PUG Challenge
Americas 2014

WebSpeed vs REST Adapter
Benchmarks, Statistics, and other BS

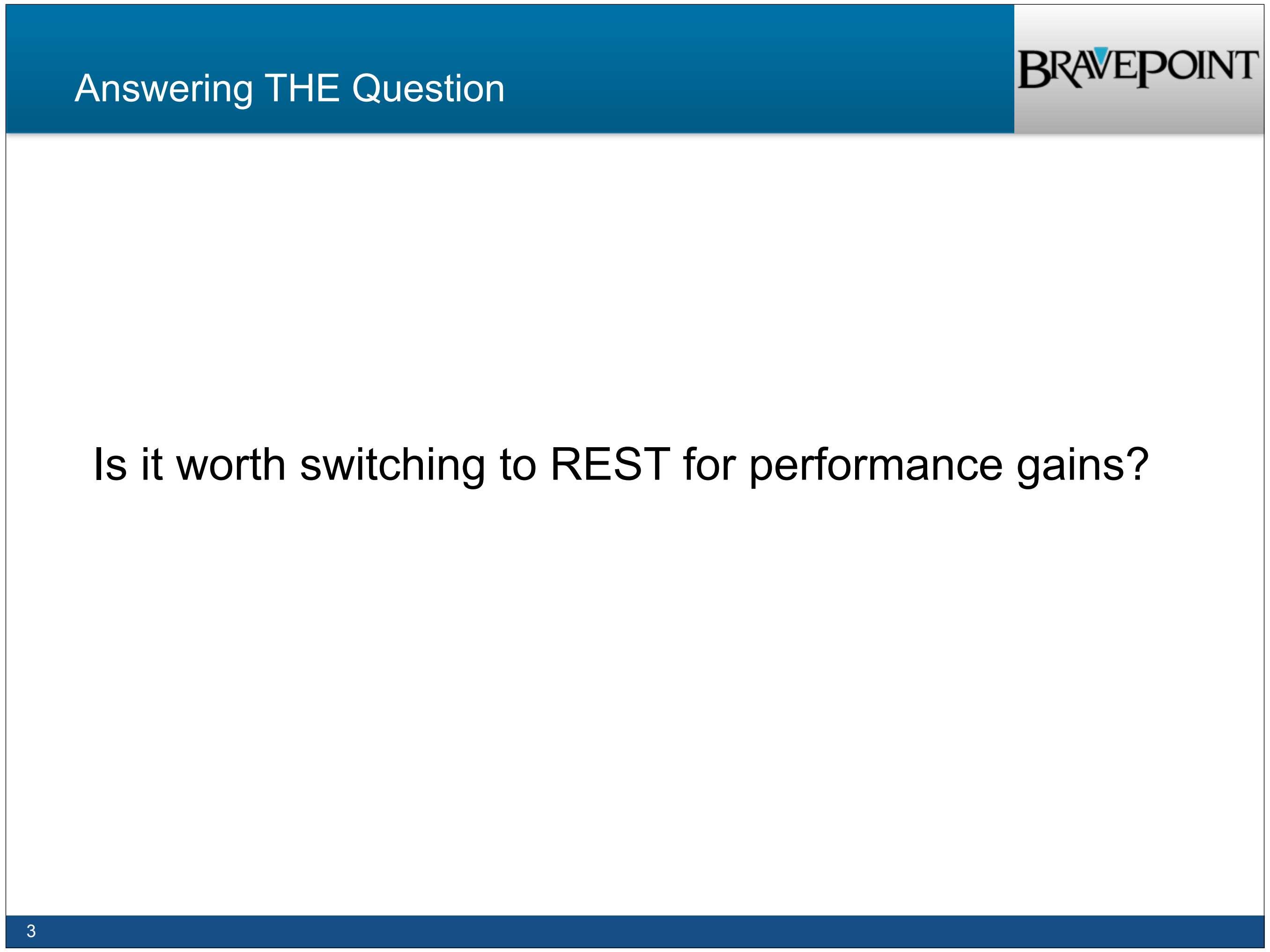
Dustin Grau, Software Architect

BRAVEPOINT

5000 Peachtree Ind. Blvd.
Suite 100
Norcross, GA 30071

About the Presenter

- Senior developer and consultant at BravePoint, Inc.
  - Founded in 1987 with currently ~125 employees
  - Consulting, training, and placement services
- WebSpeed application developer since 1999
- Implementing JS/AJAX frameworks since 2010
- Lead architect for modernization framework “Application Evolution”



Answering THE Question

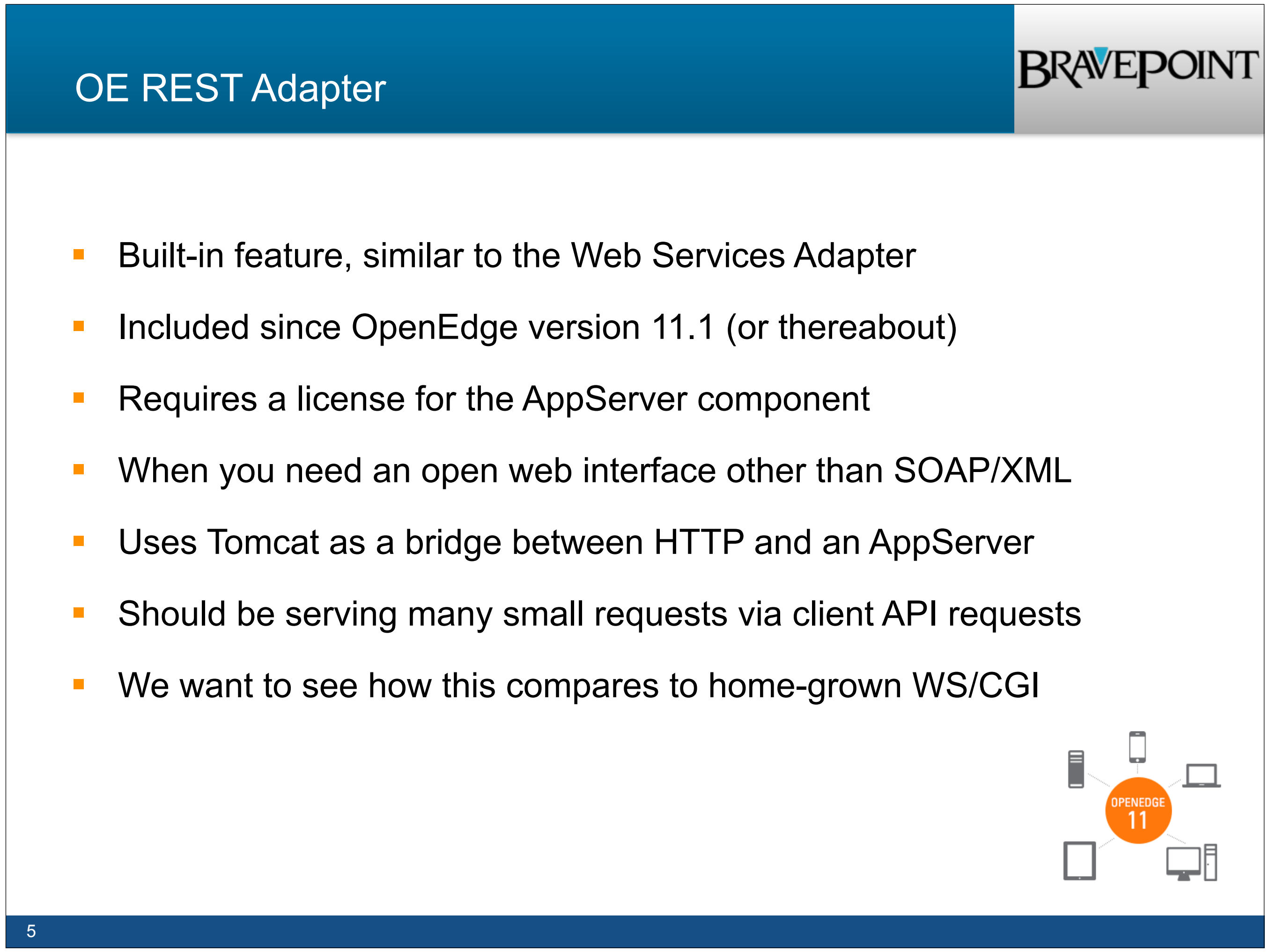
Is it worth switching to REST for performance gains?

This is the question we always get asked by anyone looking at OE 11.
Poll: How many people are using the new REST adapter?

Session Overview

BRAVEPOINT

- Technology introduction: the OE REST Adapter
- A primer on JMeter: configuration and running
- Baseline configs: WebSpeed (Apache) and OE Rest (Tomcat)
- Establishing thresholds and parameters for testing
- Benchmarking and results
- Crunching the numbers



OE REST Adapter

BRAVEPOINT

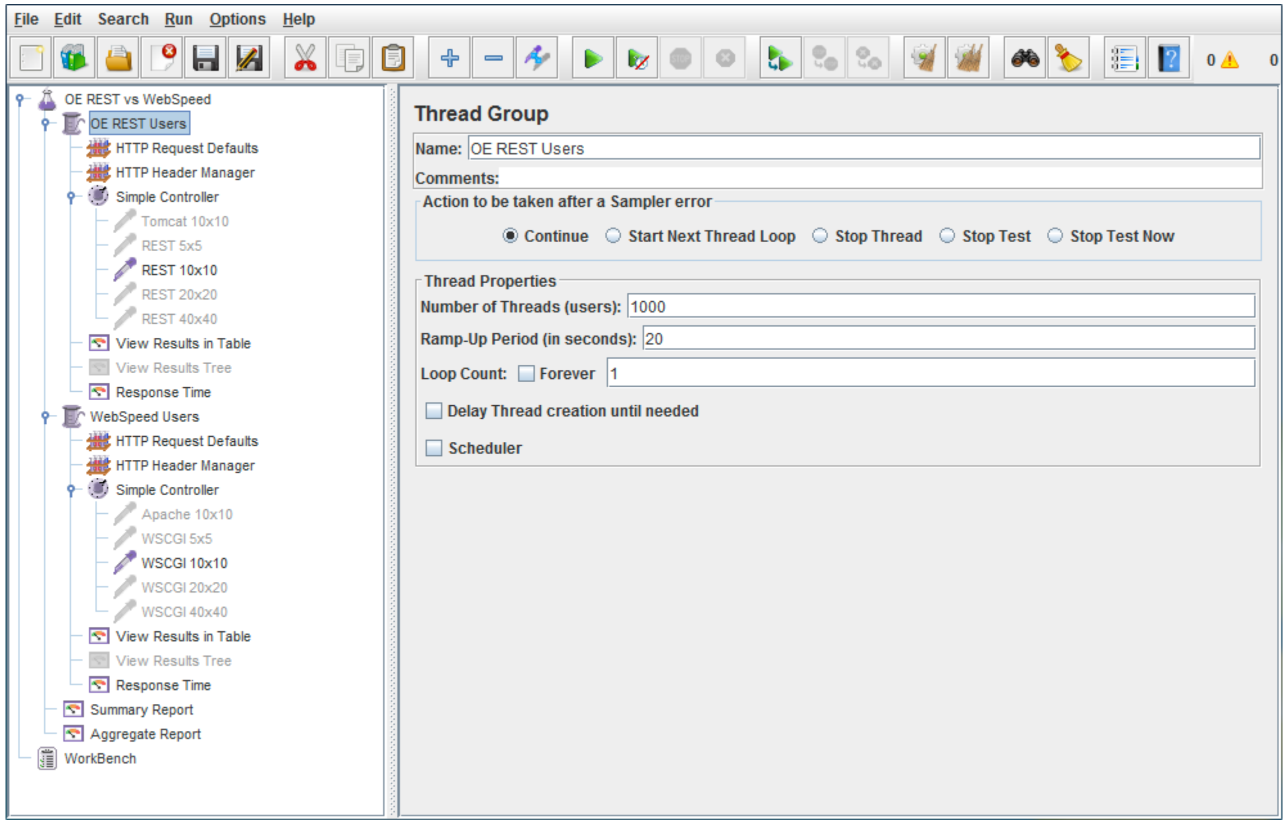
- Built-in feature, similar to the Web Services Adapter
- Included since OpenEdge version 11.1 (or thereabout)
- Requires a license for the AppServer component
- When you need an open web interface other than SOAP/XML
- Uses Tomcat as a bridge between HTTP and an AppServer
- Should be serving many small requests via client API requests
- We want to see how this compares to home-grown WS/CGI

Unfortunately there’s not enough time here to go over the setup process for REST.
Instructions are available in the PDSOE documentation from the Progress Communities site.

