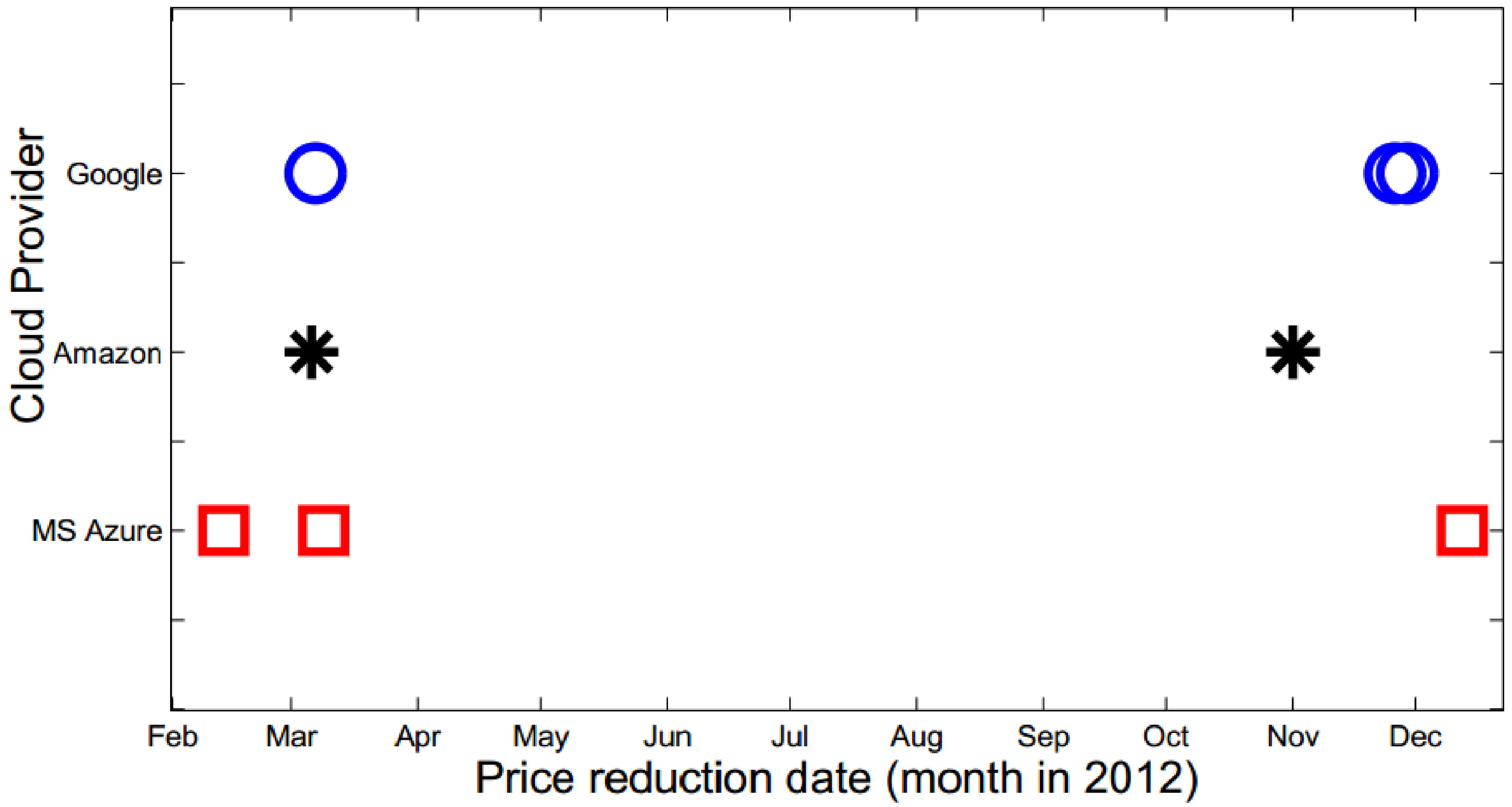
Raas+Ginseng: Resource Allocation in the Cloud

Orna Agmon Ben-Yehuda
Muli Ben-Yehuda
Eyal Posener
Ahuva Mu’alem
<u>Assaf Schuster</u>

Cloud wars – A Multi–Billion Market



The next thing after IaaS?

IaaS Trends:

- **Fine time granularity**
- **Fine resource granularity**
- **Tiered service levels**
- **Market-driven resource pricing**

Solution:

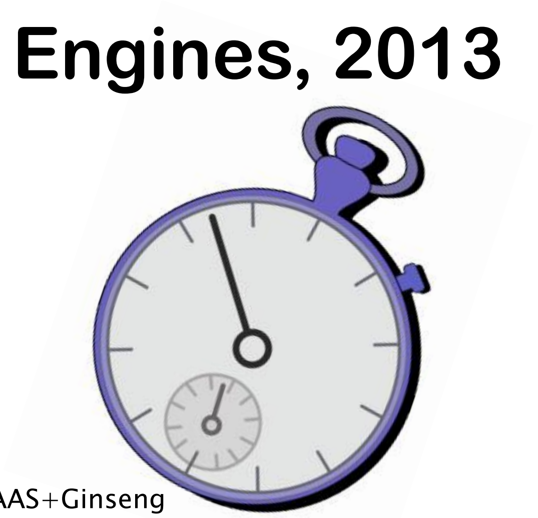
RaaS – Resource as a Service cloud

[Hotcloud 2012, CACM 2014]

Trend: Shrinking Duration of Rent

- 3 years: hardware purchase
- Months: web hosting
- Hours: EC2 on-demand (pay-as-you-go), 2006
- 5 minutes: CloudSigma, EC2 Spot Instances, 2009
- 3 minute: GridSpot, 2012
- 1 minute: ProfitBricks, 2012,
  Google App+Compute Engines, 2013

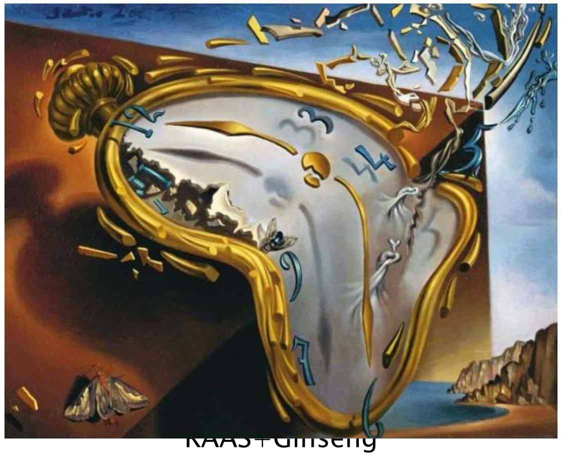
- Analogies: Car rental, telecom



Extrapolation: Duration of Rent

- Clients want to pay for resources only when they need them.
- Clients need extra resources to be allocated within seconds (e.g., when slashdotted)

- We extrapolate that cloud resources will be rented by the second.



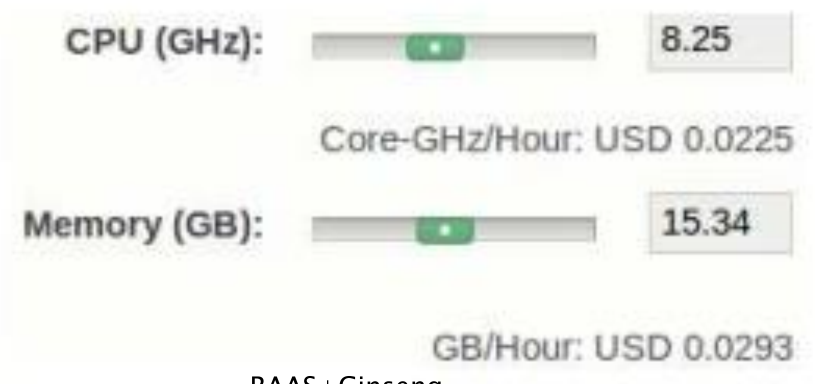
Trend: Flexible Resource Granularity

- Most cloud providers sell fixed bundles, called “instance types” or “server sizes”.
- Amazon allows adding and removing “network instances” and “block instances”, thus dynamically changing I/O resources, 2012.
- CloudSigma, 2010, Gridspot and ProfitBricks, 2012, offer clients to compose a flexible bundle.
- Google App Engine charges I/O op’s by the million + progressive network prices.

Extrapolation: Resource Granularity

- As physical servers grow up, an entire server may be too much for a single client.
- Renting a fixed bundle waste client budget.

- We extrapolate that clients will rent a basic bundle, and dynamically supplement it with resources in fine granularity.


CPU (GHz): 8.25

Core-GHz/Hour: USD 0.0225

Memory (GB): 15.34

GB/Hour: USD 0.0293

Trend: Service Level Agreements

Most cloud providers account for rigid availability only ("the machine is accessible").

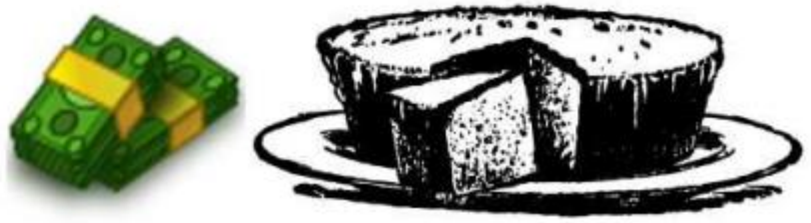
👎 / 👍

GoGrid and CloudSigma provide guarantees in terms of minimal actual delivered capacity (latency, packet loss and jitter).

Extrapolation: Service Level Agreements

We extrapolate that:

- Client pressure for efficiency will drive providers to supply levels of quality service: "For 90% of the time" or just "for 80% of the time".
- Low-QoS clients will be willing to pay less than high-QoS clients.



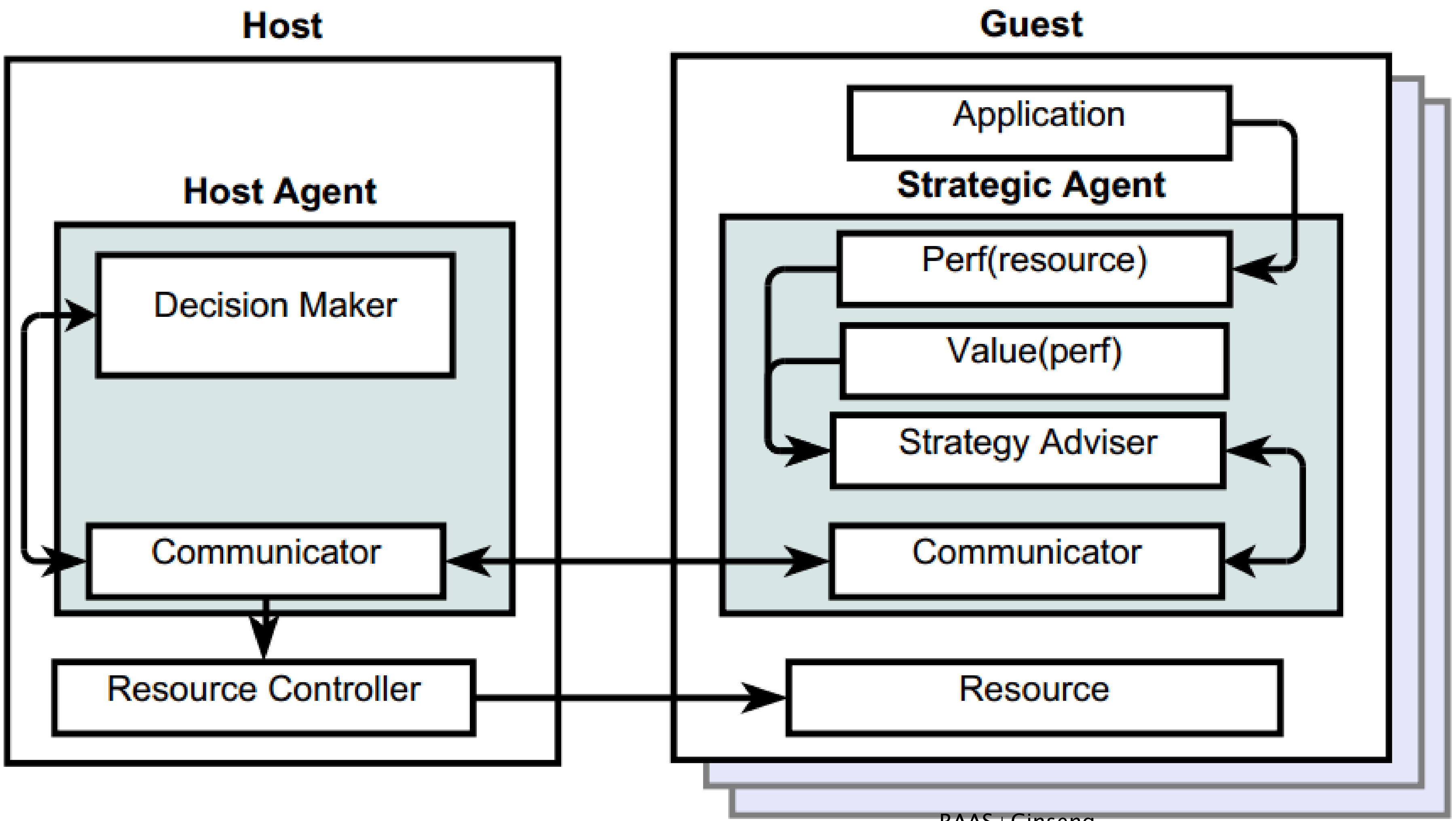
Economic Forces Acting on the Provider

- The provider must keep spare resources for rich, high-QoS clients.
- The provider can let poor, low-QoS clients use the spare resources, subject to availability.
- The provider must mix low QoS clients with high QoS clients.
  - "Robin Hood in reverse"

Economic Forces Leading to the RaaS Cloud

- Both clients and providers must continuously decide what to buy and when to buy it.
- The fine rent time granularity and bundle flexibility makes decision-making a core function.
- Both providers and clients will use economic agent software to handle decision making and economic interaction.

The RaaS Architecture



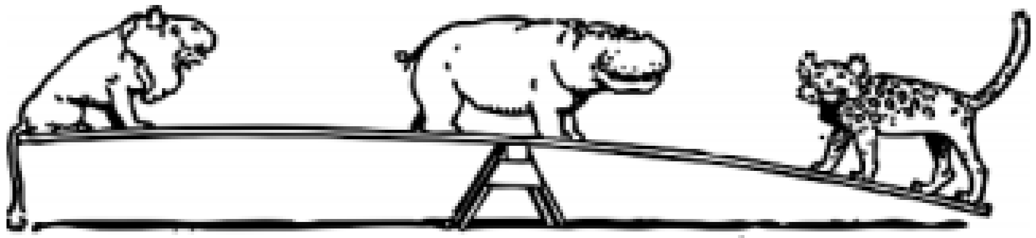
The Guest Agent

- Changes the desired amount of resources on a second-by-second basis.
- Negotiates
- Trades in the futures market.
- Sublets.

- Is not mandatory: dumb clients are still supported, with the same inefficiency of today's IaaS clouds.

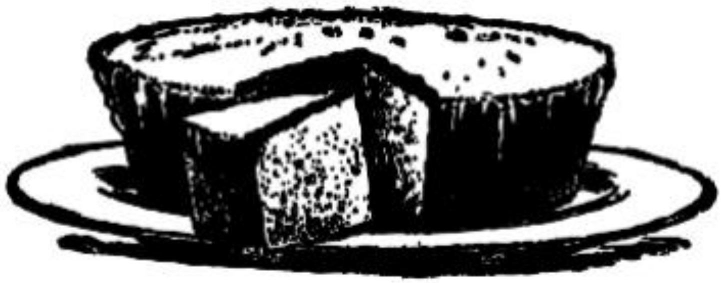
The Host Agent: Market Driven Resource Allocation

- Has a view of the global picture (total system resources, change predictions)
- Attempts to increase over-commitment when QoS is maintained (airlines analogy)
- Dictates economic mechanisms and protocols.
- Allocates resources according to agreements.
- Verifies that high-QoS clients are satisfied, possibly at the expense of low-QoS clients.

