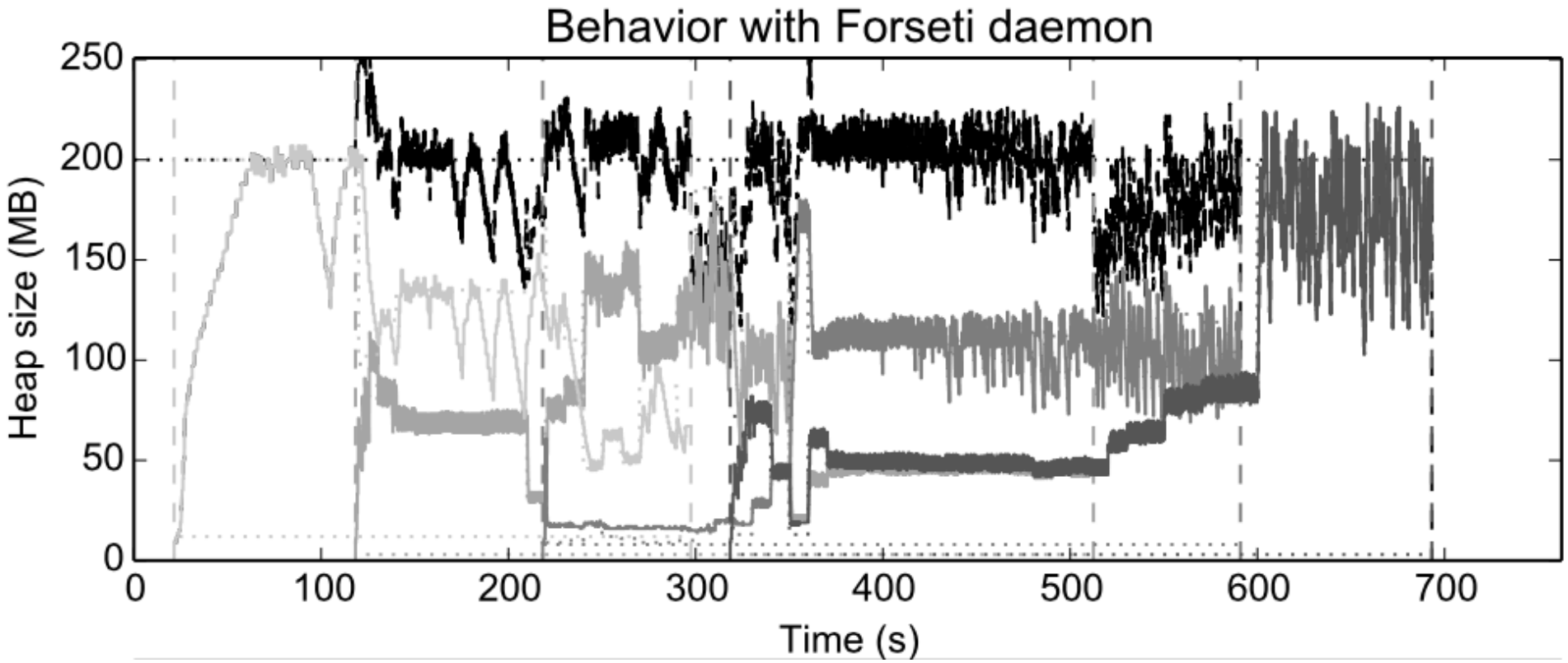
Experiment 1:

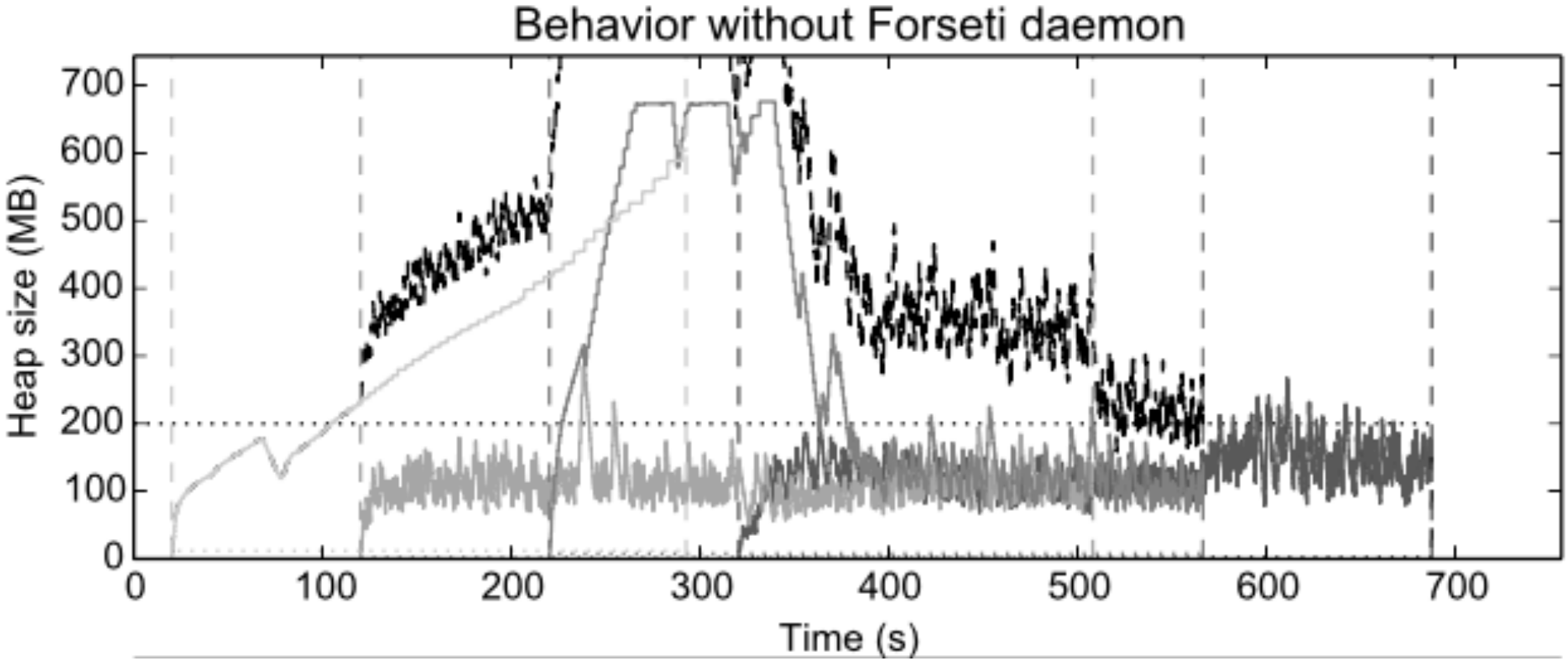
- run *4* Java benchmarks
- staggered start times
- set target total mem usage to *200MB*


heap size difference
from recommendation/MB
250
200
150
100
50
0
0
100
200
300
400
500
600
700
tomcat,large,2,15
Recommended


heap size difference
from recommendation/MB
250
200
150
100
50
0
0
100
200
300
400
500
600
700
xalan,large,2,27
Recommended


heap size difference
from recommendation/MB
250
200
150
100
50
0
0
100
200
300
400
500
600
700
tomcat,large,2,15
Recommended