Apache JMeter

- Open-source, Java-based load testing suite
- Can be obtained at http://jmeter.apache.org
- Similar to ApacheBench (ab), Siege, and others
- Includes both benchmarking and reporting tools
- Useful for building complex/looping test cases
- Let us explore how this works…

Apache JMeter - Threads



- OE REST vs WebSpeed
  - OE REST Users
    - HTTP Request Defaults
    - HTTP Header Manager
    - Simple Controller
      - Tomcat 10x10
      - REST 5x5
      - REST 10x10
      - REST 20x20
      - REST 40x40
    - View Results in Table
    - View Results Tree
    - Response Time
  - WebSpeed Users
    - HTTP Request Defaults
    - HTTP Header Manager
    - Simple Controller
      - Apache 10x10
      - WSCGI 5x5
      - WSCGI 10x10
      - WSCGI 20x20
      - WSCGI 40x40
    - View Results in Table
    - View Results Tree
    - Response Time
  - Summary Report
  - Aggregate Report
- WorkBench

Thread Group

Name: OE REST Users

Comments:

Action to be taken after a Sampler error

● Continue ○ Start Next Thread Loop ○ Stop Thread ○ Stop Test ○ Stop Test Now

Thread Properties

Number of Threads (users): 1000

Ramp-Up Period (in seconds): 20

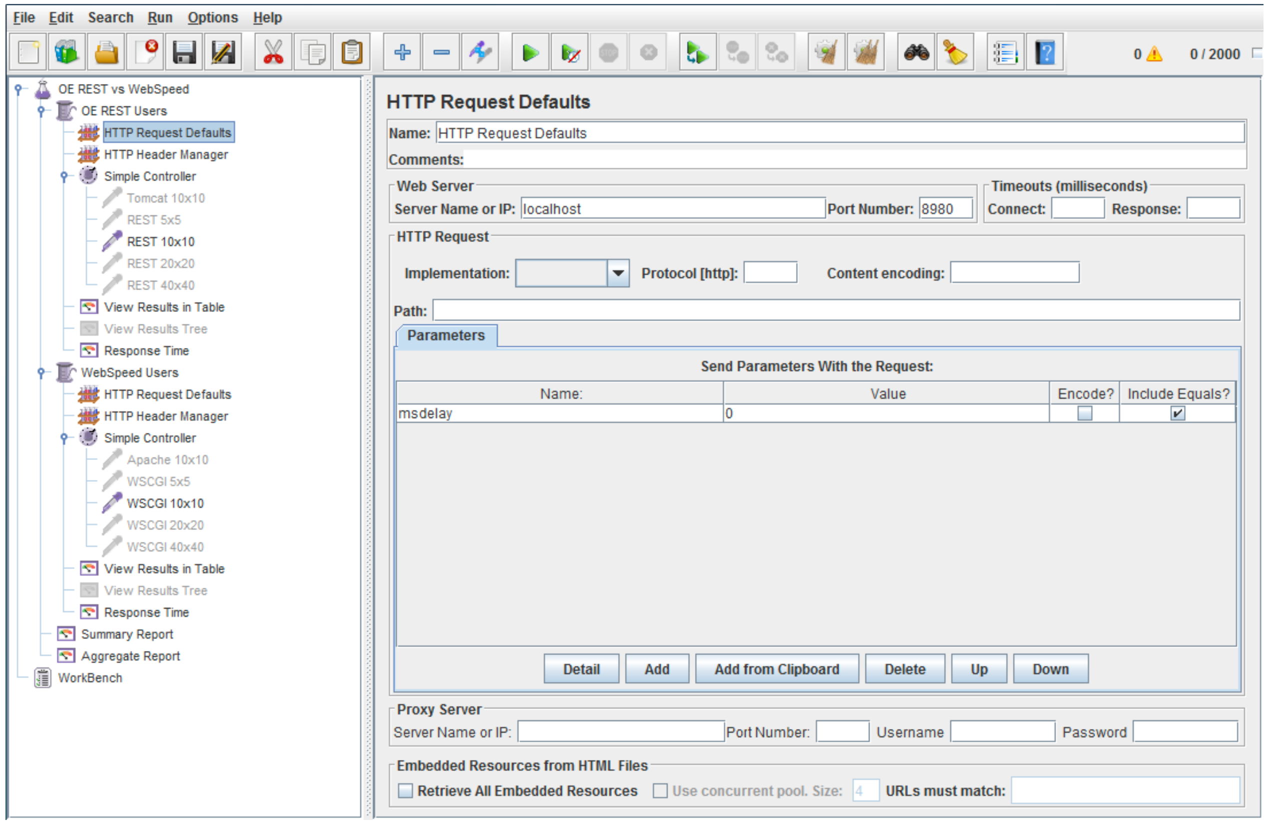
Loop Count: □ Forever 1

□ Delay Thread creation until needed

□ Scheduler

All testing begins with “threads” which defines the load you’ll be placing on your server.

Apache JMeter - Defaults



- OE REST vs WebSpeed
  - OE REST Users
    - HTTP Request Defaults
    - HTTP Header Manager
    - Simple Controller
      - Tomcat 10x10
      - REST 5x5
      - REST 10x10
      - REST 20x20
      - REST 40x40
    - View Results in Table
    - View Results Tree
    - Response Time
  - WebSpeed Users
    - HTTP Request Defaults
    - HTTP Header Manager
    - Simple Controller
      - Apache 10x10
      - WSCGI 5x5
      - WSCGI 10x10
      - WSCGI 20x20
      - WSCGI 40x40
    - View Results in Table
    - View Results Tree
    - Response Time
  - Summary Report
  - Aggregate Report
- WorkBench

HTTP Request Defaults

Name: HTTP Request Defaults

Comments:

Web Server
Server Name or IP: localhost Port Number: 8980

Timeouts (milliseconds)
Connect: Response:

HTTP Request
Implementation: Protocol [http]: Content encoding:

Path:

Parameters

Send Parameters With the Request:

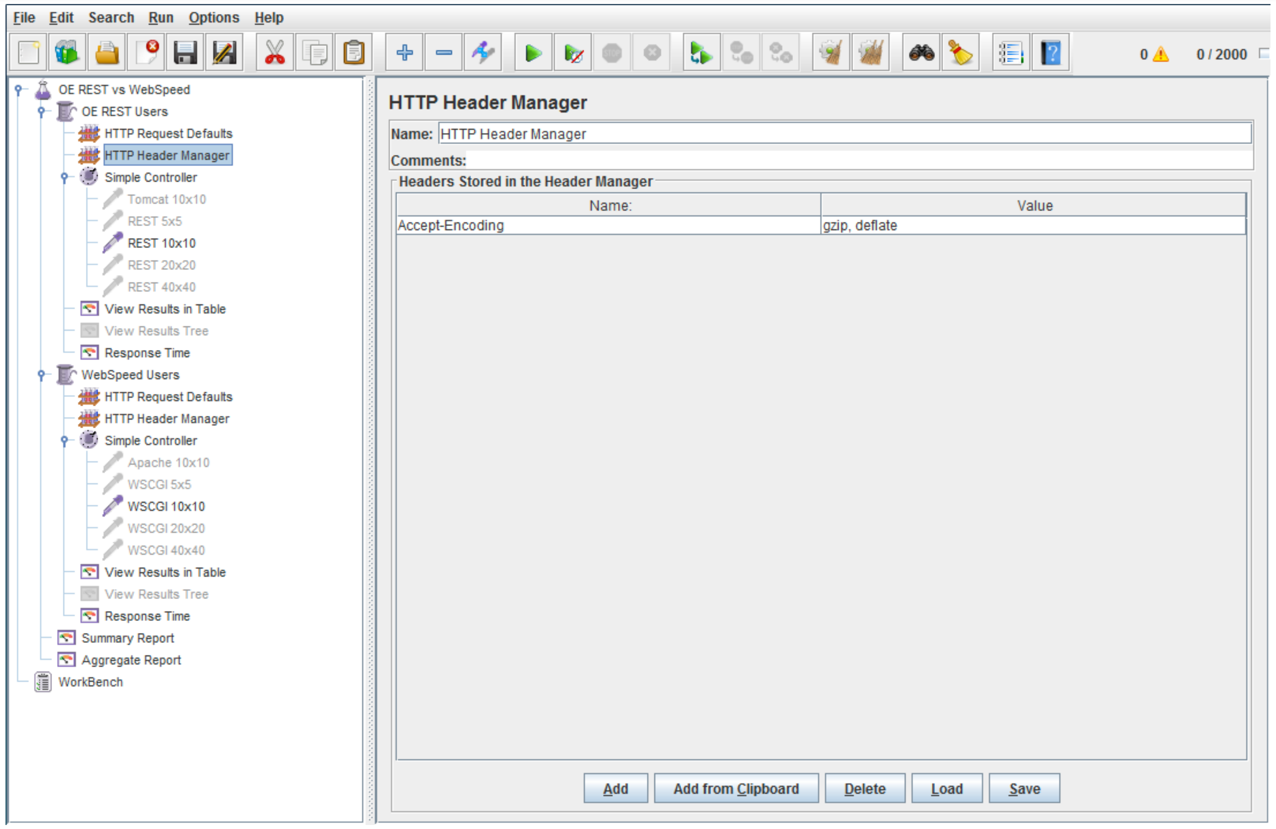
Name:	Value	Encode?	Include Equals?
msdelay	0	□	☑

Proxy Server
Server Name or IP: Port Number: Username Password

Embedded Resources from HTML Files
□ Retrieve All Embedded Resources □ Use concurrent pool. Size: 4 URLs must match:

By assigning HTTP defaults, you make your test scenarios easier to manage.