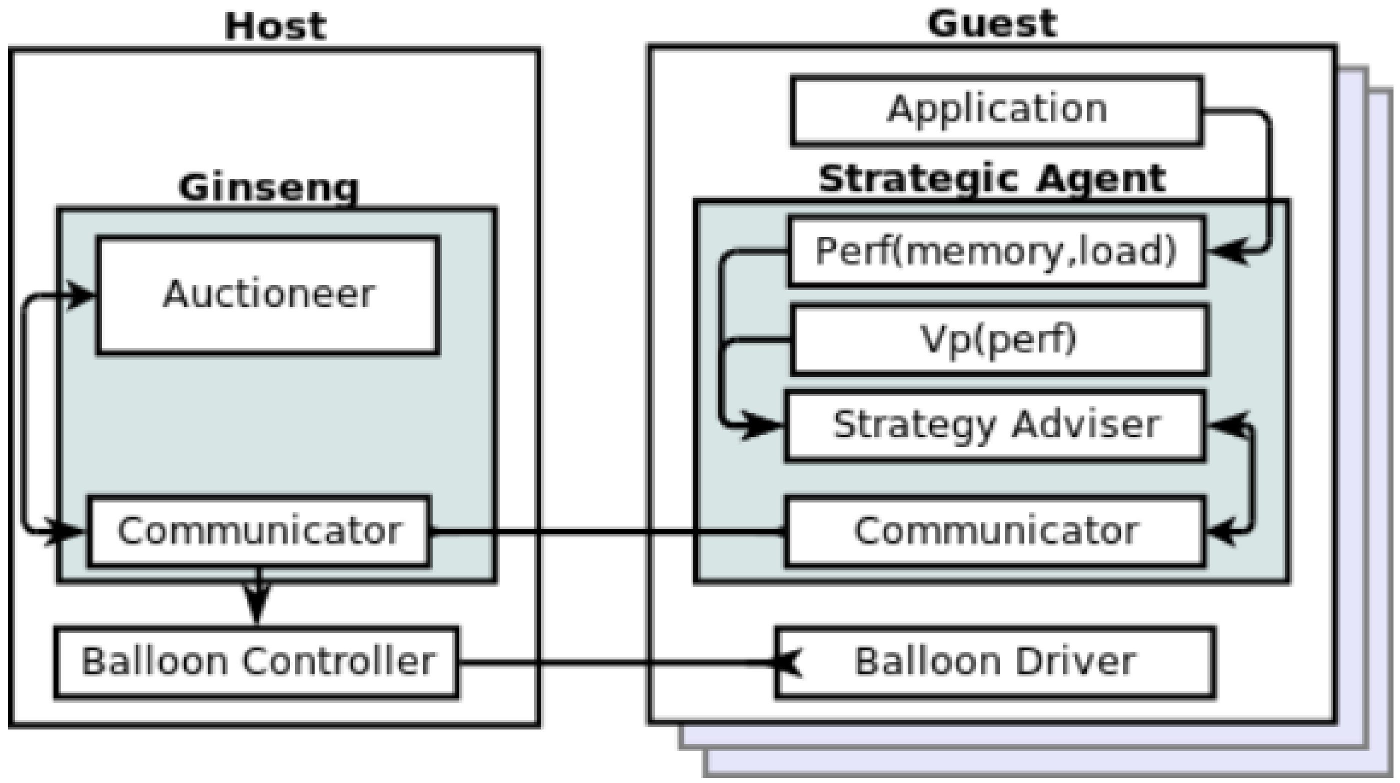


Summary – Resource Allocation

- Divide resources among selfish black-box guests?
- Give more to guests who would benefit more.
- How can the host compare guest benefits from using additional resources?
- Auction theory to the rescue ☺
- Lots of issues ☹
  - Partial information
  - Theory is limited, open problems
    - Does not cover real-life issues
      - repeated games, leftovers, learning, manipulation, spying
  - Theory involves highly-complex optimizations
  - Additional dimensions, e.g. split/merge decisions
  - Attacks, collusions, side channels, privacy issues
  - Many more…



Ginseng: Market-Driven Memory Allocation [VEE 2014]



Designing a Good Allocation Mechanism

- Vickrey-Clarke-Groves auctions
- Guest bids with a valuation of the good – how much it is worth, subjectively.
- Auctioneer finds the allocation that maximizes the social welfare SW.
- The auctioneer charges guests according to the exclusion-compensation principle: the difference between the SW when they participate to the SW when they do not.
- Prices are NOT uniform. Prices may drop to a minimal price (possibly zero) if there are extra resources

A VCG auction is **truthful**: guests bid their real types, no matter what other guests do.

Designing a Bandwidth Allocation Mechanism: PSP

- Lazar and Semret (1999): progressive second price auction (PSP).
- Guests bid with their required quantity and unit-price
  - Not with their full types
- Guests are sorted by bid price to be allocated with desired quantities.
- Guests are charged by the exclusion-compensation principle.

Multi-bid auctions

- Maille and Tuffin 2004 extended PSP to multi-bid auctions
  - PSP converges to the same SW
  - Truthfully (same $\mathcal{E}$-Nash equilibrium)

- Not really needed in dynamic systems requiring repeated auction rounds
- High computational overhead on auctioneer
- Guests need to know in advance full valuation function for full range
  - Harder to explore working points that are seemingly not optimal

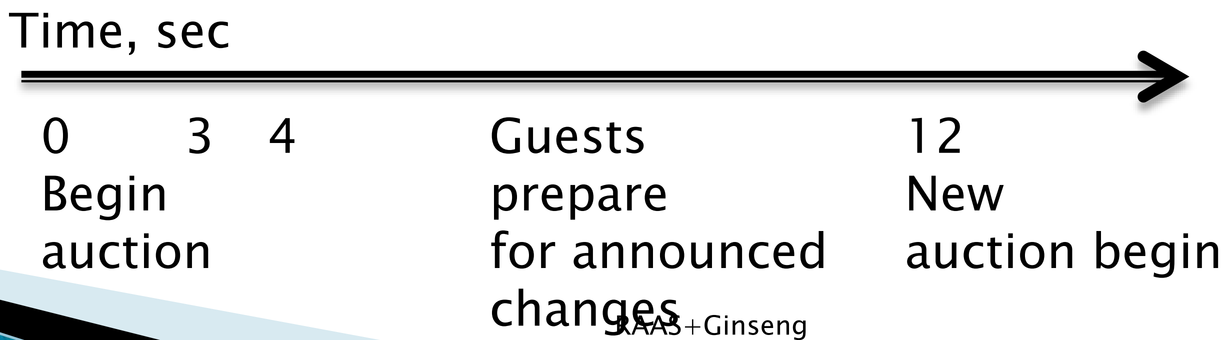
PSP is not suited for the Cloud

- **The PSP protocol requires guests to hear and analyze how other guests bid.**
- **Spoiler - our MPSP protocol is based on secret bids, so that spying on neighbor guests in an MPSP auction is limited.**

The MPSP Auction Protocol

The host sets up each guest with a base memory. Auction rounds repeat every 12 seconds:

- **The host announces the auctionable memory to guests (time = 0)**
- **The guests may bid for memory (time = 0 → 3)**
- **The host collects bids and decides on an allocation and payments (time = 3 → 4)**
- **The host announces the auction results (time = 4)**
- **The host makes allocation changes (time = 12)**

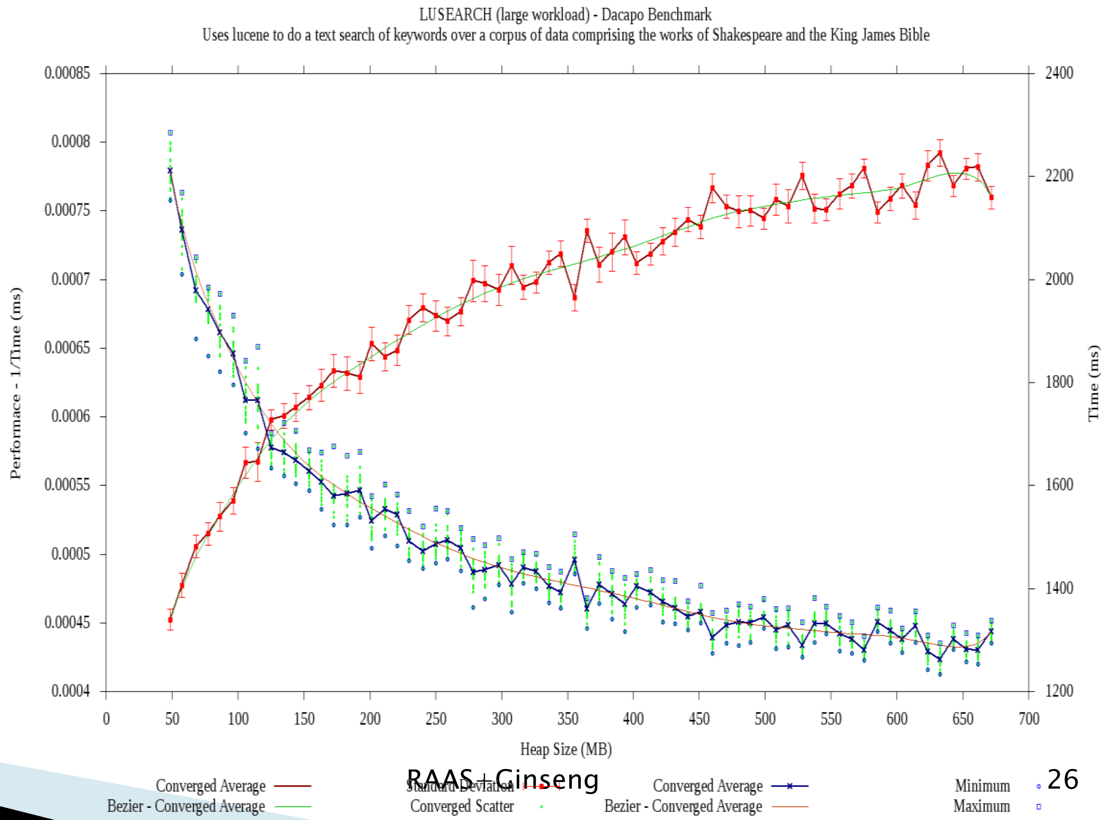


Elastic Memory Apps

- Out-of-the-box apps come with fixed cache size, heap size (e.g. JVM), data structure size
- APIs required to dynamic change app memory signature
- App should be able to measure performance as a function of memory
- Not always necessary

  to change the app.

  Can do with libraries.



Moving memory between guests in a virtualized system

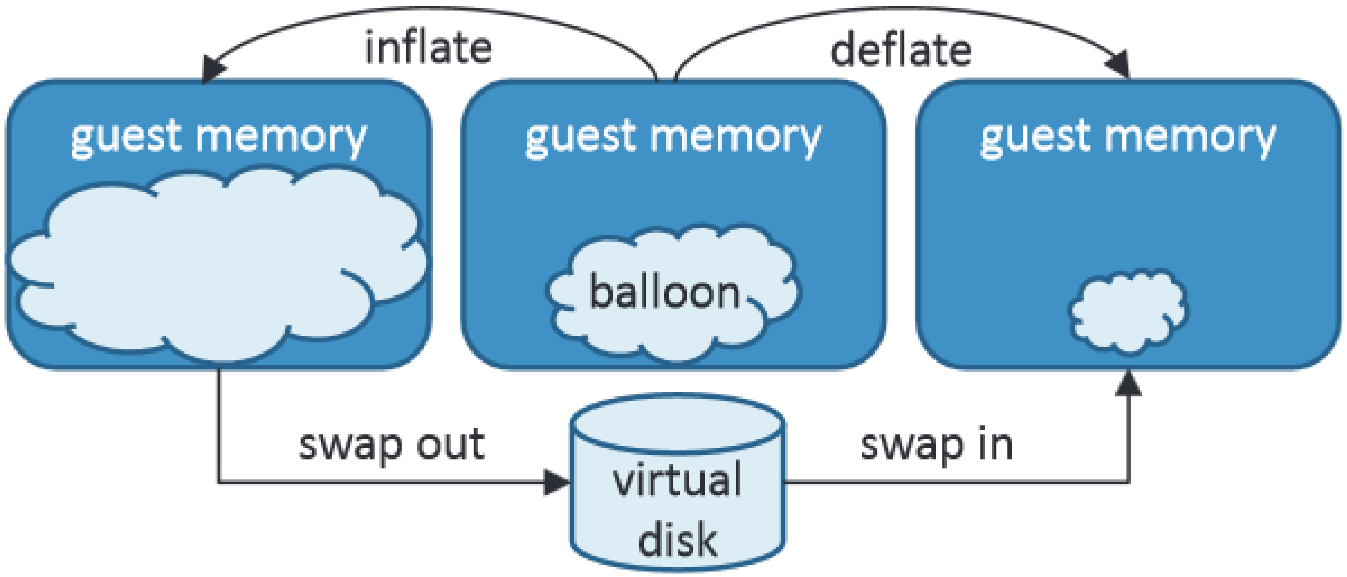


Figure 2. *Inflating the balloon increases memory pressure and prompts the guest to reclaim memory, typically by swapping out some of its pages to its virtual disk. Deflating relieves the memory pressure. (Reproduced from [82].)*

[82] Carl A. Waldspurger. Memory resource management in Vmware ESX server. In *USENIX Symposium on Operating Systems Design & Implementation (OSDI)*, volume 36, pages 181–194, 2002.

PSP is not suited for Memory Allocation

Memory is only beneficial if you use it long enough (e.g., allowing cache warmup)

- In case of a tie between guests, none of the PSP guests wins
  - Motivate bidders to break ties
  - Assumes full knowledge

- MPSP: Ties are broken in favor of the guest currently holding the memory
  - plus other ex-ante fair tie-breaking mechanisms so memory does not go unused when it is needed

PSP assumes Concavity and Monotonicity

- PSP assumes concave monotonically rising valuation functions, and a divisible good.
- Clients may sometimes value memory thus, in smart applications:

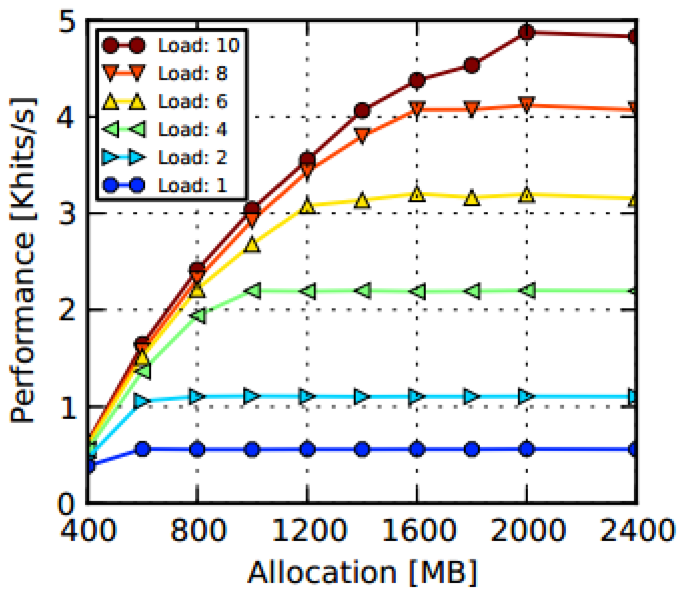


Figure: Elastic-memory memcached

PSP is not suited for Memory Allocation

- PSP assumes concave monotonically-rising valuation functions, and a divisible good.
- But legacy applications usually have a step function performance graph, which is not concave:

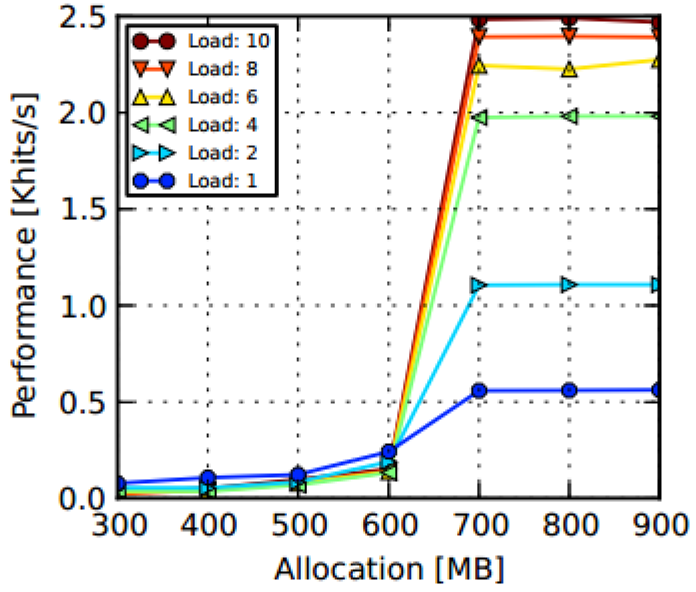


Figure: Off-the-Shelf memcached

PSP is not suited for Memory Allocation

- PSP assumes concave monotonically rising valuation functions, and a divisible good.
- Sometimes, especially when performance is measured on-line, the performance is not even monotonically rising