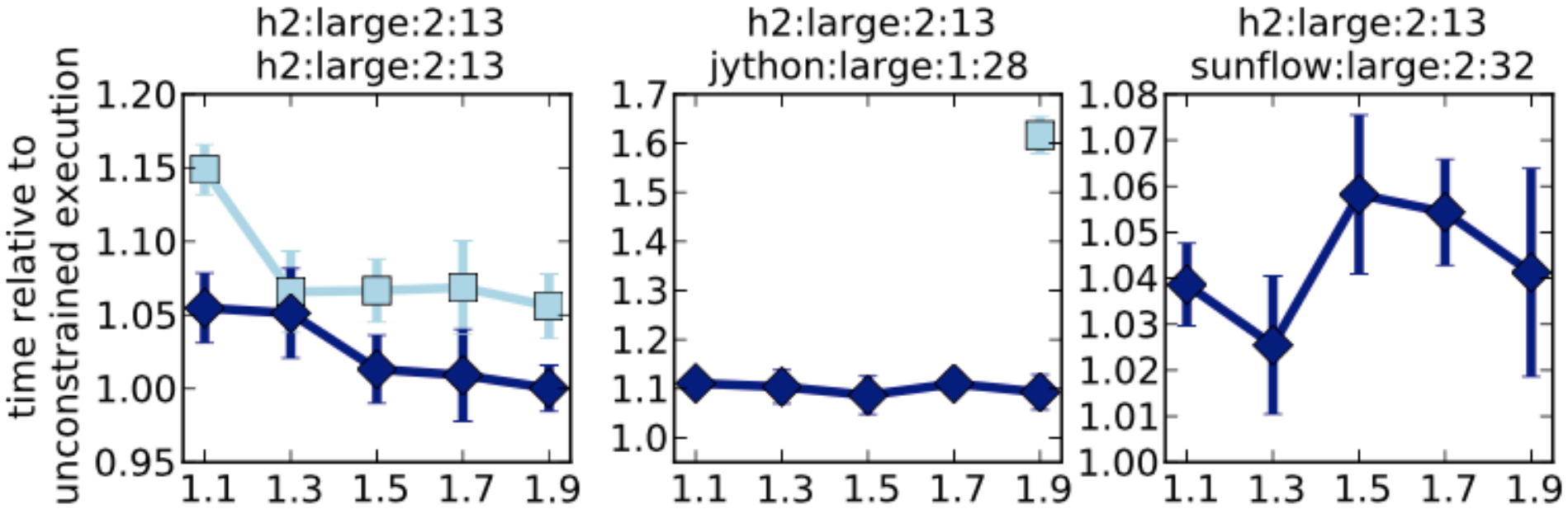

Behavior with Forseti daemon
Heap size (MB)
250
200
150
100
50
0
0
100
200
300
400
500
600
700
Time (s)


Behavior without Forseti daemon
Heap size (MB)
0
100
200
300
400
500
600
700
Time (s)

Experiment 2:

- run pairs of Java benchmarks concurrently
- Set target total mem usage to 1.1..1.9 x minheap
- compare execution time with
  - Forseti
  - Static fixed heaps
  - Unconstrained sizing

Relative Performance



Overheads

In all reported experiments, the time overhead for running the Forseti daemon is small. We analyzed the 6104 experimental runs completed for this paper:

- mean experiment wall clock time is 412 seconds (max is 2300 seconds).
- mean daemon CPU time is 1.00 seconds (max is 5.94 seconds).
- mean daemon memory footprint is 23MB (max is 29MB).

Conclusions

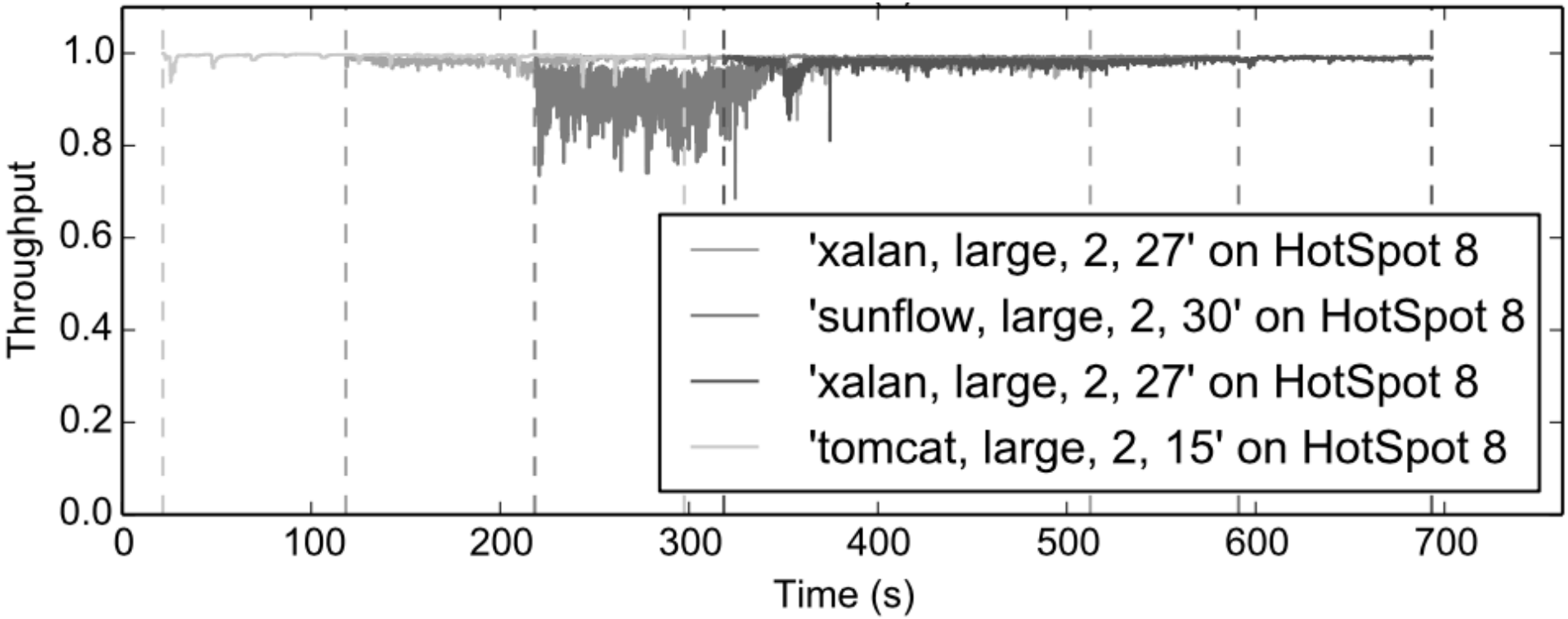
Garbage Collectors require *Holistic* Systems Optimization