Apache JMeter - Headers



HTTP Header Manager

Name: HTTP Header Manager

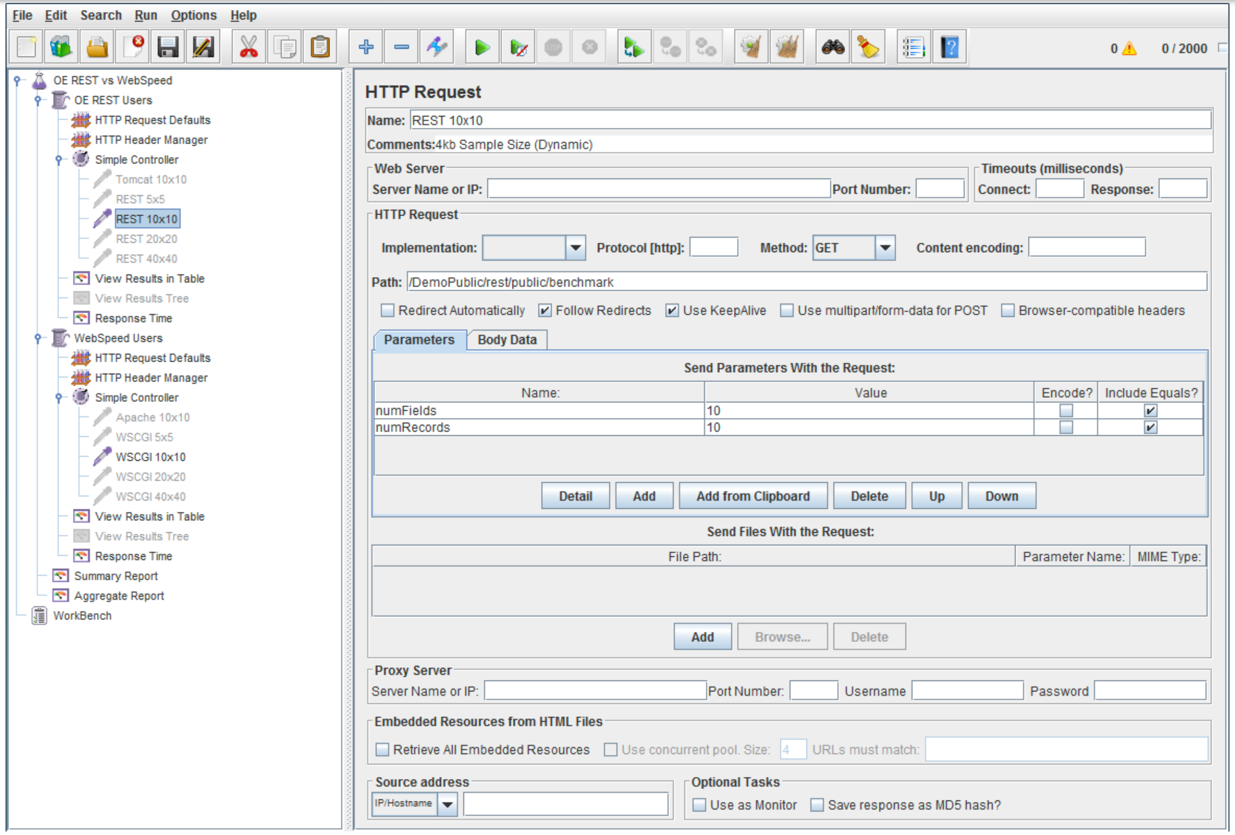
Comments:

Headers Stored in the Header Manager

Name:	Value
Accept-Encoding	gzip, deflate

Headers can include things like REALM authentication, or as seen here compression settings.

Apache JMeter - Requests



HTTP Request

Name: REST 10x10

Comments: 4kb Sample Size (Dynamic)

Web Server

Server Name or IP:
Port Number:

Timeouts (milliseconds)

Connect:
Response:

HTTP Request

Implementation:
Protocol [http]:
Method: GET
Content encoding:

Path: /DemoPublic/rest/public/benchmark

□ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive □ Use multipart/form-data for POST □ Browser-compatible headers

Send Parameters With the Request:

Name:	Value	Encode?	Include Equals?
numFields	10	□	☑
numRecords	10	□	☑

Send Files With the Request:

File Path:	Parameter Name:	MIME Type:

Proxy Server

Server Name or IP:
Port Number:
Username
Password

Embedded Resources from HTML Files

□ Retrieve All Embedded Resources □ Use concurrent pool. Size: 4 URLs must match:

Source address

IP/Hostname

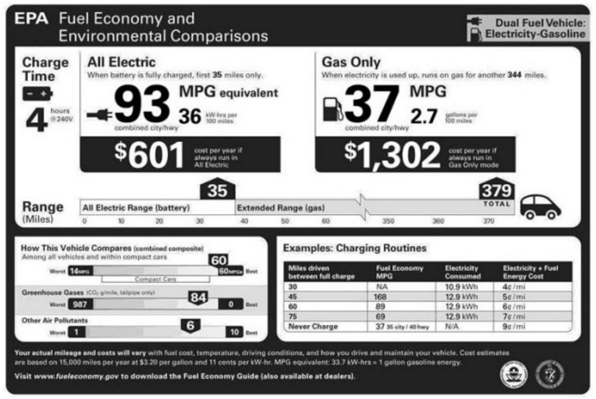
Optional Tasks

□ Use as Monitor □ Save response as MD5 hash?

Test scenarios can include URL parameters and a specific URL.

Known Variables

- Mid-2012 MacBook Pro (Quad-Core i7, 16GB)
- VMWare running Windows 7 32-bit
  - 2-CPU with 2 GB Memory
  - All requests go to localhost
  - Rebooted VM between tests
- Static test is “10x10” JSON object
  - 10 rows of data (temp-table)
  - 10 fields of consistent size
  - 4.4 kilobytes (4,481 bytes)



EPA Fuel Economy and Environmental Comparisons

Dual Fuel Vehicle: Electricity-Gasoline

Charge Time
4 hours @240V

All Electric
When battery is fully charged, first **35** miles only.
93 MPG equivalent combined city/hwy
36 kW-hrs per 100 miles
$601 cost per year if always run in All Electric

Gas Only
When electricity is used up, runs on gas for another **344** miles.
37 MPG combined city/hwy
2.7 gallons per 100 miles
$1,302 cost per year if always run in Gas Only mode

Range (Miles)
All Electric Range (battery) 35
Extended Range (gas) 379 TOTAL

How This Vehicle Compares (combined composite)
Among all vehicles and within compact cars
Worst 14MPG — 60 — 60MPGe Best
Compact Cars

Greenhouse Gases (CO₂ g/mile, tailpipe only)
Worst 987 — 84 — 0 Best

Other Air Pollutants
Worst 1 — 6 — 10 Best

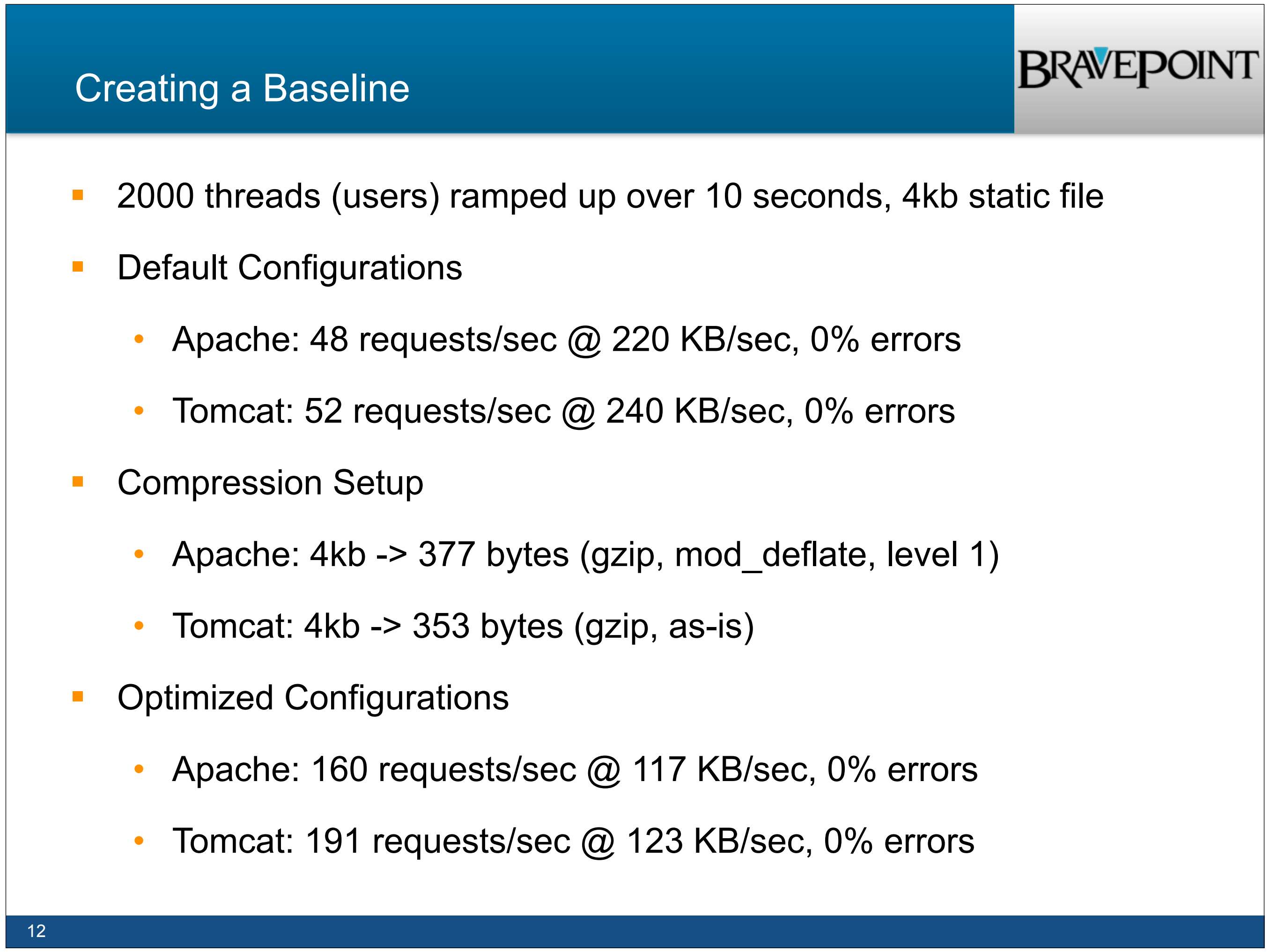
Examples: Charging Routines

Miles driven between full charge	Fuel Economy MPG	Electricity Consumed	Electricity + Fuel Energy Cost
30	NA	10.9 kWh	4¢ /mi
45	168	12.9 kWh	5¢ /mi
60	89	12.9 kWh	6¢ /mi
75	69	12.9 kWh	7¢ /mi
Never Charge	37 35 city / 40 hwy	N/A	9¢ /mi

Your actual mileage and costs will vary with fuel cost, temperature, driving conditions, and how you drive and maintain your vehicle. Cost estimates are based on 15,000 miles per year at $3.20 per gallon and 11 cents per kW-hr. MPG equivalent: 33.7 kW-hrs = 1 gallon gasoline energy.

Visit *www.fueleconomy.gov* to download the Fuel Economy Guide (also available at dealers).

Your mileage will always vary, and is about as ridiculous as assigning MPG on an electric car…