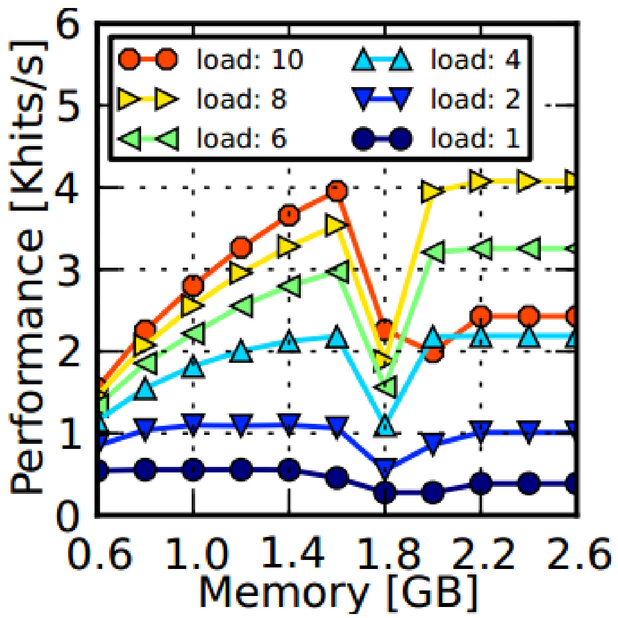


Figure: Elastic Memory memcached, in a non-optimal environment

TOMCAT, DaCappo

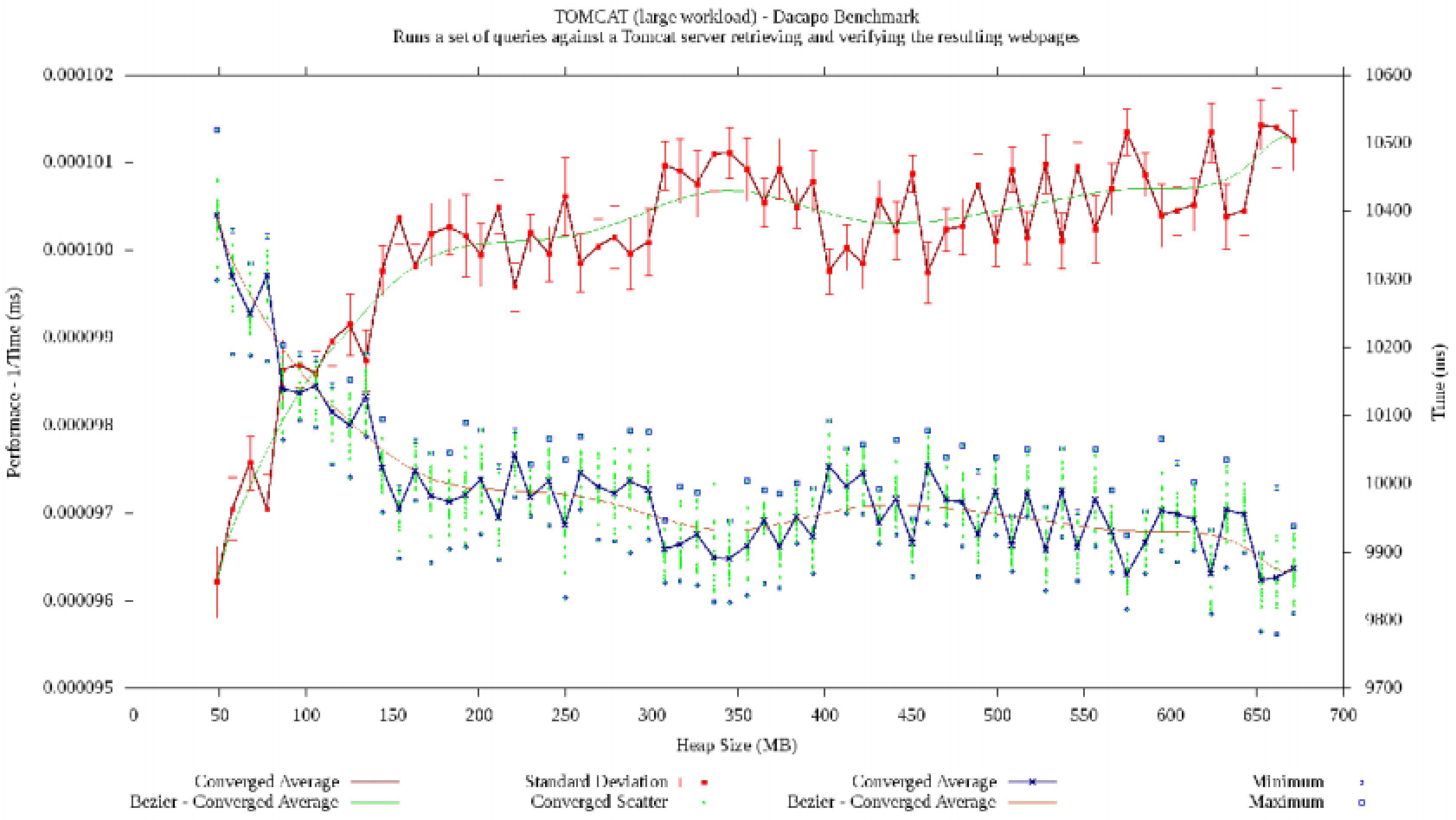
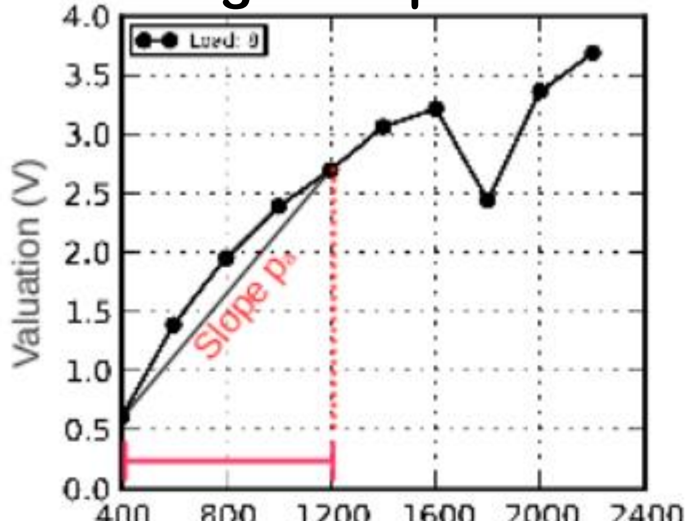
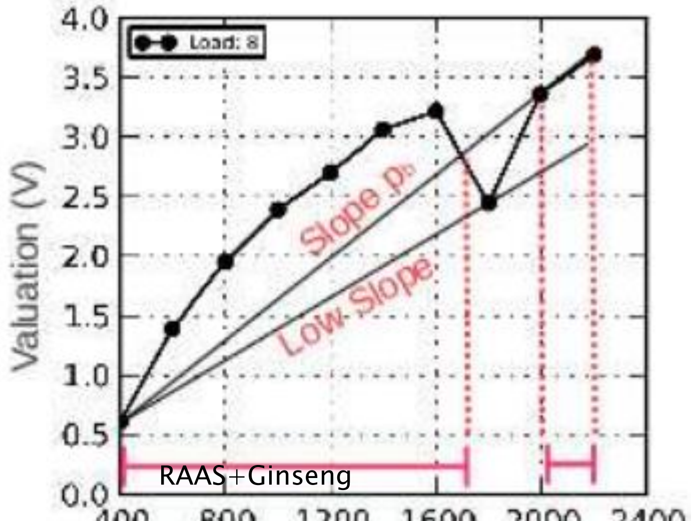


Figure: TOMCAT benchmark, from the DaCappo suite

The MPSP Protocol Supports Memory Ranges

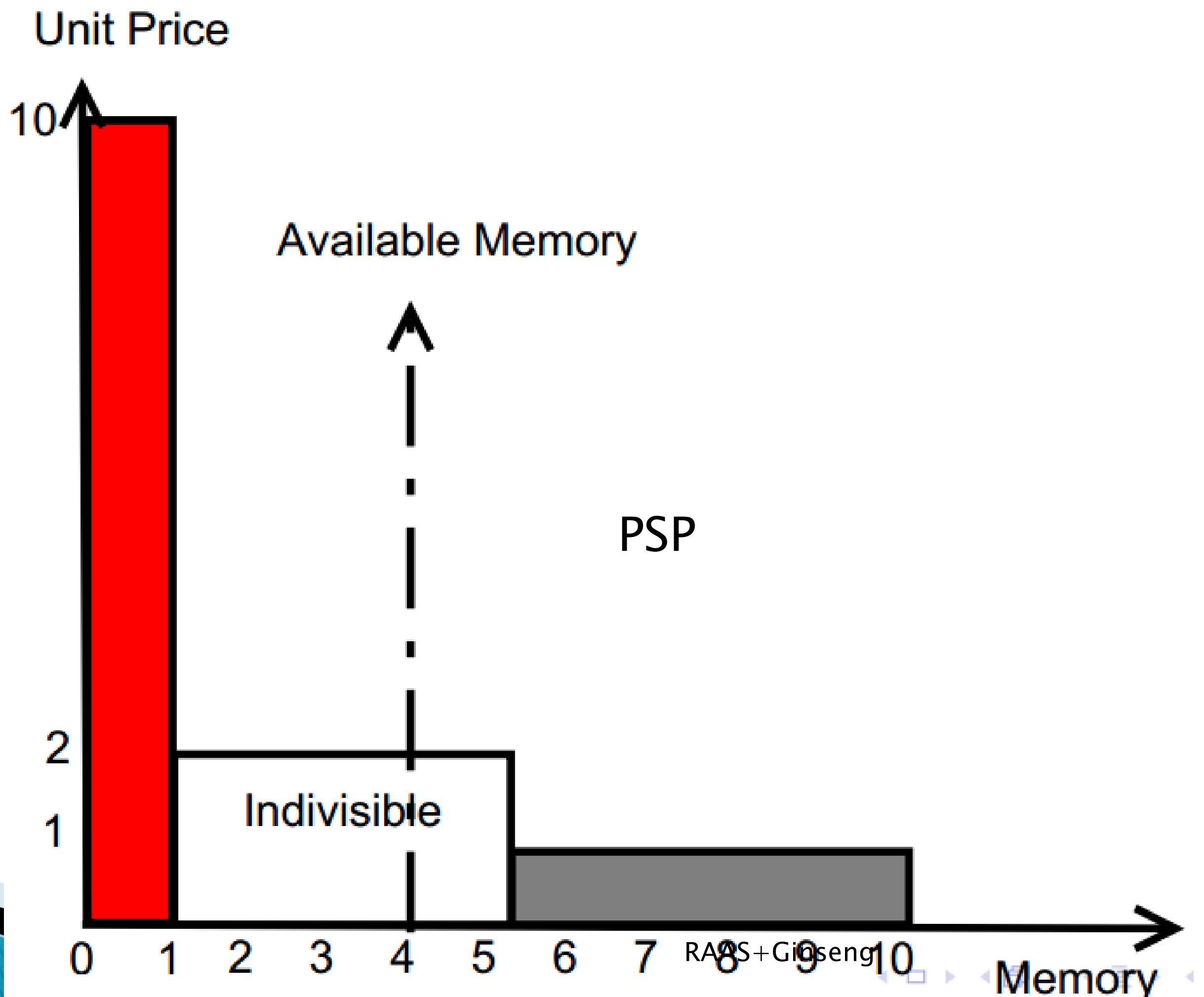
- The memory progressive second price auction (MPSP) supports non-concave, non-monotonically rising functions:
- A bid is composed of a single unit-price and multiple memory ranges *(q, r, p).*
- Bidding a price which is the slope of the valuation graph is almost always the best strategy.
  - *p* = average unit price





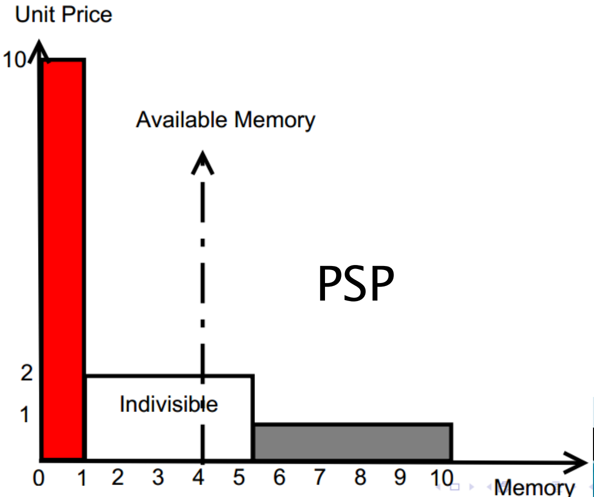
The MPSP Allocation Choice

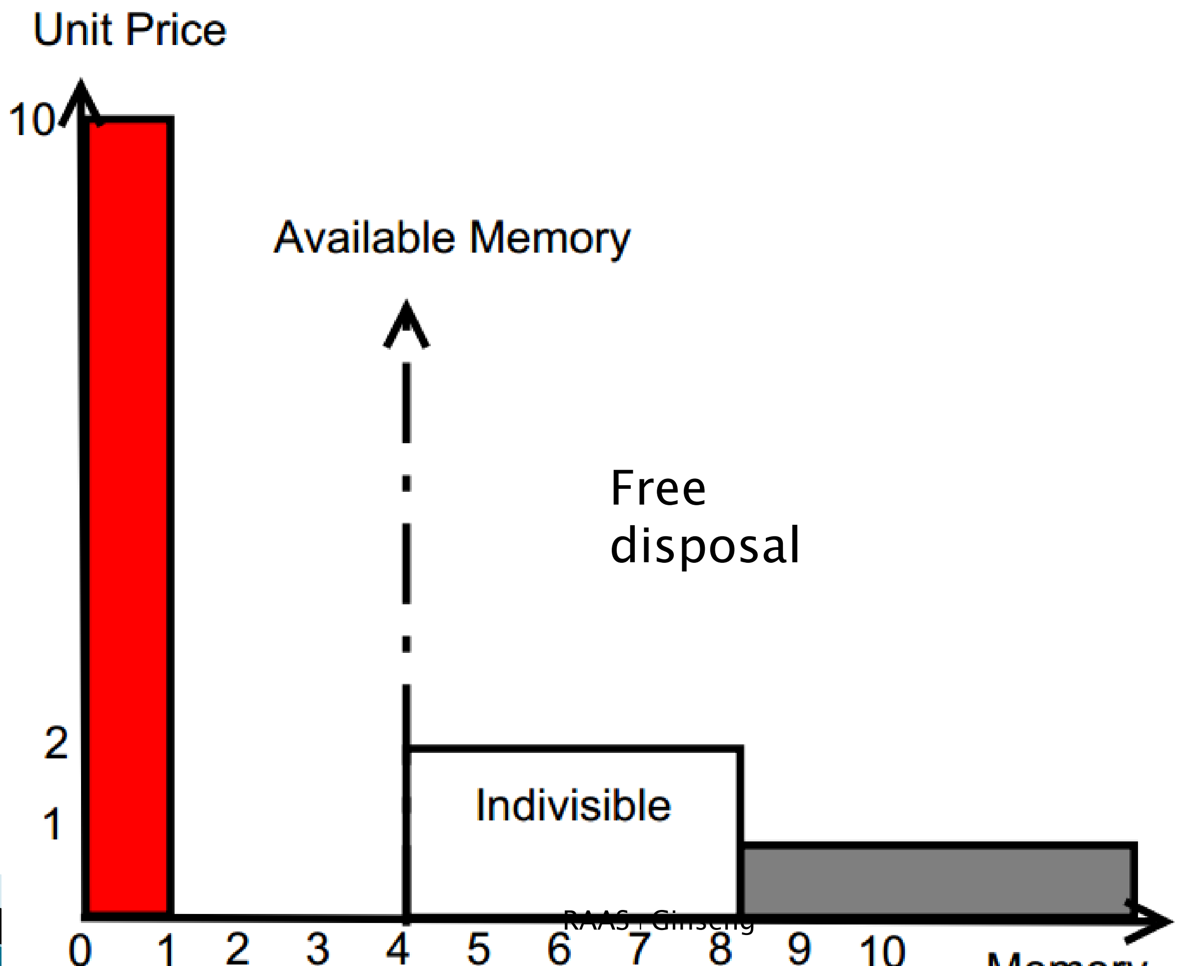
The PSP allocation prefers higher unit prices, assuming there are no forbidden ranges:



The MPSP Allocation Choice

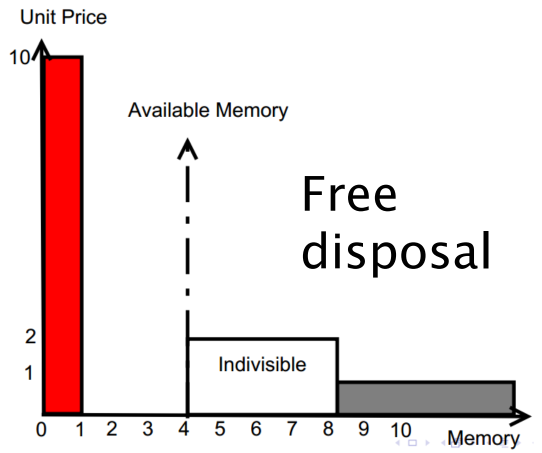
Free disposal of auction results supports forbidden ranges, but is inefficient:

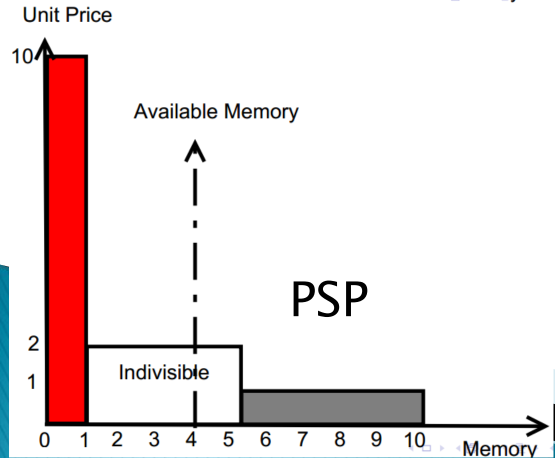


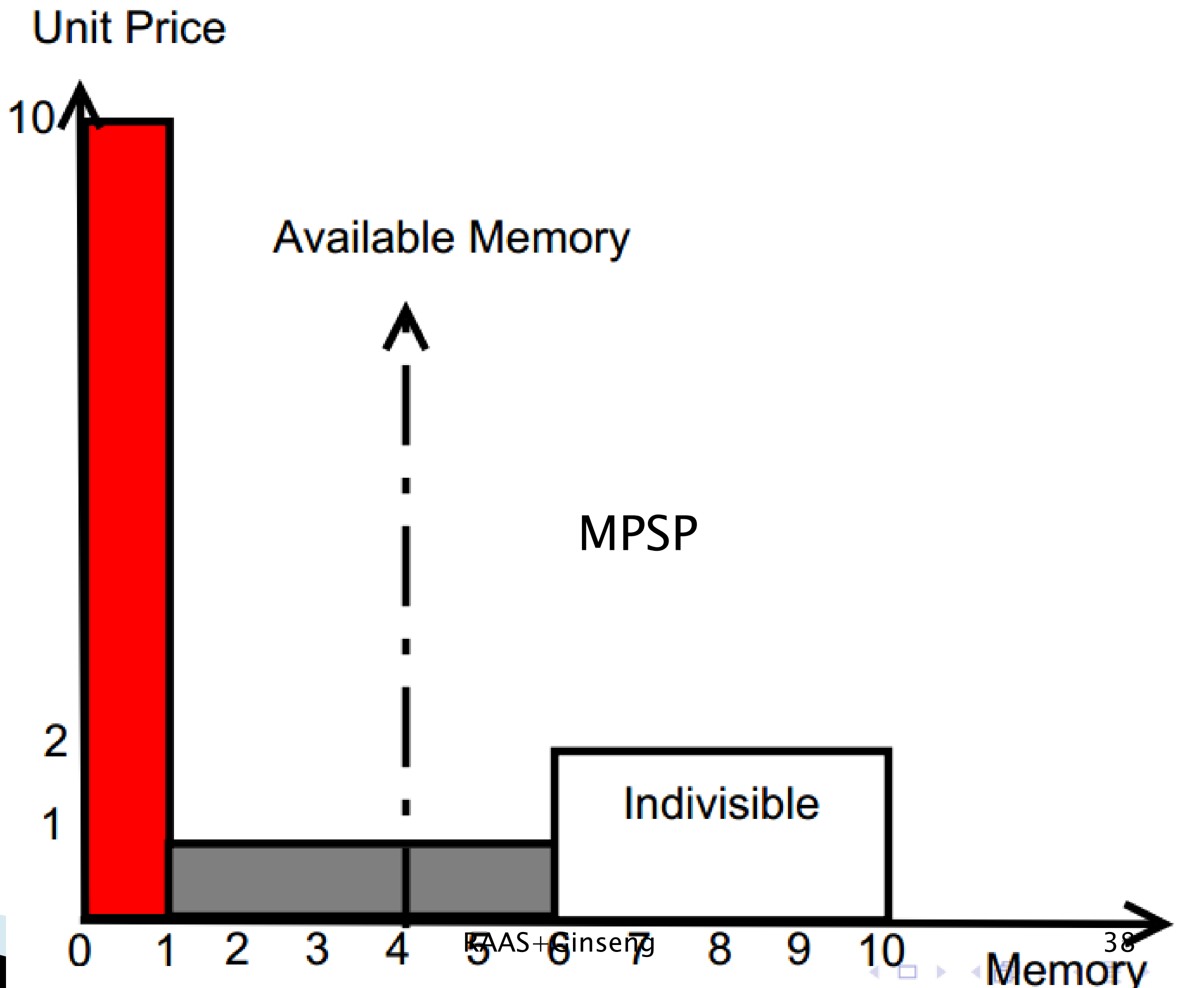


The MPSP Allocation Choice

The MPSP allocation finds the allocation with the highest social welfare under the forbidden ranges constraints:







PSP is not suited for Memory Allocation

- PSP assumes no overhead in transfer of resources
  - When finding best SW allocation
  - Makes sense for bandwidth
  - But not for memory

https://www.usenix.org/legacy/event/osdi02/tech/full_papers/waldspurger/waldspurger_html/node6.html

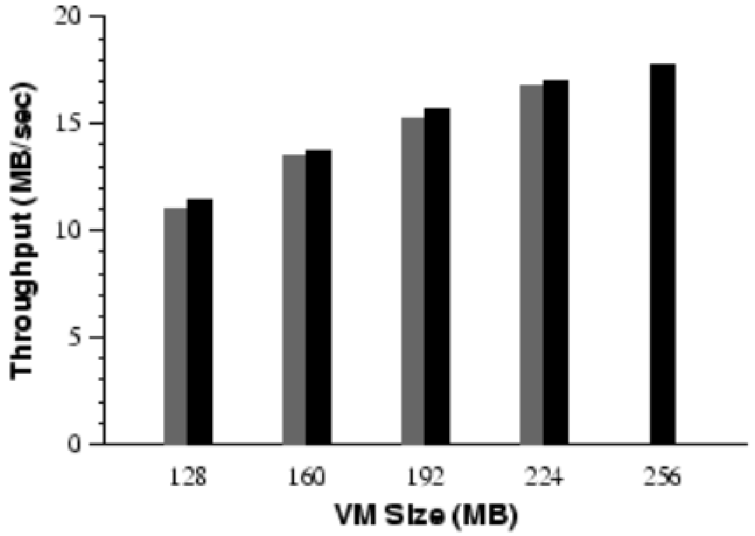


Figure 2: Balloon Performance. Throughput of single Linux VM running dbench with 40 clients. The black bars plot the performance when the VM is configured with main memory sizes ranging from 128 MB to 256 MB. The gray bars plot the performance of the same VM configured with 256 MB, ballooned down to the specified size.

Affine Maximizers

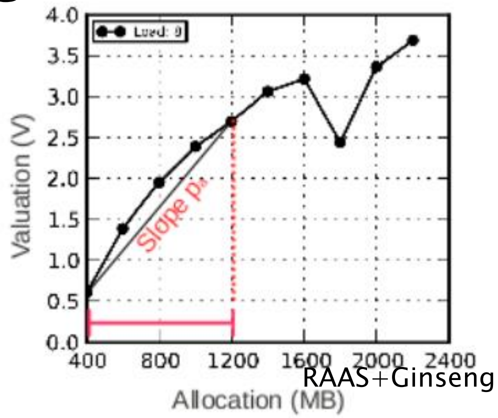
- A social choice function f assigning allocation a is called **affine maximizer if** $f(v_1,\ldots,v_n) \in \arg\max_a (c_a + \sum_i w_i v_i(a))$

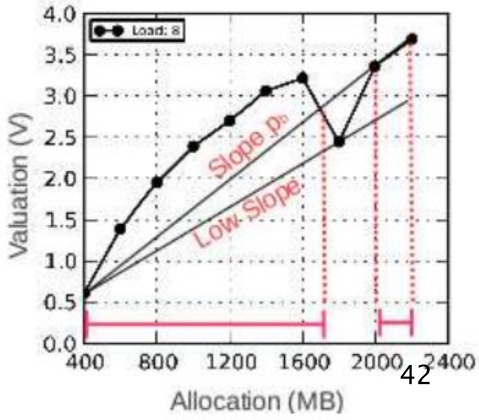
- Let f be affine maximizer. If the payment for the good is computed using the exclusion-compensation principle, then the mechanism $(f, p_1,\ldots, p_2)$ is **incentive compatible**.

- **Robert's theorem**: there are no incentive compatible mechanisms except for those based on affine maximizers.

Guest finds a bid *b=(q,r,p)*

- Find estimated payment for every *q* by online-learning previous results
  - All previously received allocations *a=(p',q')*
  - Closer in value *q'* provides more information
  - Closer in time allocation *a* provides more information
  - Reserved price *pmin* is a lower-bound
  - A recent *pmax-rejected* is an upper-bound
    - By exclusion-compensation principle
- Valuation *p* is known for every *q*
  - For forbidden ranges a lower-bound is known
- Get *b=(q,r,p)* by scanning for *q* obtaining highest utility for guest

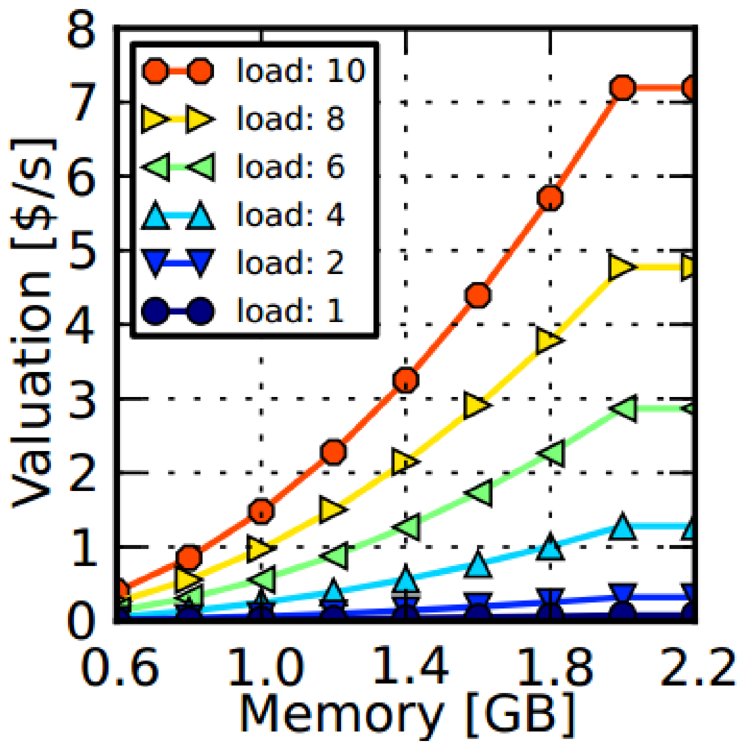




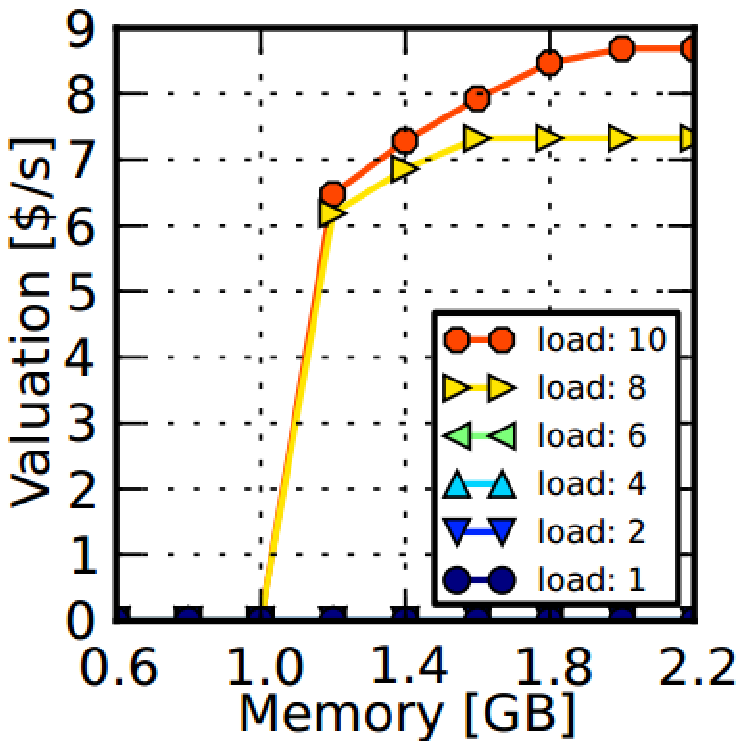
Summary – Positive results

- MPSP maximizes the SW for every guest
  - Even for non-concave non-monotonically rising valuations
  - By solving optimization problem with affine maximizer, and
  - Inspecting recursively all 0-1 allocations of forbidden ranges
- Bidding the true valuation of a memory quantity is the best course of action for the guest when:
  - It asks for a specific quantity (not a range), or
  - The valuation function is concave, monotonically rising, or
  - The system is at a steady state.