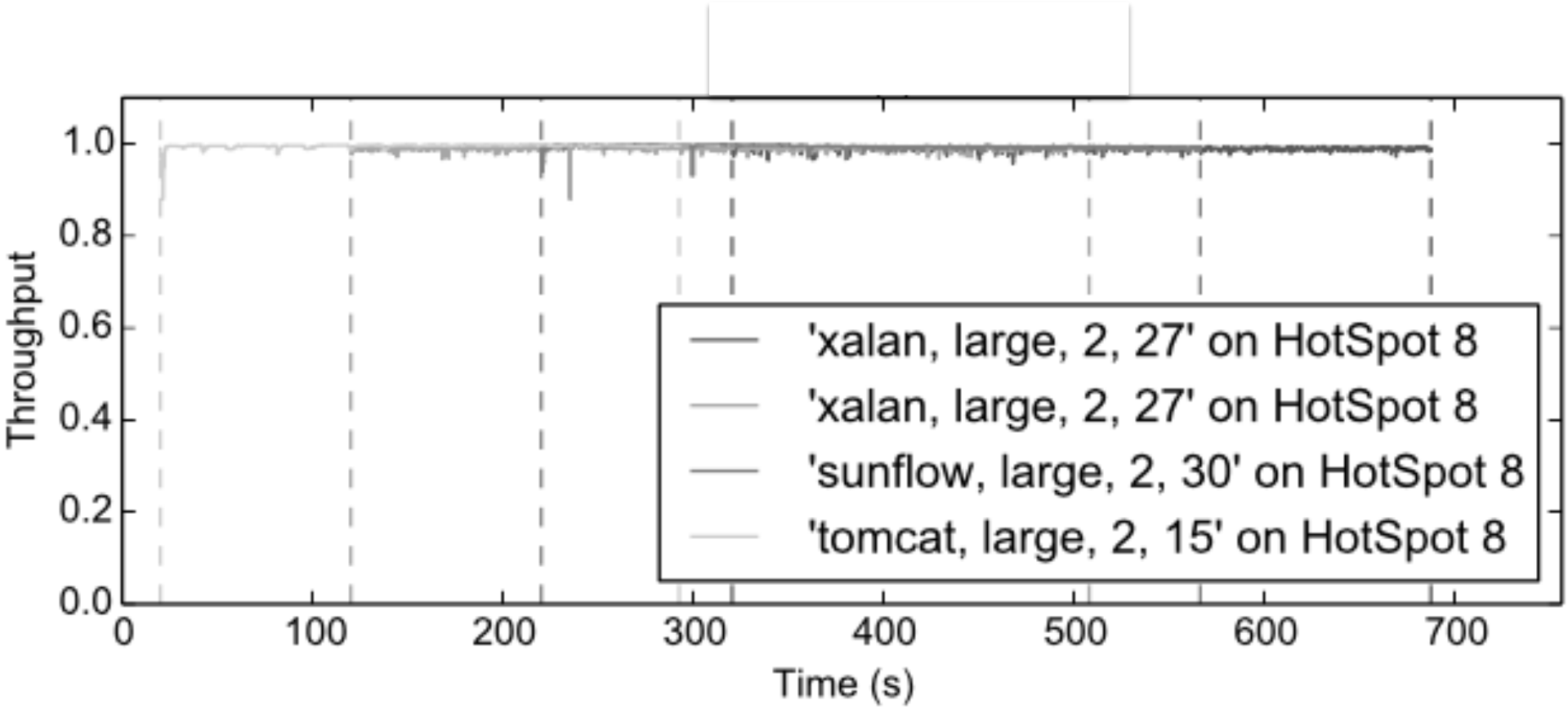
- Model must consider all VMs in system
- Optimize holistically, not in isolation

- Is this a new OS service?
- Generalizability?

end of presentation

Throughput for staggered multi-VM experiment




Throughput
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
600
700
Time (s)
'xalan, large, 2, 27' on HotSpot 8
'xalan, large, 2, 27' on HotSpot 8
'sunflow, large, 2, 30' on HotSpot 8
'tomcat, large, 2, 15' on HotSpot 8