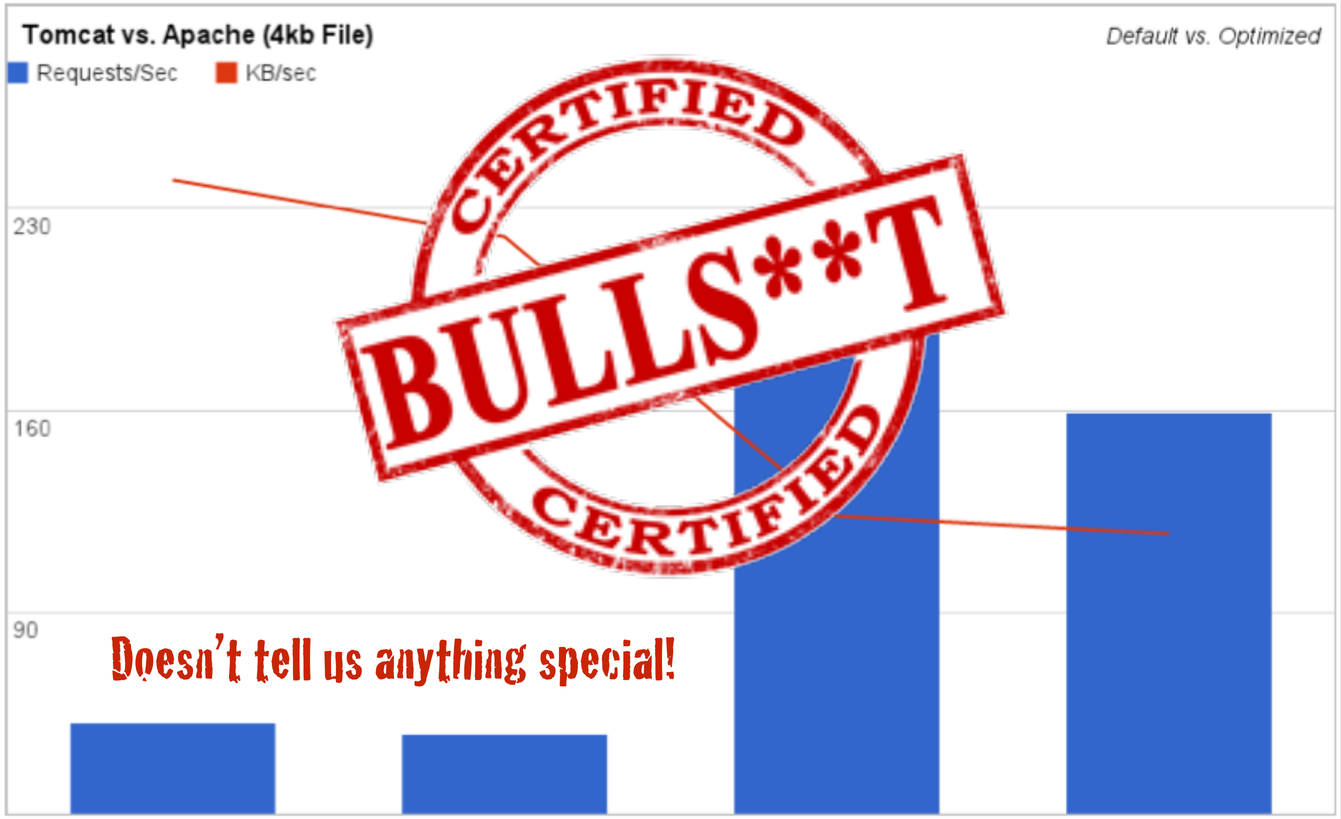
Creating a Baseline

BRAVEPOINT

- 2000 threads (users) ramped up over 10 seconds, 4kb static file
- Default Configurations
  - Apache: 48 requests/sec @ 220 KB/sec, 0% errors
  - Tomcat: 52 requests/sec @ 240 KB/sec, 0% errors
- Compression Setup
  - Apache: 4kb -> 377 bytes (gzip, mod_deflate, level 1)
  - Tomcat: 4kb -> 353 bytes (gzip, as-is)
- Optimized Configurations
  - Apache: 160 requests/sec @ 117 KB/sec, 0% errors
  - Tomcat: 191 requests/sec @ 123 KB/sec, 0% errors

This will only tell us how quickly Apache and Tomcat can handle requests.

Creating a Baseline



CERTIFIED
BULLS**T
CERTIFIED

Doesn't tell us anything special!

You should always be using compression. Period.

Thresholds and Parameters

- What are we testing?
  - Dynamic data packets, created by WebSpeed or AppServer
- CPU and memory concerns?
  - I’m using a VM; CPU and memory could be increased
  - Expect different results with different hardware (surprise!)
- I/O, HDD vs SSD?
  - Not a factor, all data being generated never touches disk
  - No attached databases, but we are logging (minimal info)

Other Considerations

- Using 10 agents (min/max/initial) for WebSpeed and AppServer
- WebSpeed is not using Nameserver; directly to port (cgiip)
- Same for AppServer, using AppserverDC in runtime.props
- No special performance tuning (wsbroker1, restbroker1)
- No changes to default Tomcat memory
- No further tweaks to Apache workers
- Only making GET requests

Adding WS and AS

- Same configuration (2000 users, 10 seconds), 4kb dynamic packet
  - WebSpeed: 85 requests/sec @ 153 KB/sec, 82% errors
  - AppServer: 87 requests/sec @ 161 KB/sec, 84% errors
- Looking at broker statistics (maximums)
  - WS: Queue Depth 0; Req. Wait 2ms; Req Duration 2,726ms
  - AS: Queue Depth 30; Req Wait 1,843ms; Req Duration 2,230ms
- Errors indicate refused connections, HTTP-500, or other problems
- We want to test near the limits of our system, not exceed them…

Re-running WS and AS

- New configuration (1000 users, 10 seconds), 4kb dynamic packet
  - WebSpeed: 67 requests/sec @ 67 KB/sec, 36% errors
  - AppServer: 80 requests/sec @ 86 KB/sec, 30% errors
- Looking at broker statistics (maximums)
  - WS: Queue Depth 8; Req Wait 618ms; Req Duration 722ms
  - AS: Queue Depth 8; Req Wait 178ms; Req Duration 1,774ms
- Errors have been reduced but the load on the brokers is still high
- So it’s better, but let’s try this again…

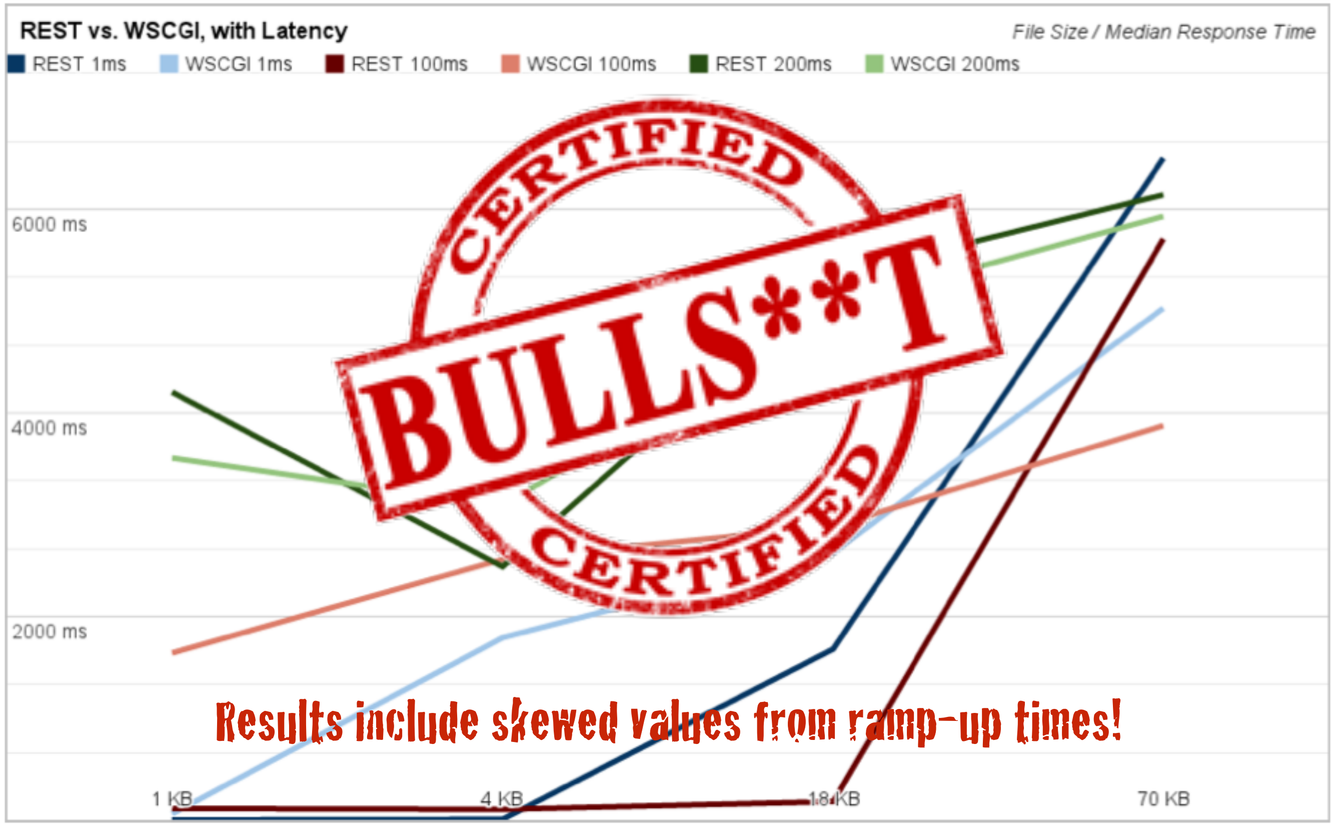
Re-running WS and AS

- Final configuration (1000 users, 20 seconds), 4kb dynamic packet
  - WebSpeed: 49 requests/sec @ 31 KB/sec, 0% errors
  - AppServer: 51 requests/sec @ 32 KB/sec, 0% errors
- Looking at broker statistics (maximums)
  - WS: Queue Depth 3; Req Wait 307ms; Req Duration 347ms
  - AS: Queue Depth 2; Req Wait 46ms; Req Duration 1,405ms
- Lesson: the HTTP servers can keep up better than the brokers
- We have a threshold, so that’s good enough to continue!

Running the Benchmarks

- Ramping up data packets (file size)
  - 5x5 = 1kb (ideal)
  - 10x10 = 4kb (common)
  - 20x20 = 18kb (large)
  - 40x40 = 70kb (object)
- Ramping up processing time (latency)
  - 0-5ms (base tests)
  - 100ms (typical case)
  - 200ms (heavy query)
  - Using sub-second pause

File Size vs. Latency (1000 Users, 10 Agents)

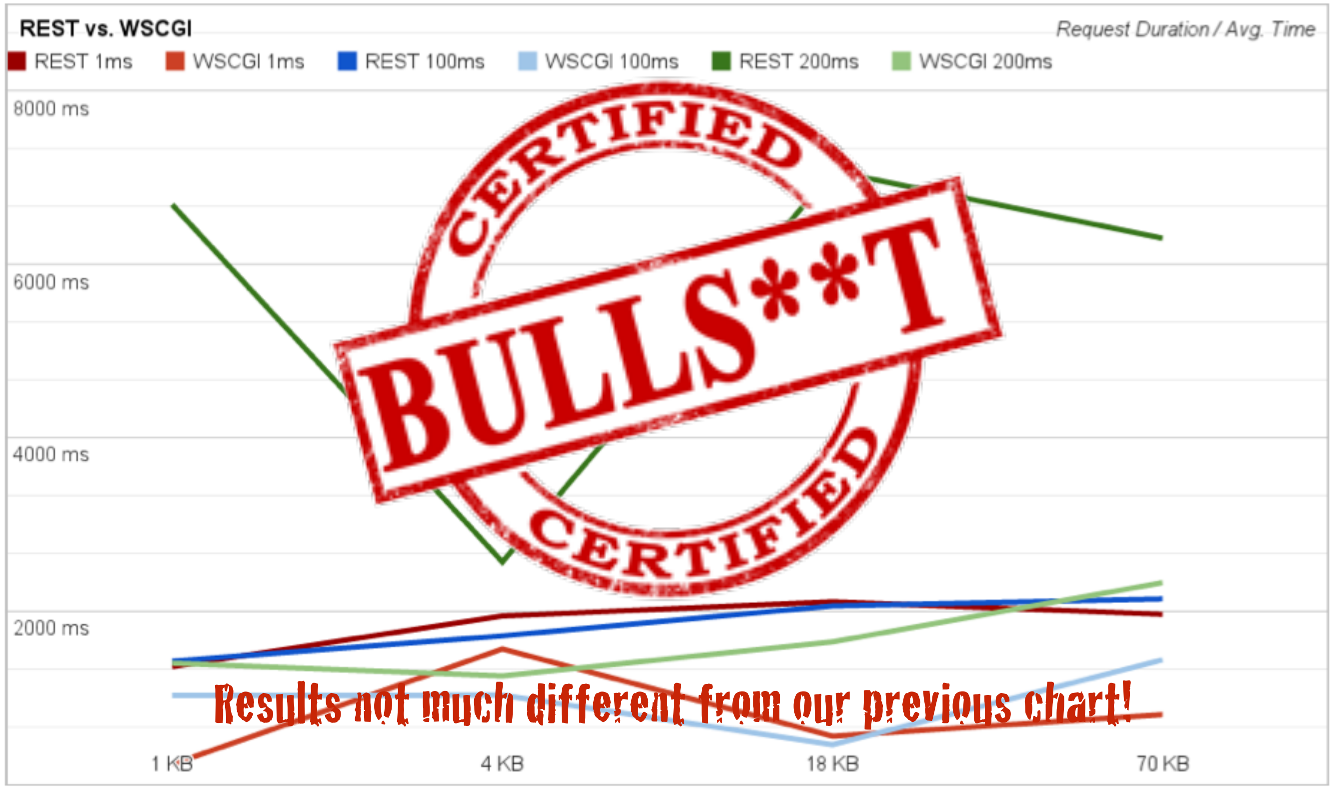


Results include skewed values from ramp-up times!