Experimental Evaluation: Non-Concave Valuation Functions



(c) ***MemoryConsumer (square of performance)***



(d) ***Dynamic Memcached (partially linear)***

Experimental Evaluation: Dynamic-Memory Benchmark

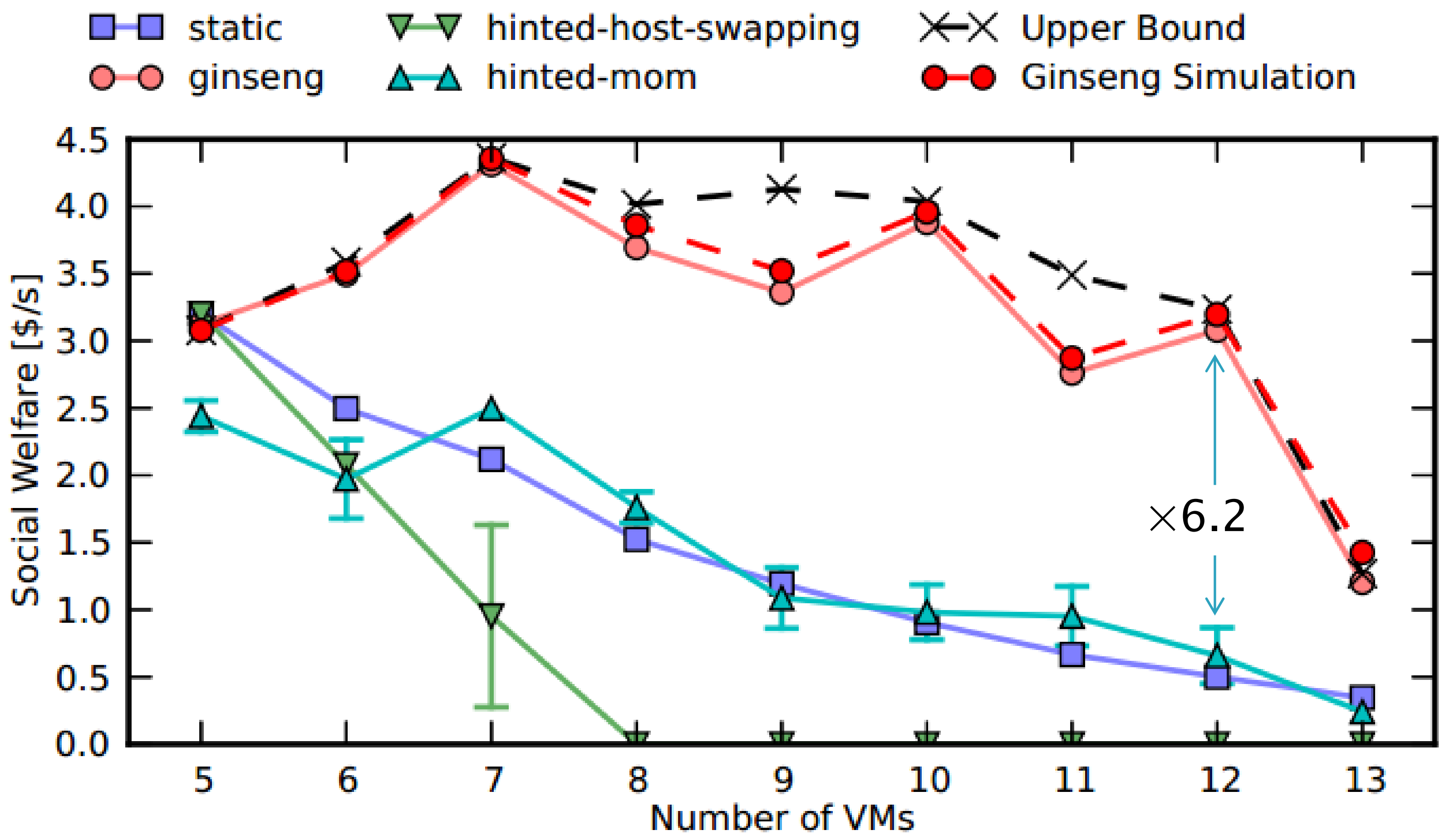


Figure: MemoryConsumer, valuation is square of performance

Experimental Evaluation: Dynamic-Memory Memcached

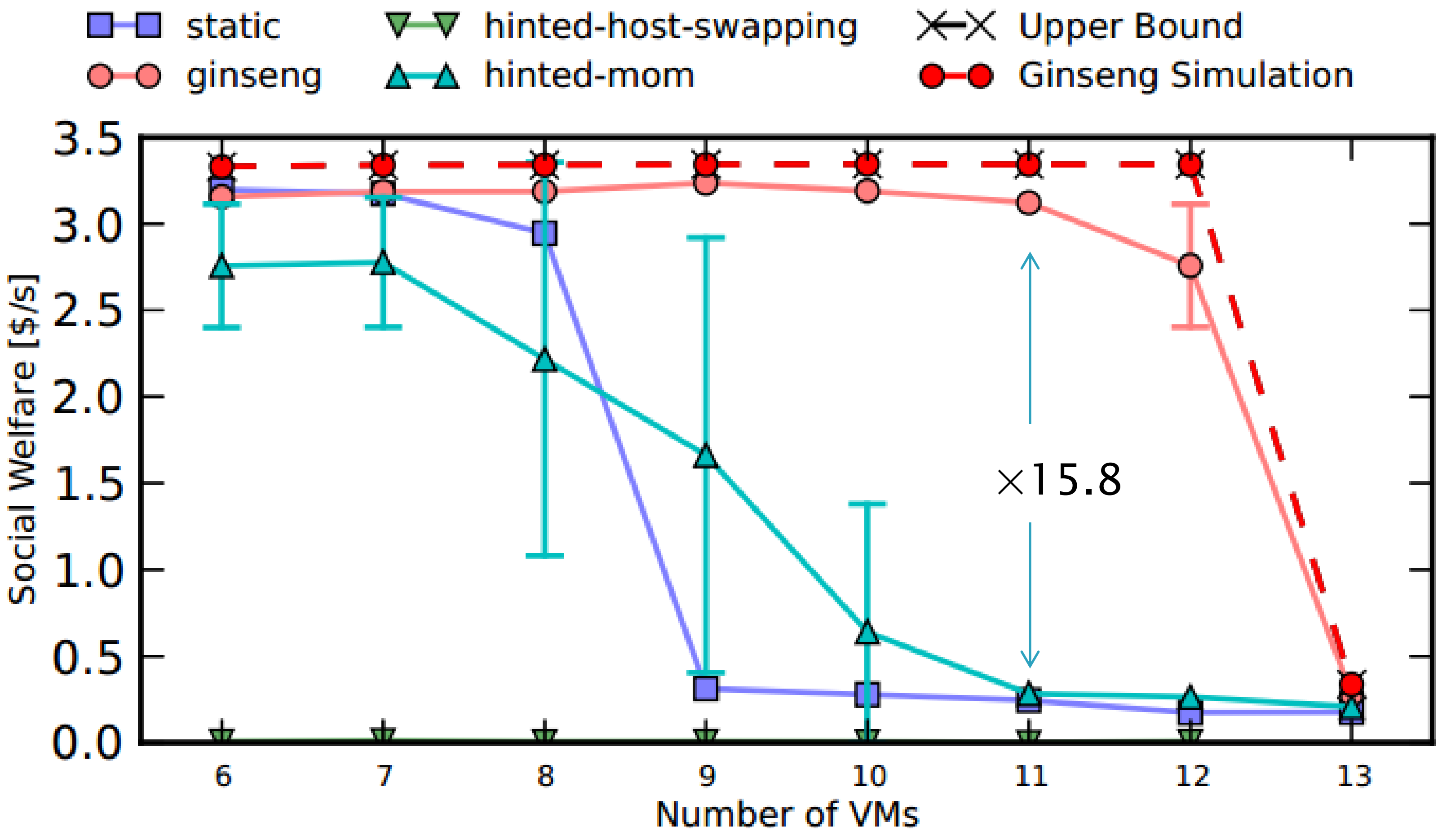


Figure: Memcached, first guest valuation is piecewise linear

Conclusion

- The Resource-as-a-Service cloud is the future cloud model.
- Ginseng is an efficient prototype implementation of RaaS for memory. It improves the social welfare by ×6.2 -×15.8.

- Future work
  - Full multi-resource RaaS machine: the RaaS software stack
  - Allocation and migration algorithms
  - Better dynamic game-theoretic mechanisms
  - Security and Privacy

Questions?

Contact: assaf@cs.technion.ac.il

Thank You!

- ***Ginseng: Market-Driven Memory Allocation.*** **Orna Agmon Ben-Yehuda, Eyal Posener, Muli Ben-Yehuda, Assaf Schuster, Ahuva Mu'alem. VEE 2014.**
- ***The rise of RaaS: the Resource as a Service Cloud*****. Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, Dan Tsafrir. Communications of the ACM. Forthcoming, July 2014.**
- ***The Resource-as-a-Service (RaaS) Cloud*****. Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. HotCloud, June 2012.**

End – do not go beyond this slide!!

Experimental Evaluation: Performance is comparable

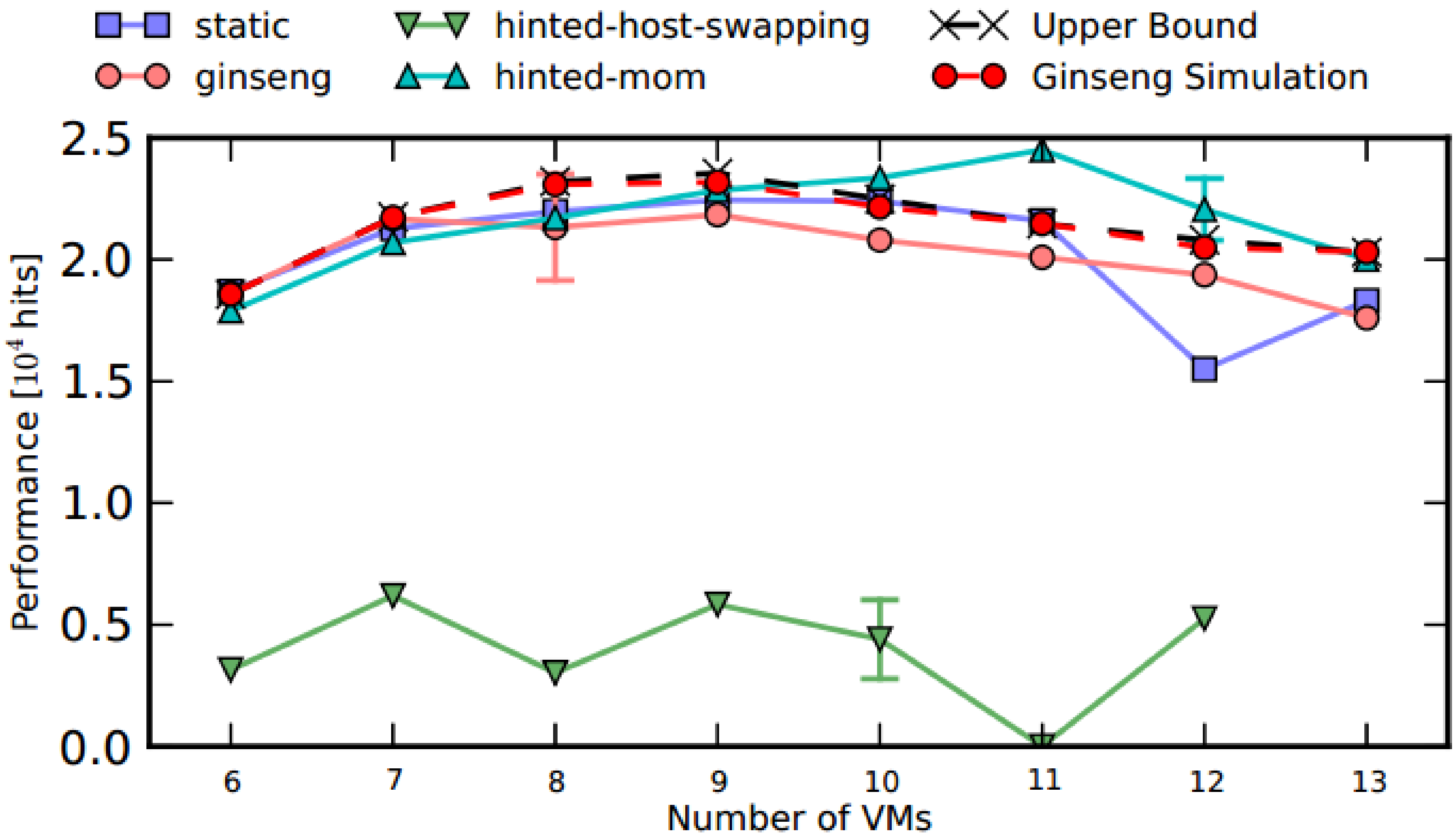


Figure: Memcached, first guest valuation is piecewise linear

Offline profiling is good enough

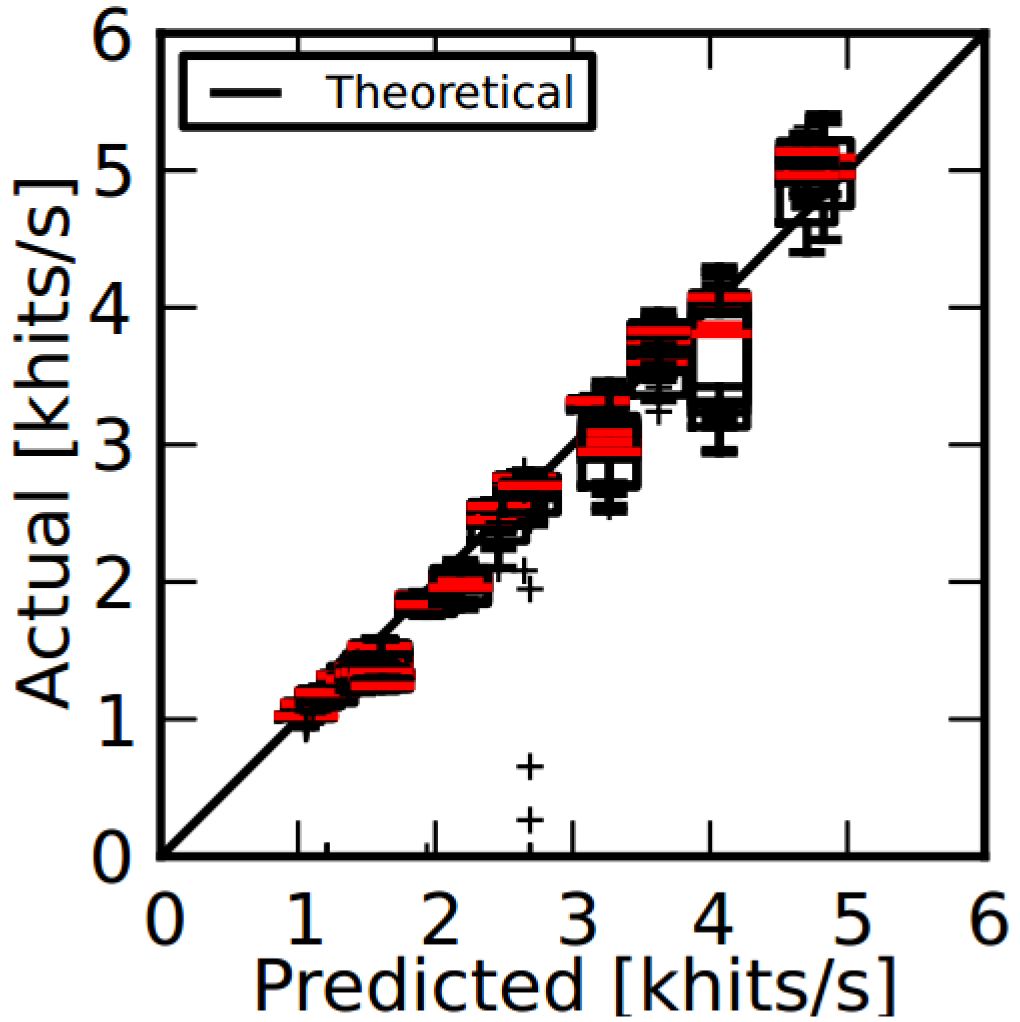
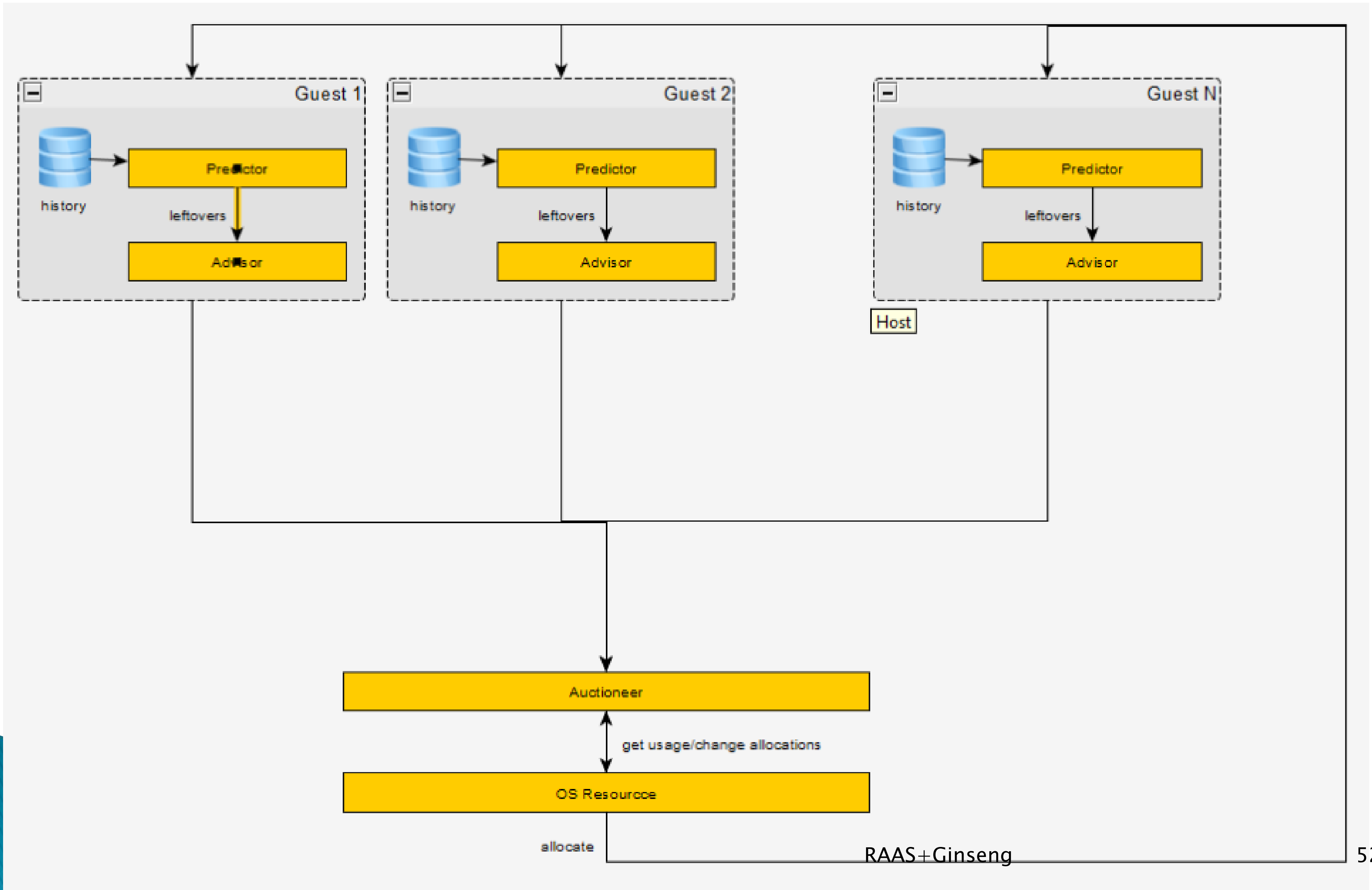