We can't live on averages alone, because the actual test results show more information.

Agent Request Duration (Average) [1000 / 10]

BRAVEPOINT

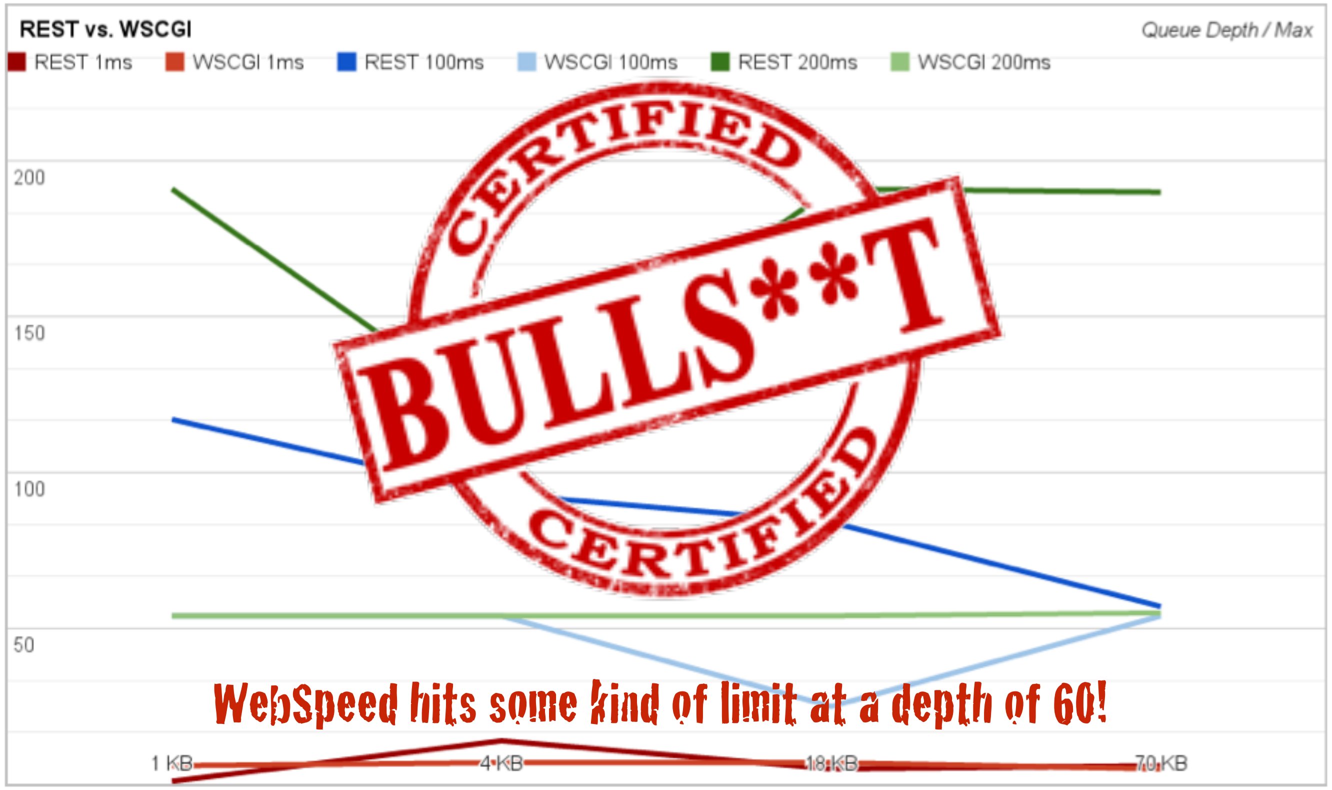


CERTIFIED BULLS**T CERTIFIED

Results not much different from our previous chart!

We’re still working with averages, just from the Progress perspective.

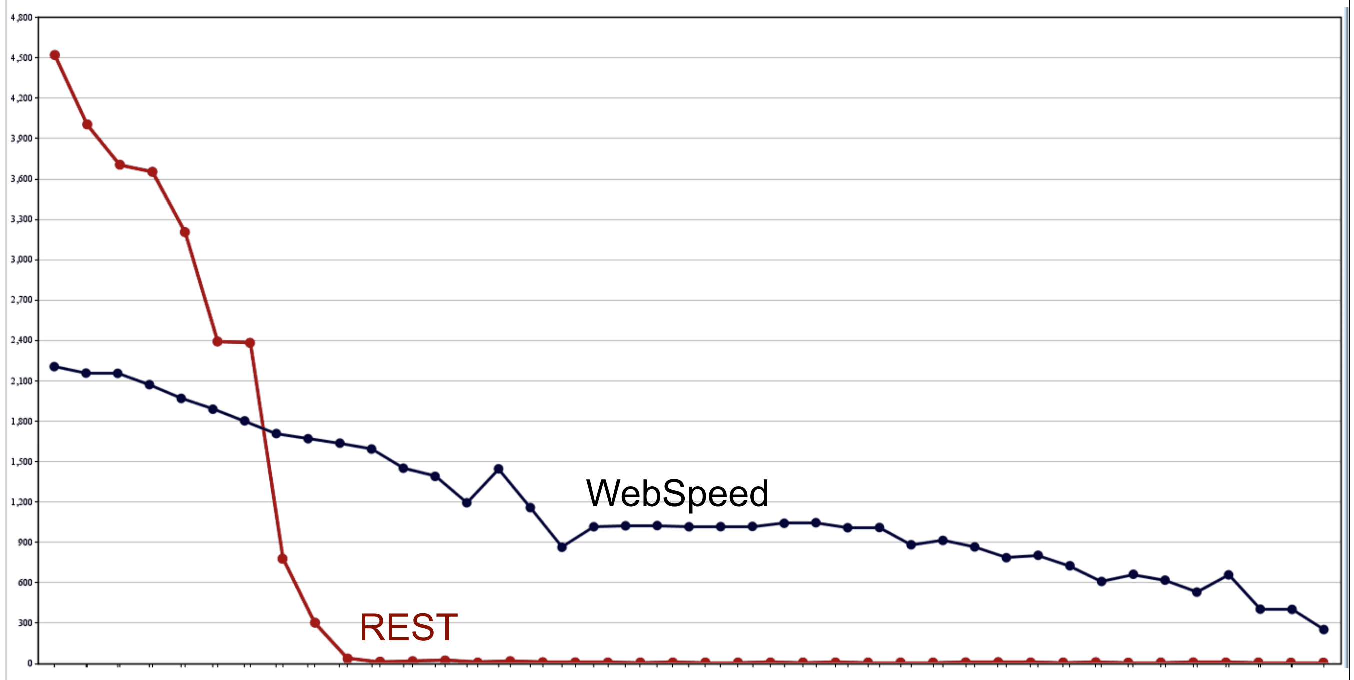
Agent Queue Depth (Max) [1000 / 10]



WebSpeed hits some kind of limit at a depth of 60!

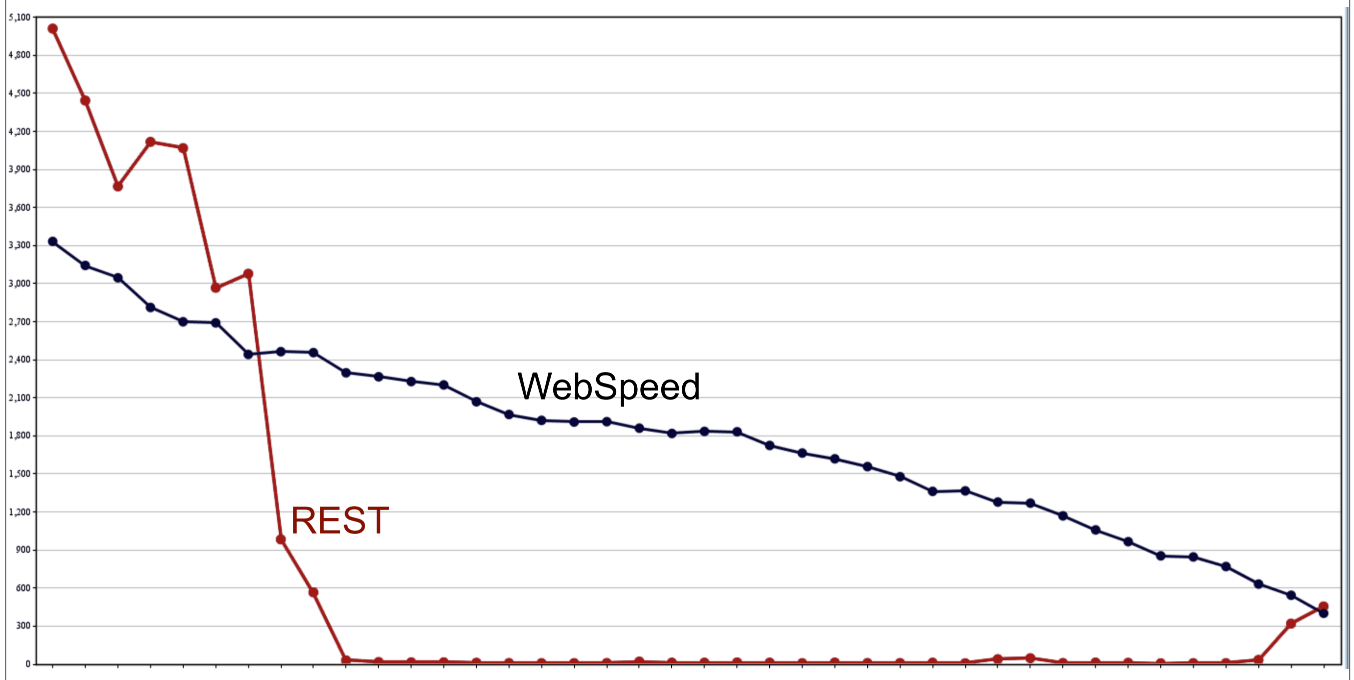
We may be limited by some kind of process limit in Apache, so this can’t be taken as truth.

1kb @ 0ms (1000 Users, 10 Agents)



Theory: Tomcat needs to ramp up memory to handle the initial load, but afterwards can process and clear the backlog.
WebSpeed manages to keep up with the load relatively well for small packet sizes.