Figure: Memcached

Bandwidth allocation

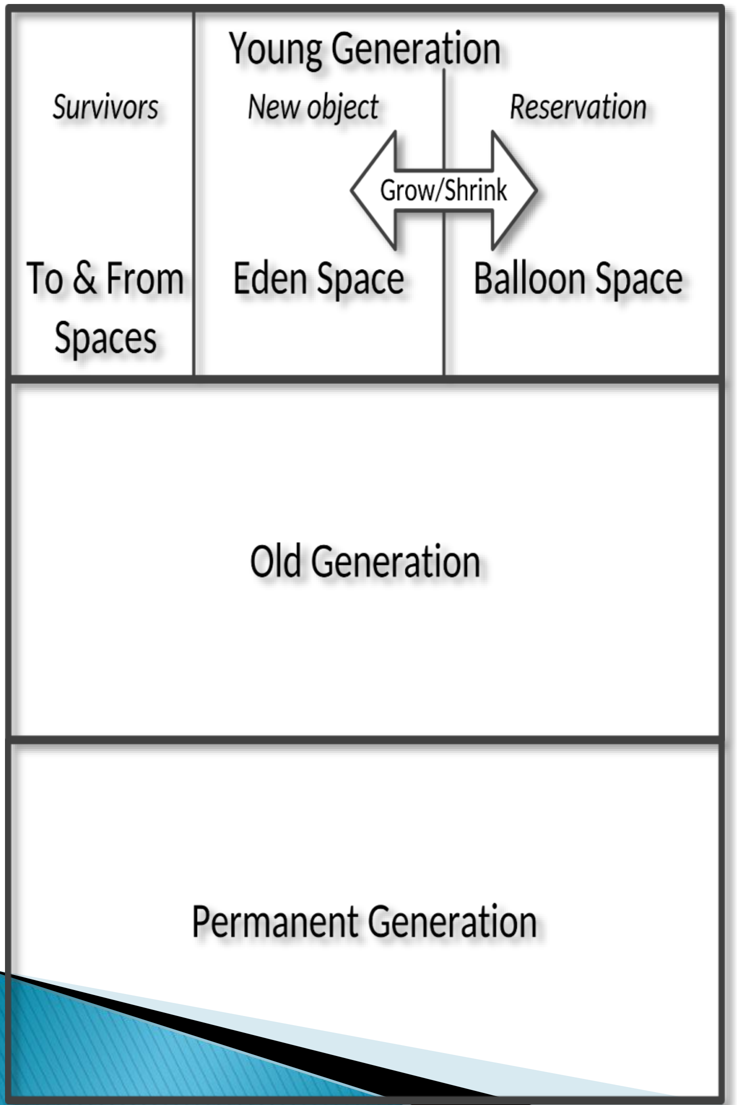


Avrora, Batik, Fop, Luindex and SUNFLOW

- The above benchmarks were too short and were hardly affected by changes in the JVM heap size.
- Sunflow crushed multiple times on our machine.

Tudor Salomie's Java Balloon

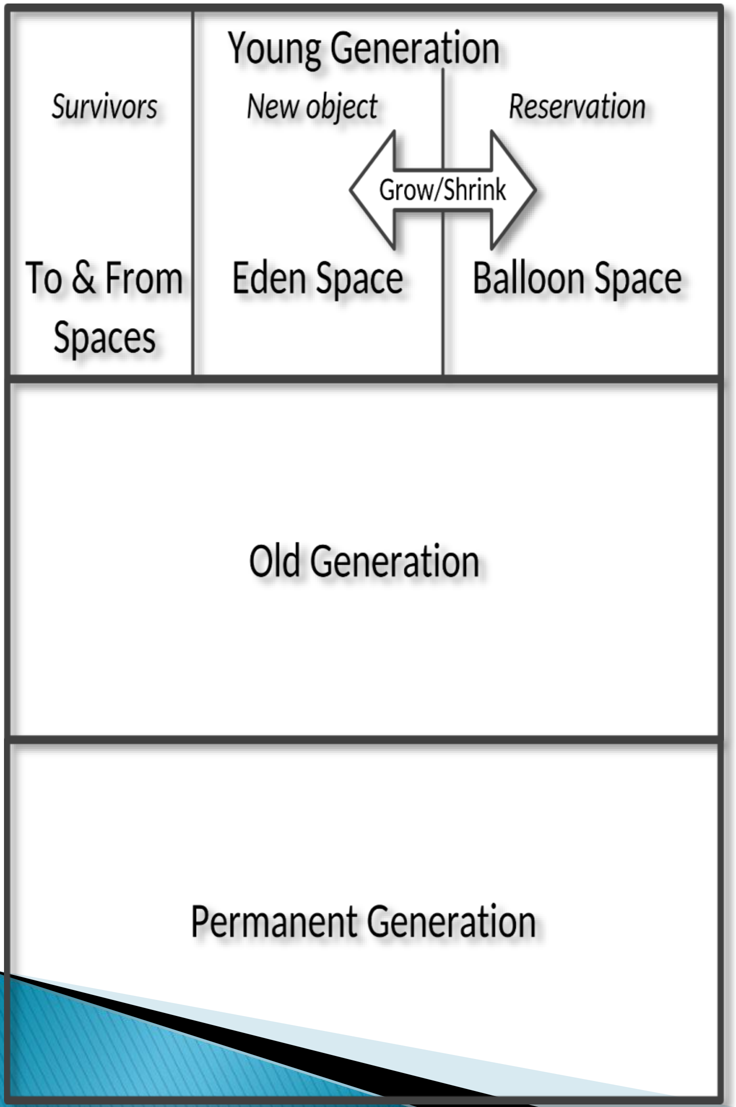
Tudor Salomie's Java Balloon



Parallel Scavenge Garbage Collector

- The **permanent generation** holds class metadata.
- The **young generation** holds recently-created objects, and those that have survived a few garbage collections.
- Objects that survive longer are moved to the **old generation.**
- The **young generation** is also split into **Eden** and **survivor** spaces.
- Spaces are bounded by start and end addresses, and all allocated memory sits between the start and top address, which is incremented as objects are allocated.
- Collection compacts space between the start and top addresses, removing holes and moving objects to other spaces.

Tudor Salomie's Java Balloon



Balloon Space

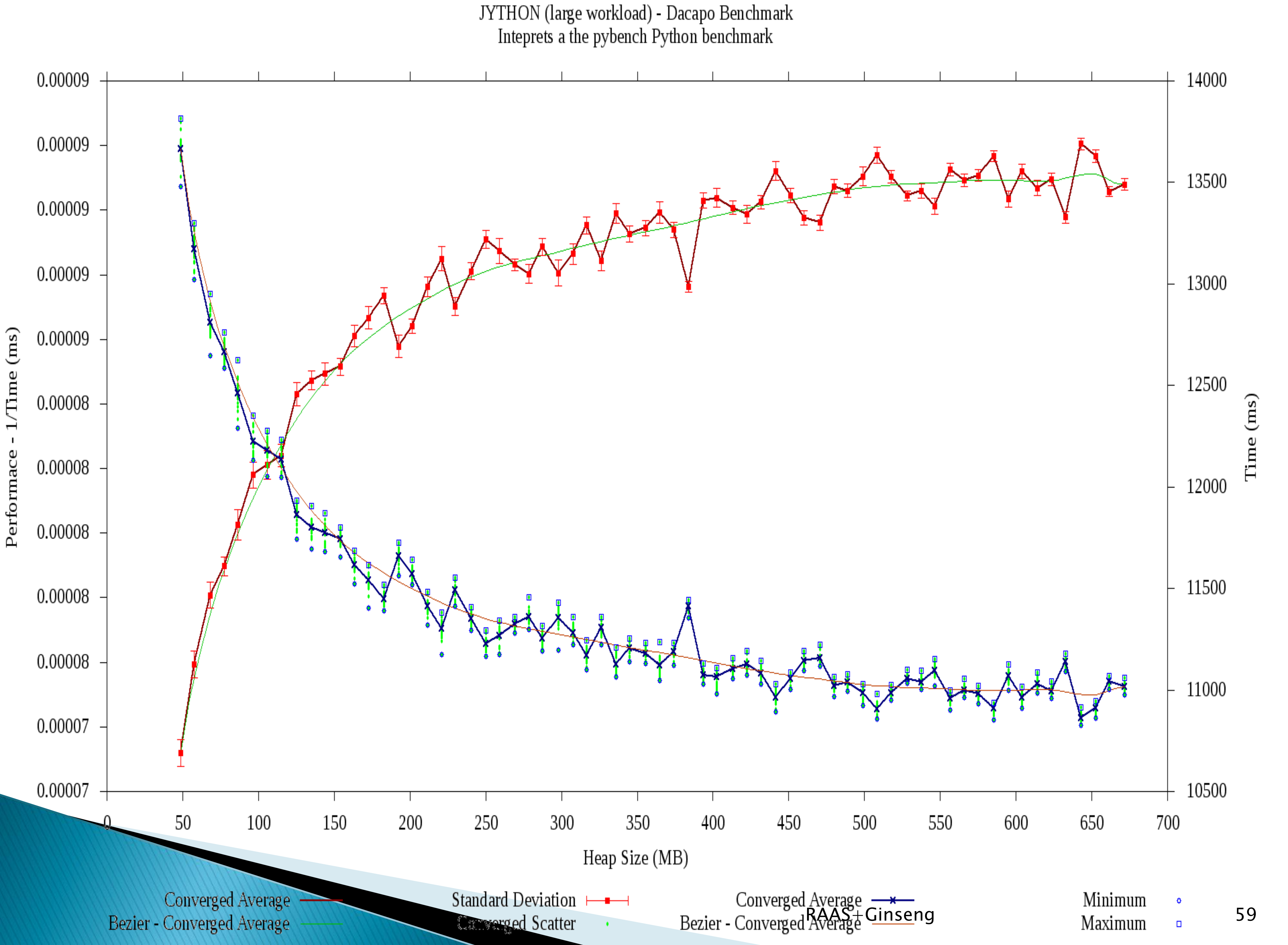
- The balloon space in the young generation can grow and increase the pressure on the Eden space when necessary, or contract and reduce the pressure.
- In order to resize the spaces composing the heap, we need to compact them and prevent any other operations during ballooning.
- For this reason, the ballooning operation is performed at the same time as a full garbage collection.
- Before returning from a full collection, we perform all outstanding ballooning operations.
- This means that the cost for ballooning operations in the JVM is influenced by the time needed to perform a garbage collection.

Dacapo Benchmark

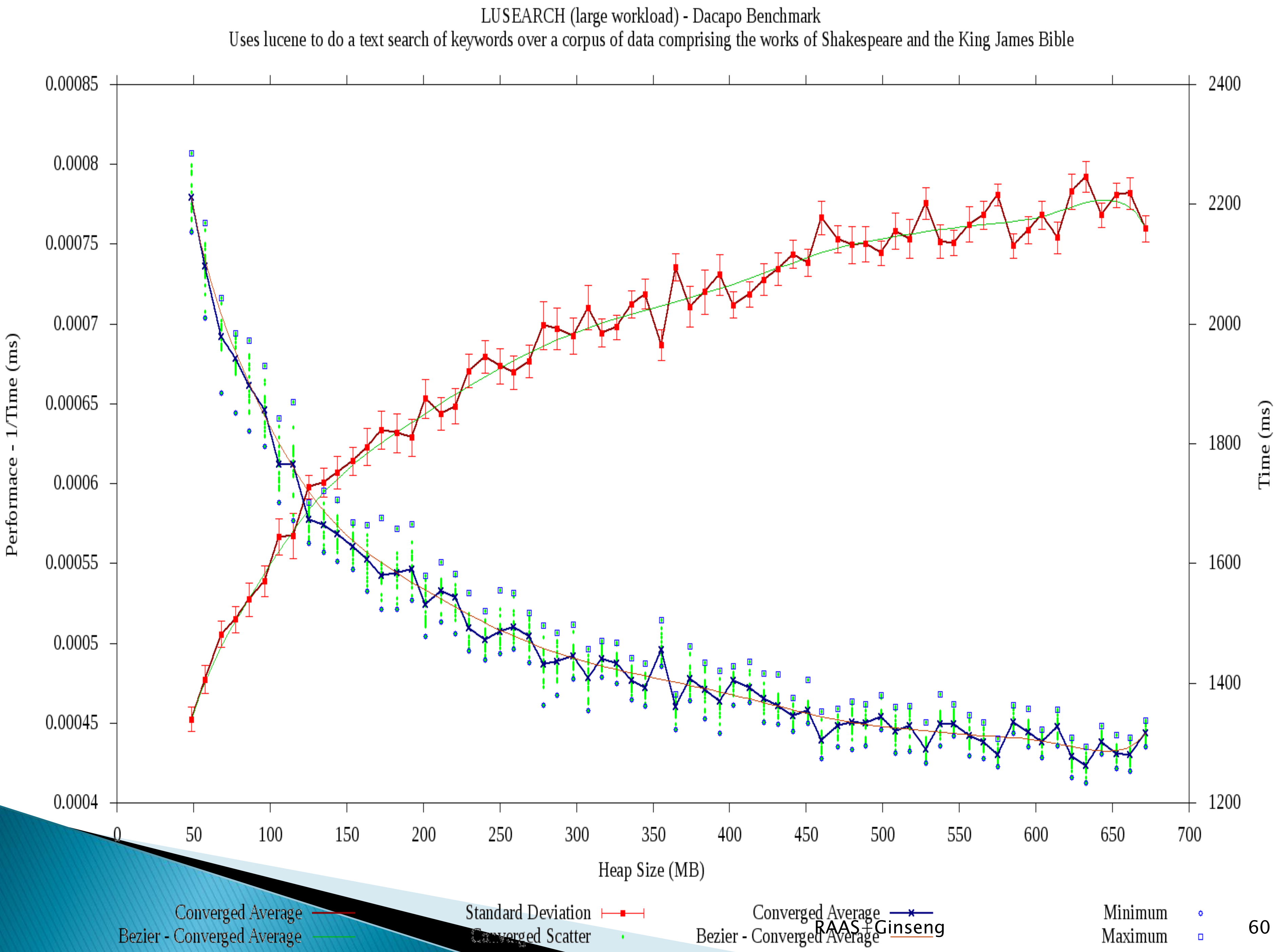
Performance graphs for different JVM heap sizes