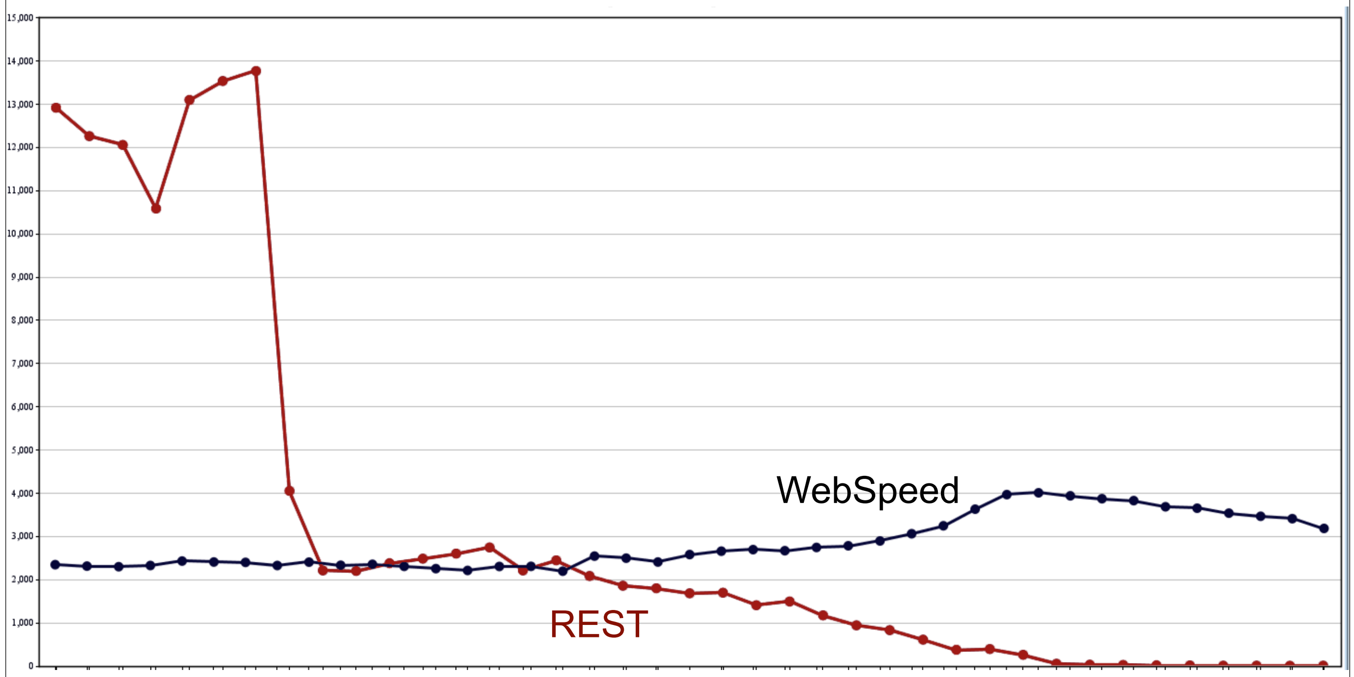
4kb @ 0ms (1000 Users, 10 Agents)



Nearly the same view as before, just overall longer response times.

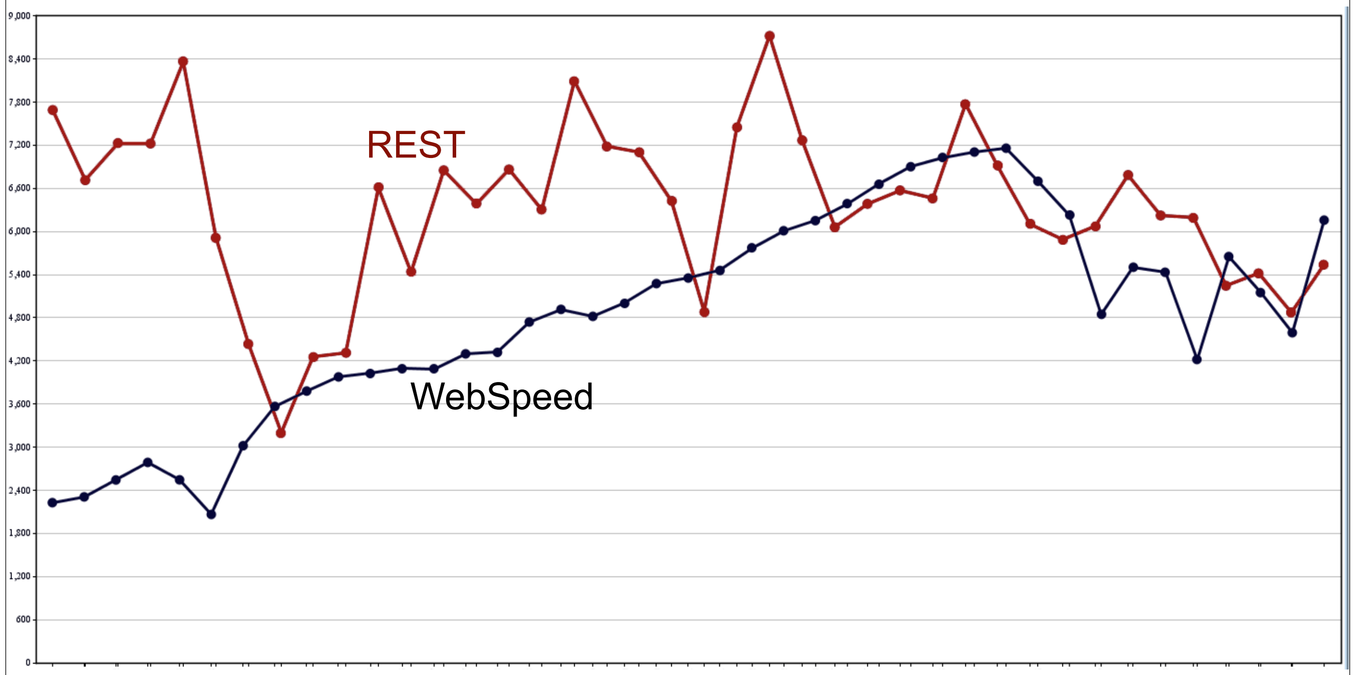
18kb @ 0ms (1000 Users, 10 Agents)

BRAVEPOINT



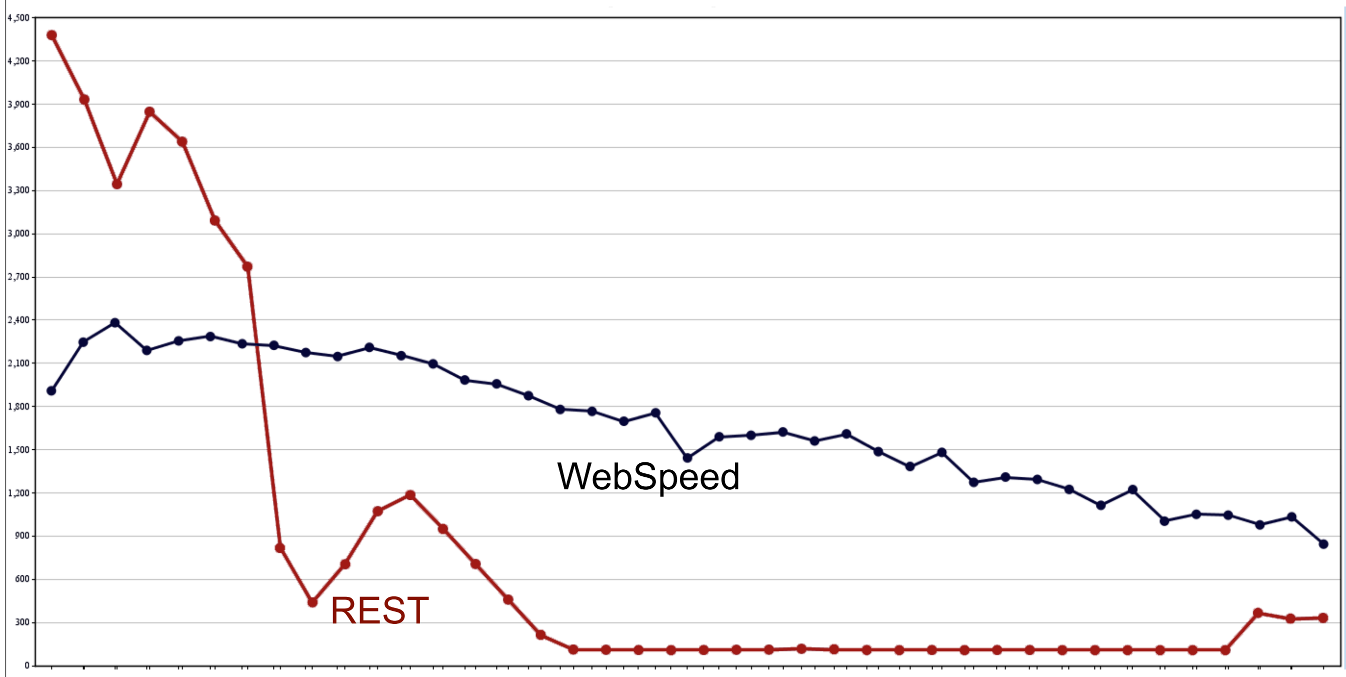
Now we're seeing some results. As packet size increases, the playing field is more level.

70kb @ 0ms (1000 Users, 10 Agents)



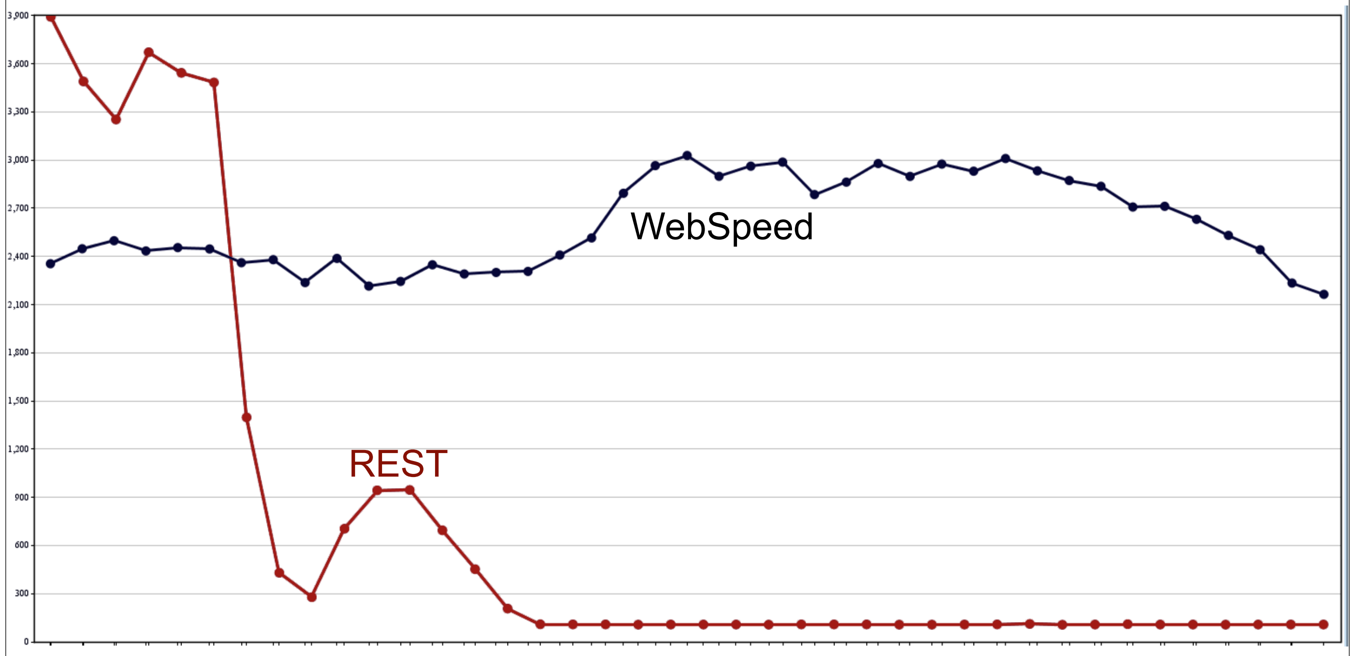
Large packet sizes are no good for anyone over extended periods of time.