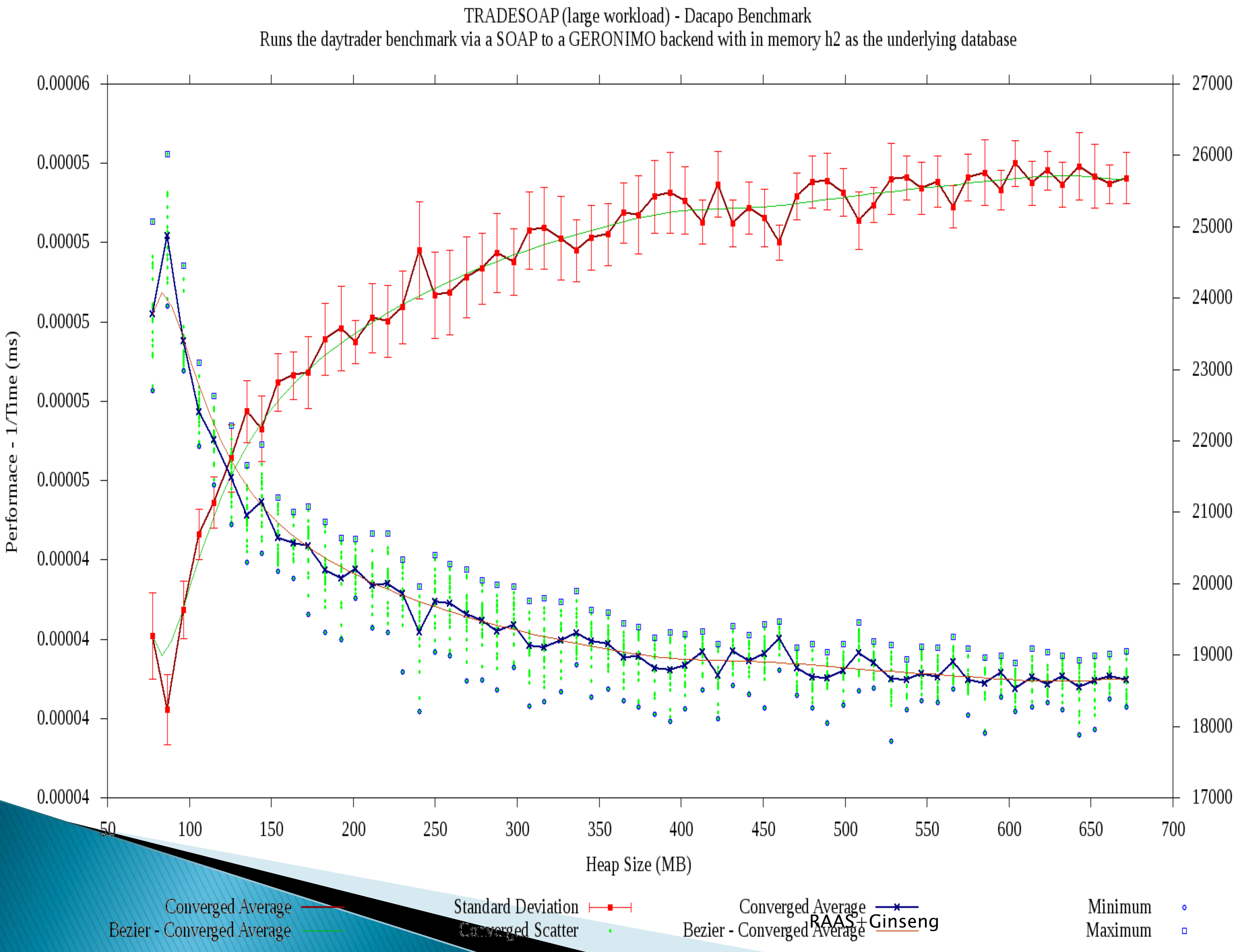
The DaCapo-9.12-bach benchmark suite, released in 2009, consists of the following benchmarks

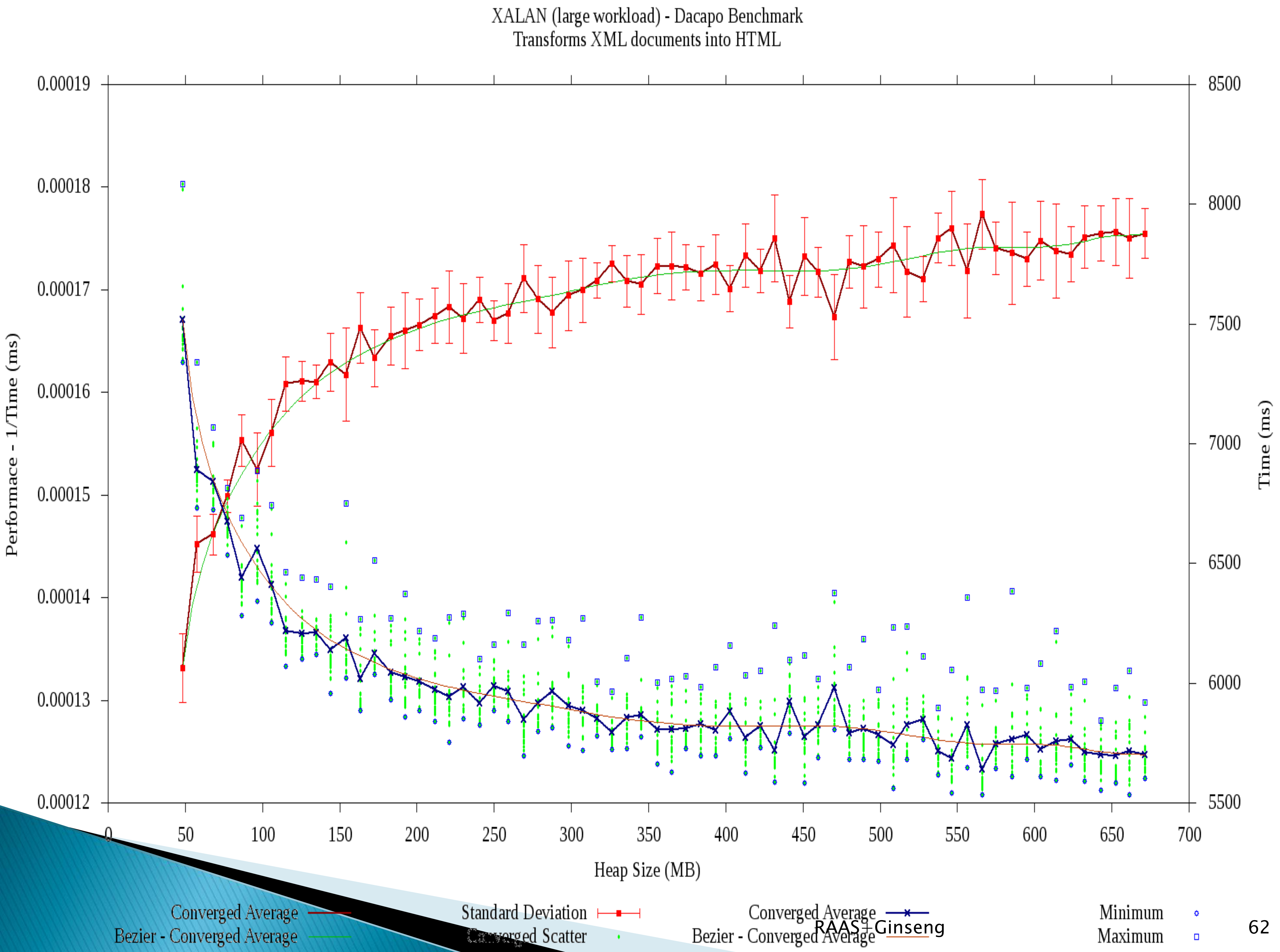
- **Avrora**: simulates a number of programs run on a grid of AVR microcontrollers
- **Batik**: produces a number of Scalable Vector Graphics (SVG) images based on the unit tests in Apache Batik
- **Eclipse**: executes some of the (non-gui) jdt performance tests for the Eclipse IDE
- **Fop**: takes an XSL-FO file, parses it and formats it, generating a PDF file.
- **H2**: executes a JDBCbench-like in-memory benchmark, executing a number of transactions against a model of a banking application, replacing the hsqldb benchmark
- **Jython**: inteprets a the pybench Python benchmark
- **Luindex**: Uses lucene to indexes a set of documents; the works of Shakespeare and the King James Bible
- **Lusearch**: Uses lucene to do a text search of keywords over a corpus of data comprising the works of Shakespeare and the King James Bible
- **Pmd**: analyzes a set of Java classes for a range of source code problems
- **Sunflow**: renders a set of images using ray tracing
- **Tomcat**: runs a set of queries against a Tomcat server retrieving and verifying the resulting webpages
- **Tradebeans**: runs the daytrader benchmark via a Java Beans to a GERONIMO backend with an in memory h2 as the underlying database
- **Tradesoap**: runs the daytrader benchmark via a SOAP to a GERONIMO backend with in memory h2 as the underlying database
- **Xalan**: transforms XML documents into HTML



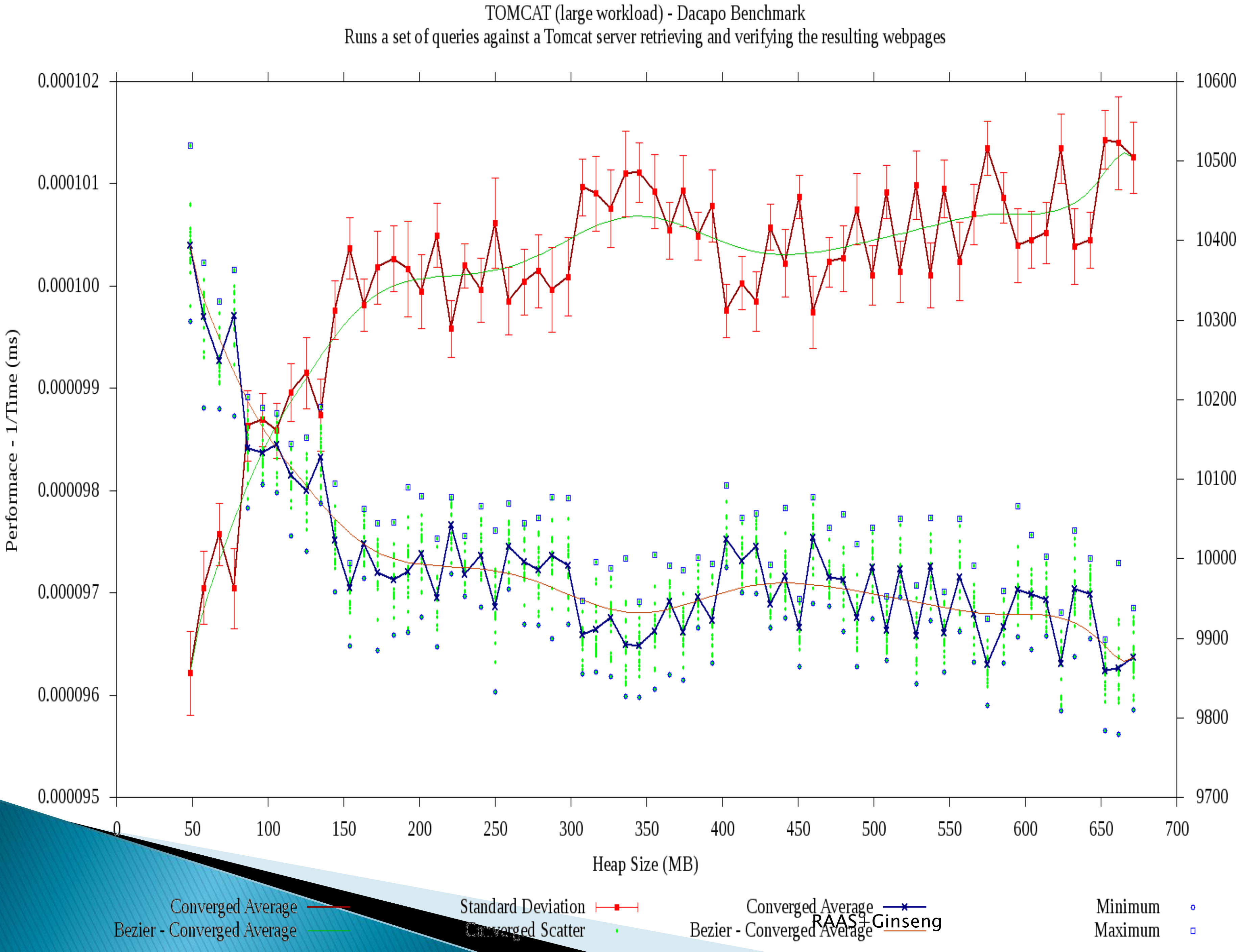
JYTHON (large workload) - Dacapo Benchmark
Inteprets a the pybench Python benchmark
Performace - 1/Time (ms)
Time (ms)
Heap Size (MB)
0.00009
0.00008
0.00007
14000
13500
13000
12500
12000
11500
11000
10500
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
Converged Average
Bezier - Converged Average
Standard Deviation
Converged Scatter
Converged Average
Bezier - Converged Average
Minimum
Maximum

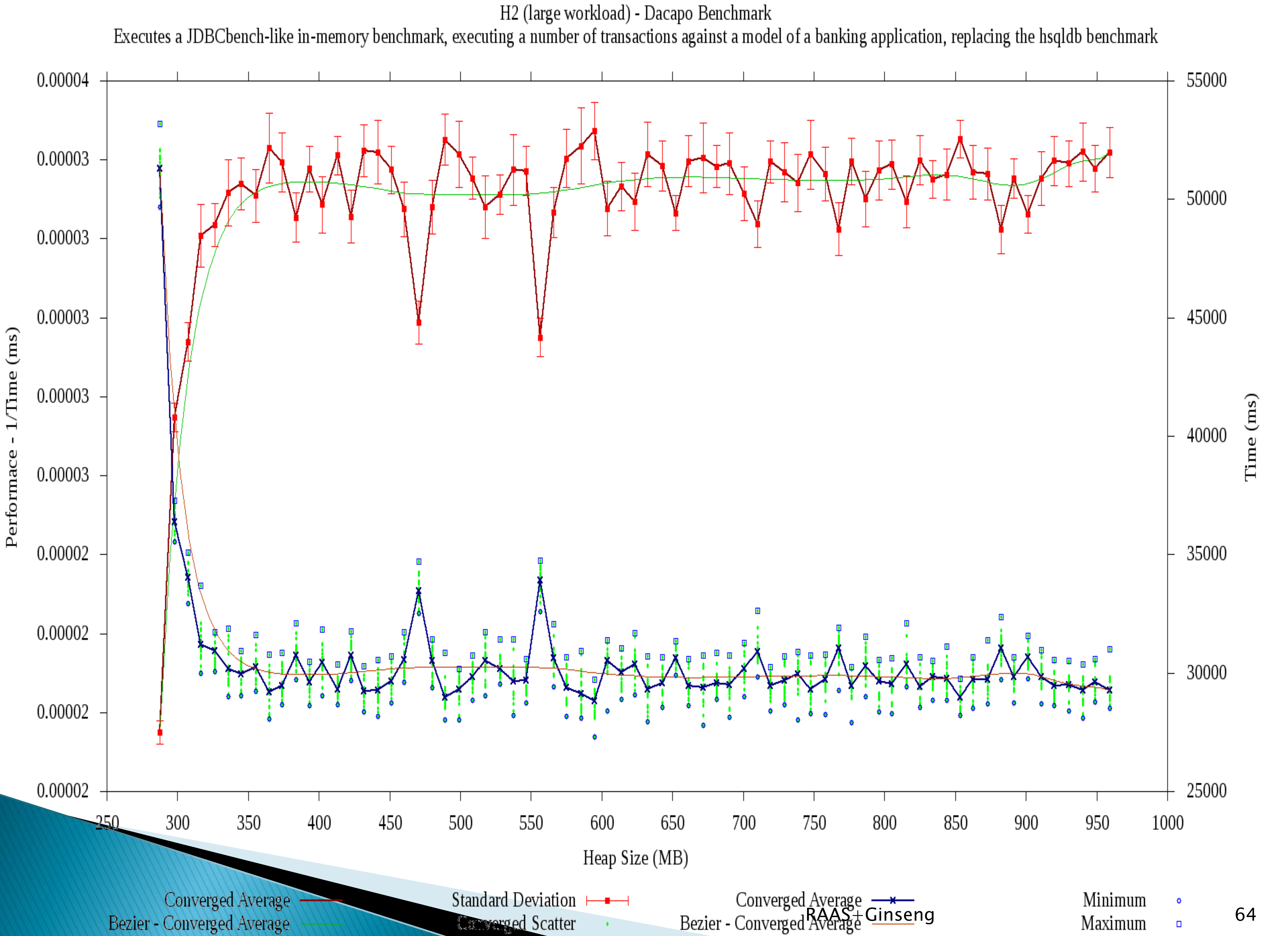

LUSEARCH (large workload) - Dacapo Benchmark
Uses lucene to do a text search of keywords over a corpus of data comprising the works of Shakespeare and the King James Bible
Performace - 1/Time (ms)
Time (ms)
Heap Size (MB)
0.00085
0.0008
0.00075
0.0007
0.00065
0.0006
0.00055
0.0005
0.00045
0.0004
2400
2200
2000
1800
1600
1400
1200
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
Converged Average
Bezier - Converged Average
Standard Deviation
Converged Scatter
Converged Average
Bezier - Converged Average
Minimum
Maximum