1kb @ 100ms (1000 Users, 10 Agents)



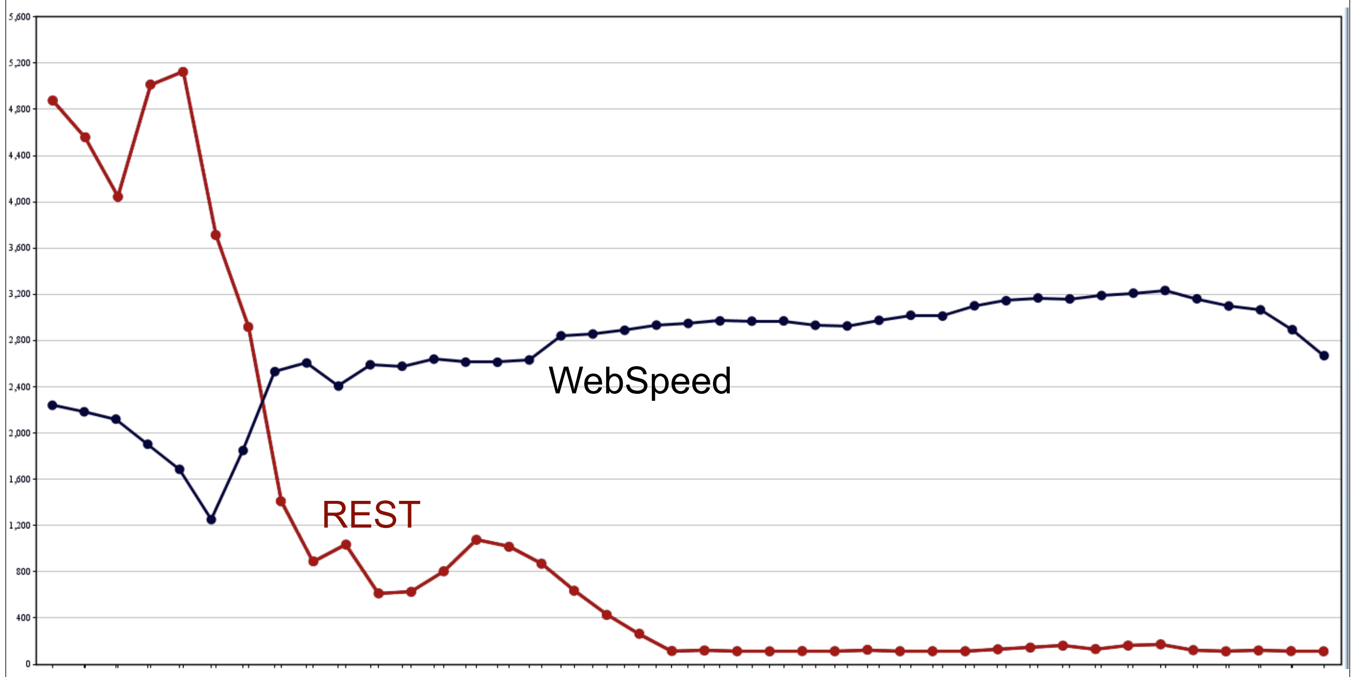
Starting over with our small packet, but taking longer to process on the back-end.
This is the same pattern we saw earlier, just with slightly longer response times.

4kb @ 100ms (1000 Users, 10 Agents)



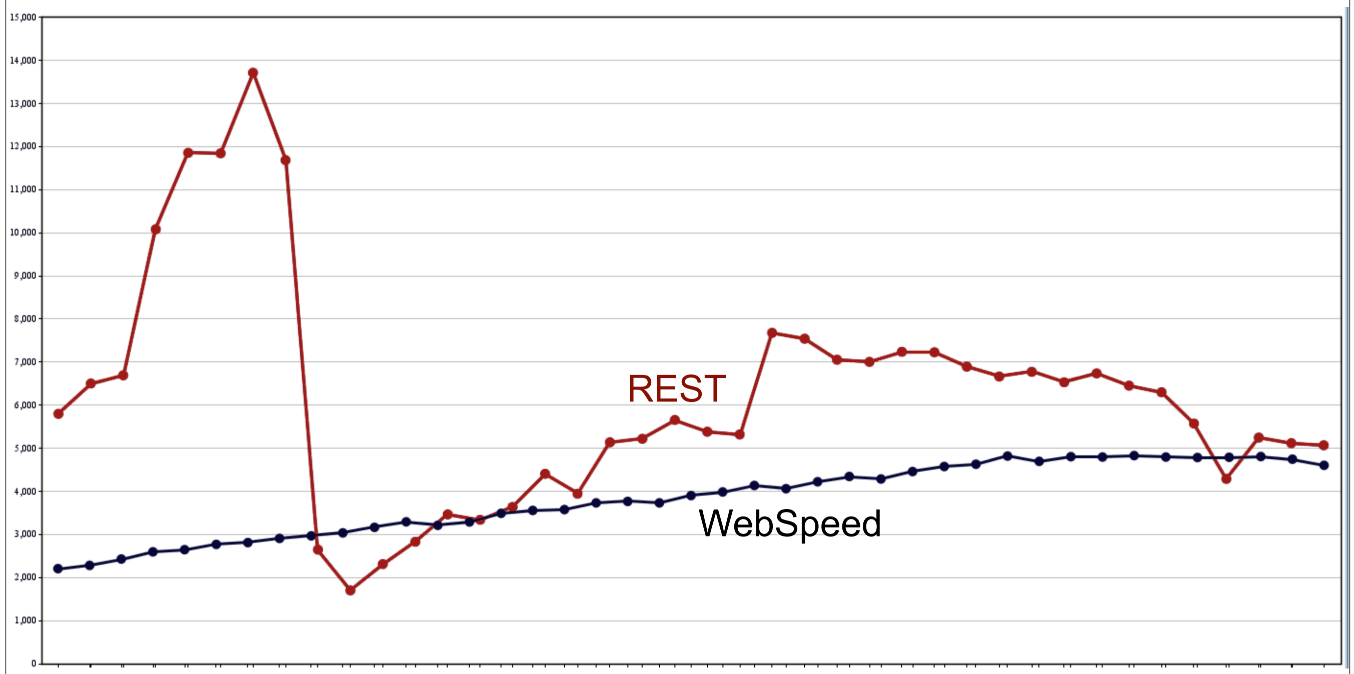
Again, once Tomcat has ramped up memory and threads, it can handle the load better.
WebSpeed begins to show it's limits, probably related to that queue depth limit we saw earlier.

18kb @ 100ms (1000 Users, 10 Agents)



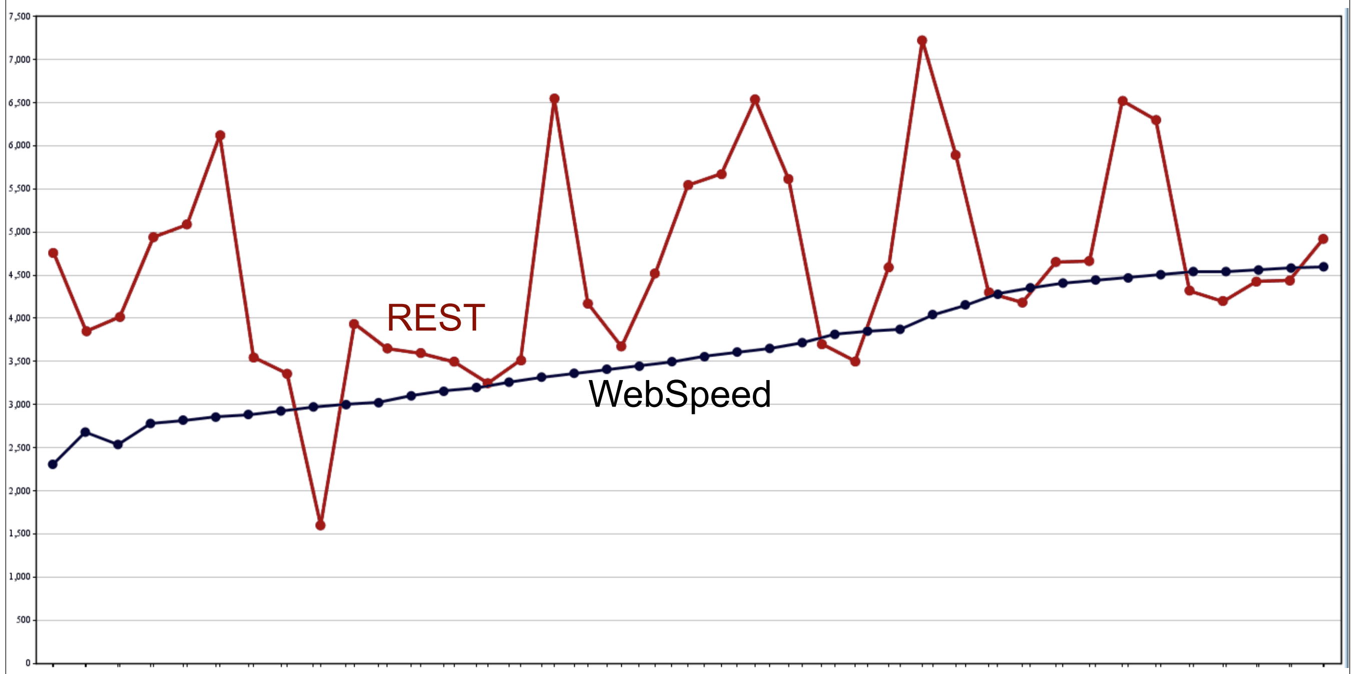
The response times for WebSpeed are stil consistent with the last test, further proving a possible limit.
Meanwhile Tomcat is just taking slightly longer to respond with data.

70kb @ 100ms (1000 Users, 10 Agents)



And we’re back to our largest packet size with a moderate delay on the server. Nobody likes this much.

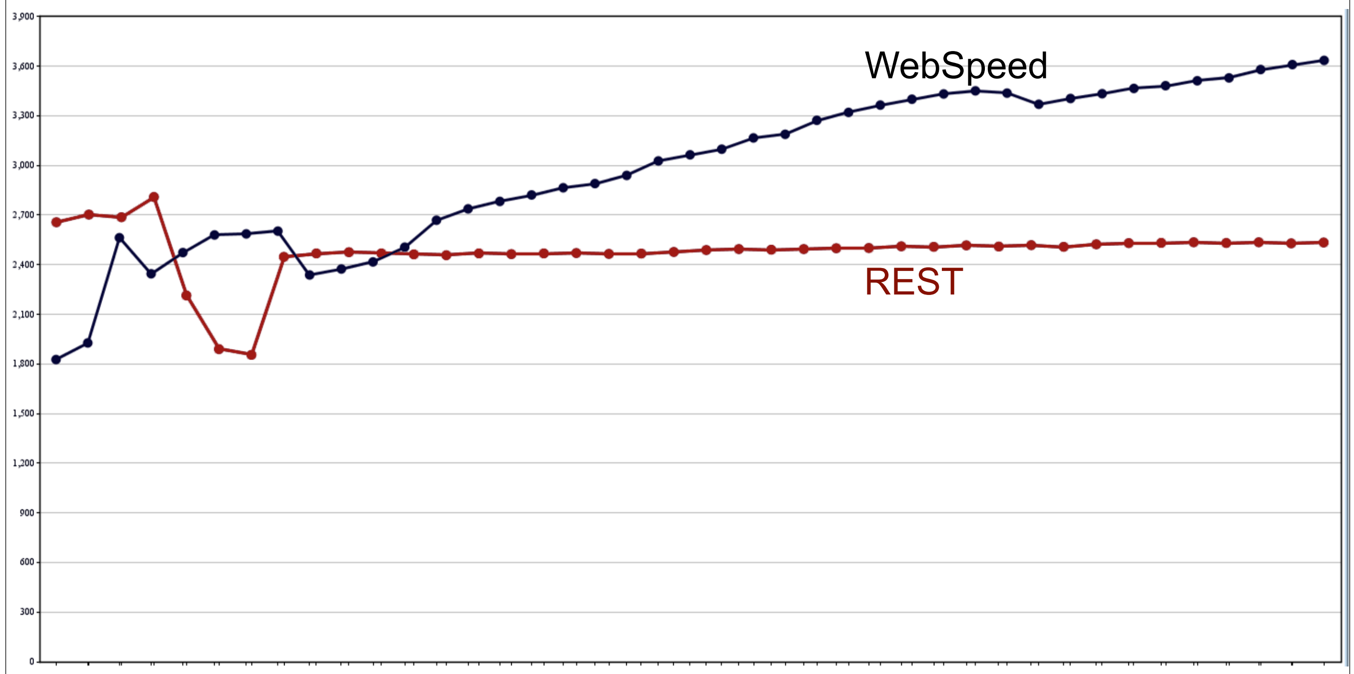
1kb @ 200ms (1000 Users, 10 Agents)