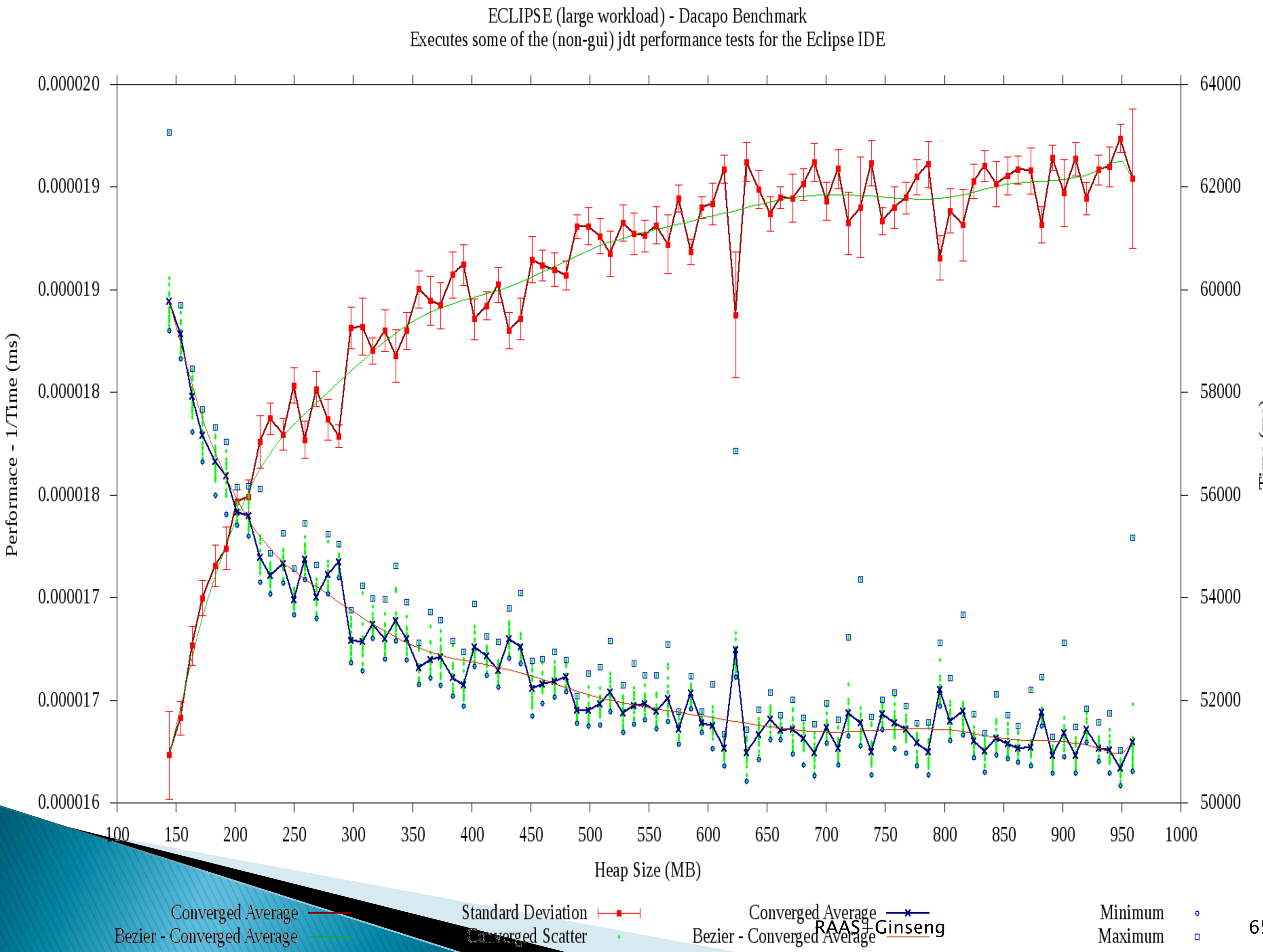

TRADESOAP (large workload) - Dacapo Benchmark
Runs the daytrader benchmark via a SOAP to a GERONIMO backend with in memory h2 as the underlying database
Performace - 1/Time (ms)
Time (ms)
Heap Size (MB)
0.00006
0.00005
0.00004
27000
26000
25000
24000
23000
22000
21000
20000
19000
18000
17000
50
100
150
200
250
300
350
400
450
500
550
600
650
700
Converged Average
Bezier - Converged Average
Standard Deviation
Converged Scatter
Converged Average
Bezier - Converged Average
Minimum
Maximum

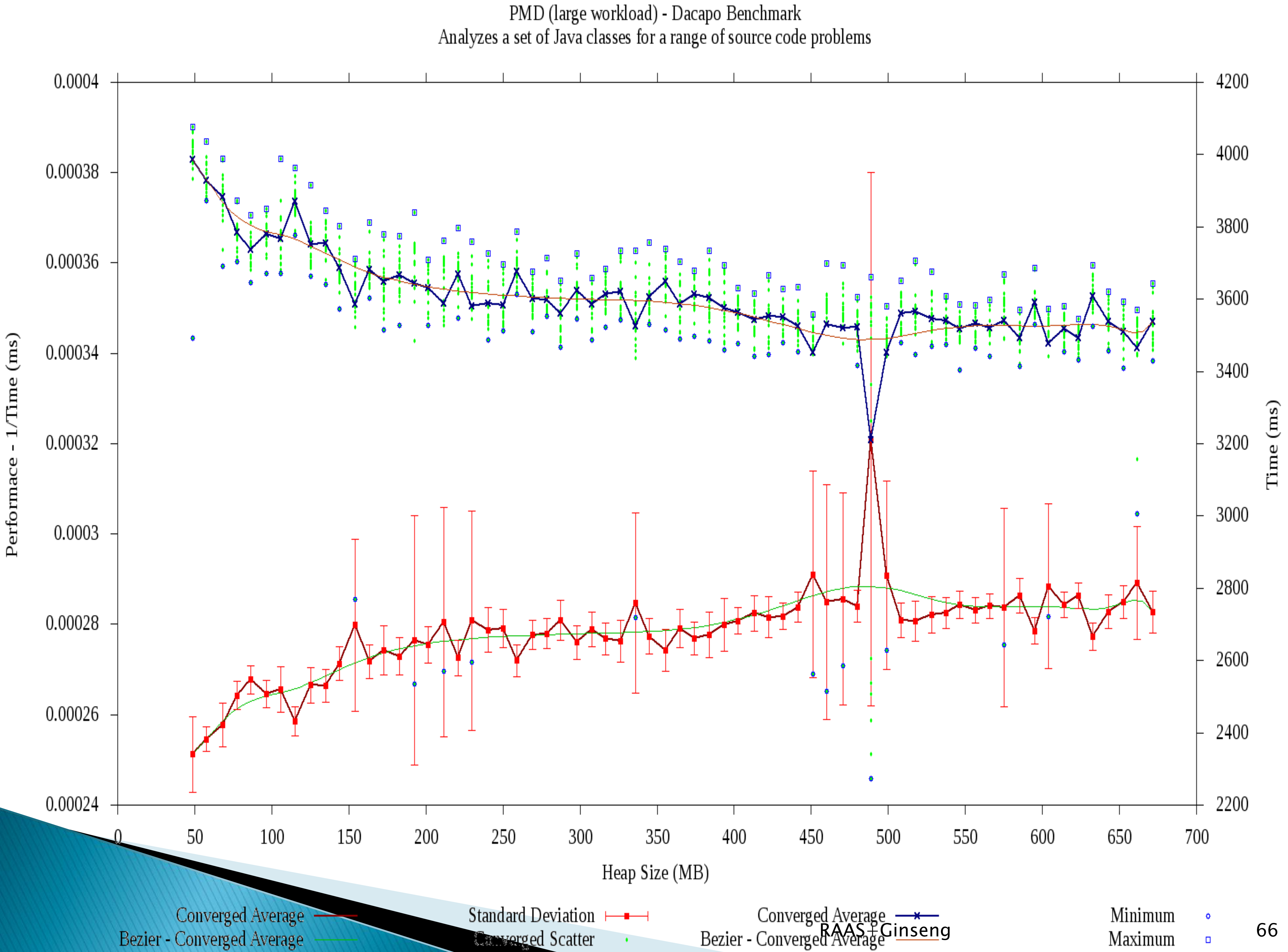


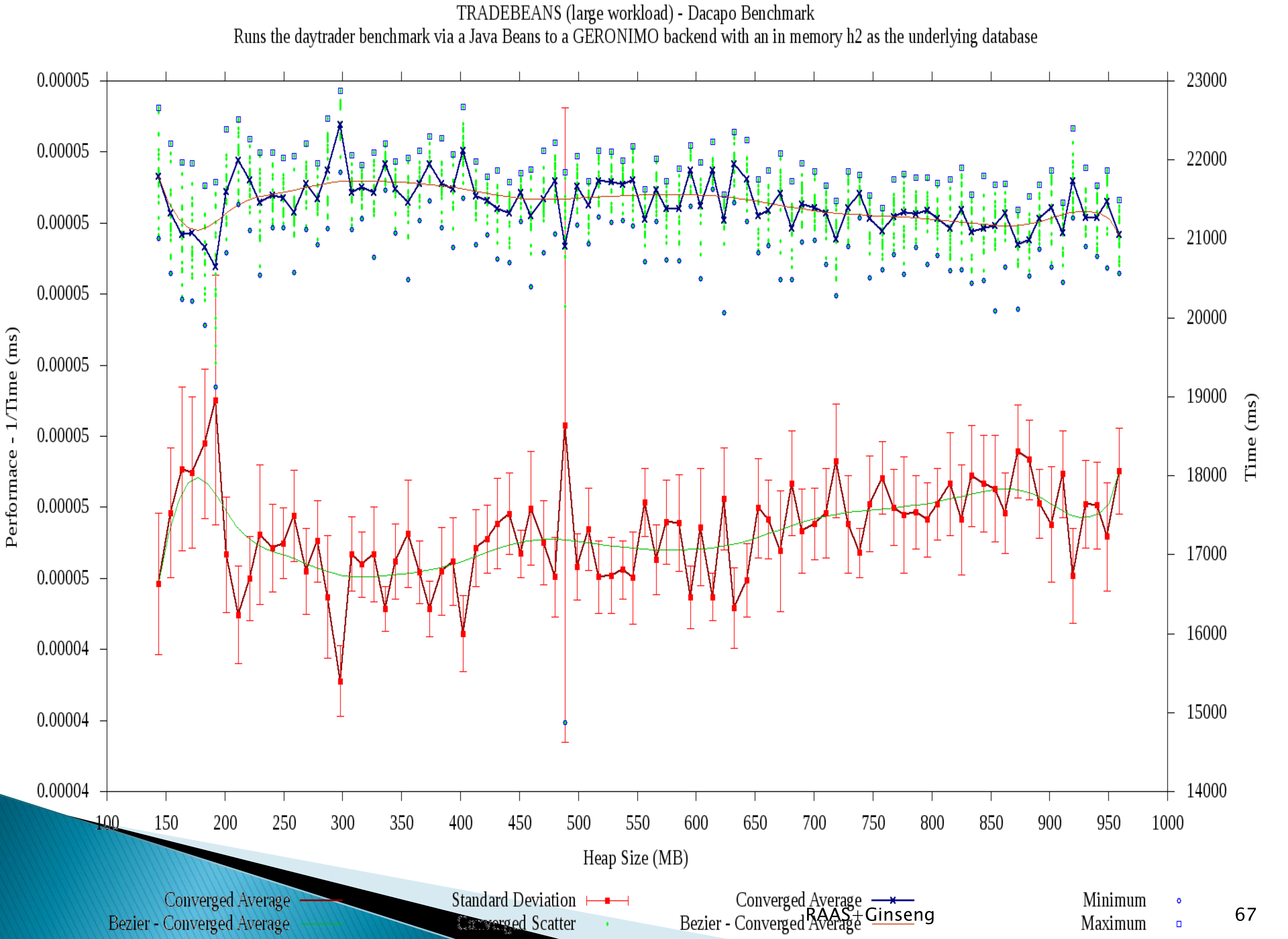
RAAS+Ginseng


TOMCAT (large workload) - Dacapo Benchmark
Runs a set of queries against a Tomcat server retrieving and verifying the resulting webpages
Performace - 1/Time (ms)
Time (ms)
Heap Size (MB)
0.000102
0.000101
0.000100
0.000099
0.000098
0.000097
0.000096
0.000095
10600
10500
10400
10300
10200
10100
10000
9900
9800
9700
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
Converged Average
Bezier - Converged Average
Standard Deviation
Converged Scatter
Converged Average
Bezier - Converged Average
Minimum
Maximum


H2 (large workload) - Dacapo Benchmark
Executes a JDBCbench-like in-memory benchmark, executing a number of transactions against a model of a banking application, replacing the hsqldb benchmark
Performace - 1/Time (ms)
Time (ms)
Heap Size (MB)
0.00004
0.00003
0.00002
55000
50000
45000
40000
35000
30000
25000
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1000
Converged Average
Bezier - Converged Average
Standard Deviation
Converged Scatter
Converged Average
Bezier - Converged Average
Minimum
Maximum


ECLIPSE (large workload) - Dacapo Benchmark
Executes some of the (non-gui) jdt performance tests for the Eclipse IDE
Performace - 1/Time (ms)
Time (ms)
Heap Size (MB)
0.000020
0.000019
0.000019
0.000018
0.000018
0.000017
0.000017
0.000016
64000
62000
60000
58000
56000
54000
52000
50000
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1000
Converged Average
Bezier - Converged Average
Standard Deviation
Converged Scatter
Converged Average
Bezier - Converged Average
Minimum
Maximum


PMD (large workload) - Dacapo Benchmark
Analyzes a set of Java classes for a range of source code problems
Performace - 1/Time (ms)
Time (ms)
Heap Size (MB)
0.0004
0.00038
0.00036
0.00034
0.00032
0.0003
0.00028
0.00026
0.00024
4200
4000
3800
3600
3400
3200
3000
2800
2600
2400
2200
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
Converged Average
Bezier - Converged Average
Standard Deviation
Converged Scatter
Converged Average
Bezier - Converged Average
Minimum
Maximum


TRADEBEANS (large workload) - Dacapo Benchmark
Runs the daytrader benchmark via a Java Beans to a GERONIMO backend with an in memory h2 as the underlying database
Performace - 1/Time (ms)
0.00005
0.00004
Time (ms)
23000
22000
21000
20000
19000
18000
17000
16000
15000
14000
Heap Size (MB)
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1000
Converged Average
Bezier - Converged Average
Standard Deviation
Converged Scatter
Converged Average
Bezier - Converged Average
Minimum
Maximum

$$\int B$$