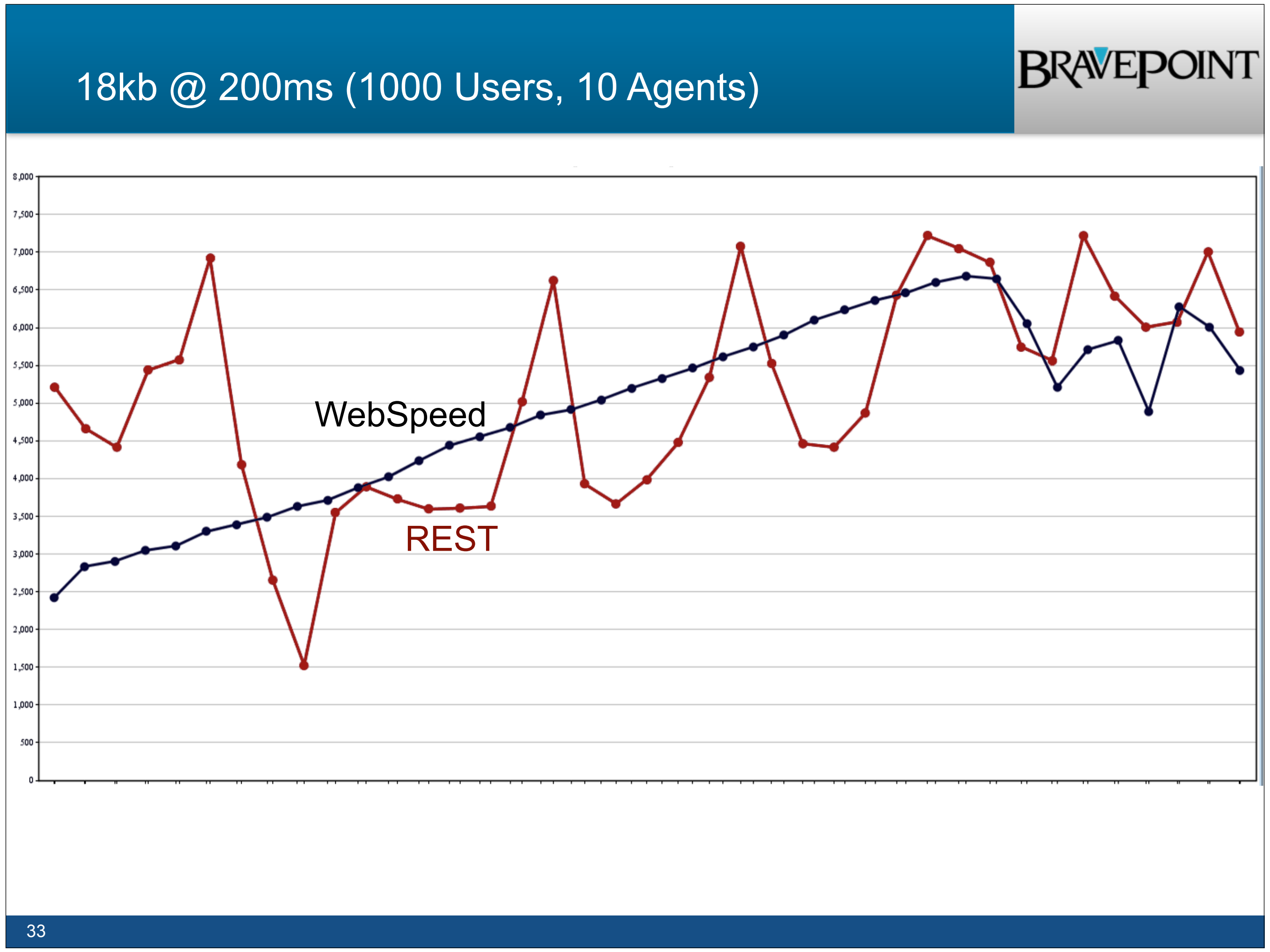
Interesting. Increasing the time to return data from the brokers means little to your HTTP daemon.
At this point the bottleneck is at the agents, which are busy. Requests are waiting for the next available.
I ran this benchmark numerous times, all resulting in the same jagged appearance.

4kb @ 200ms (1000 Users, 10 Agents)

BRAVEPOINT



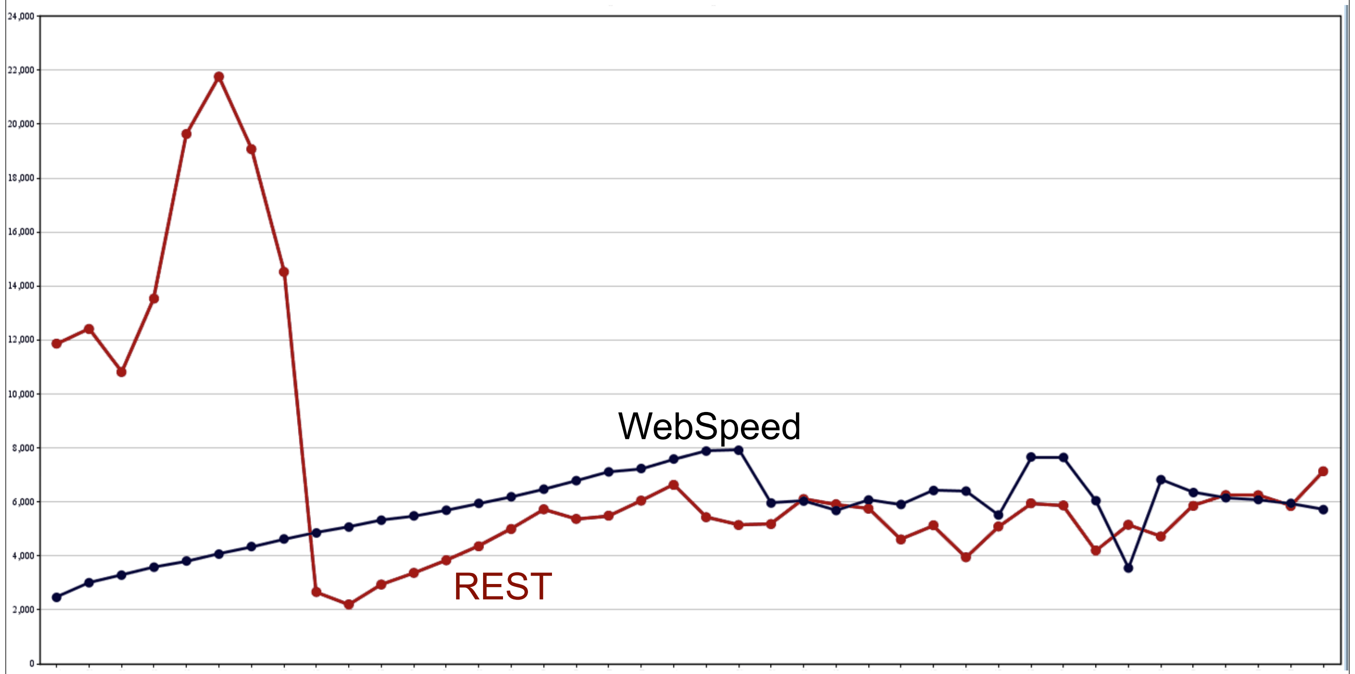
The longer your query, the worse the response overall. Packet size means little at this point.



18kb @ 200ms (1000 Users, 10 Agents)

Ouch. This should be the worst case you should experience…

70kb @ 200ms (1000 Users, 10 Agents)



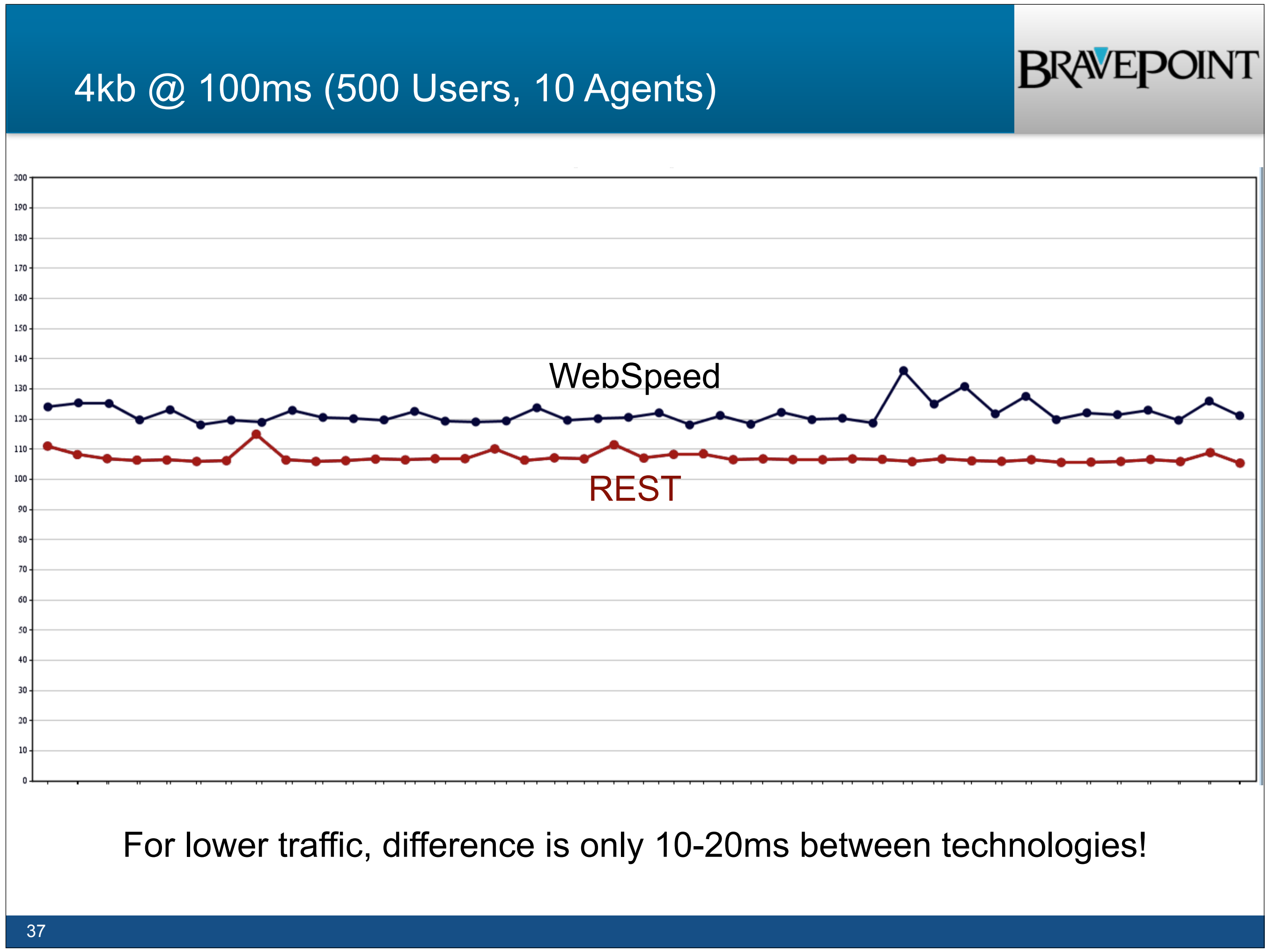
Ok, maybe this is the worst. That's a minimum 2 second wait before we even start to see data.
At this point everyone is waiting on the next available agent.

Number Crunching

- How many requests can we handle without error?
  - Depends on your hardware/VM configuration
  - Possibly dependent on number of agents
- What makes the biggest difference, if any?
  - Concurrency: agents can only handle so many requests
  - Payload: puts a bottleneck at the network, HTTP daemon
  - Processing: puts the bottleneck at the OE broker’s agent

Observations

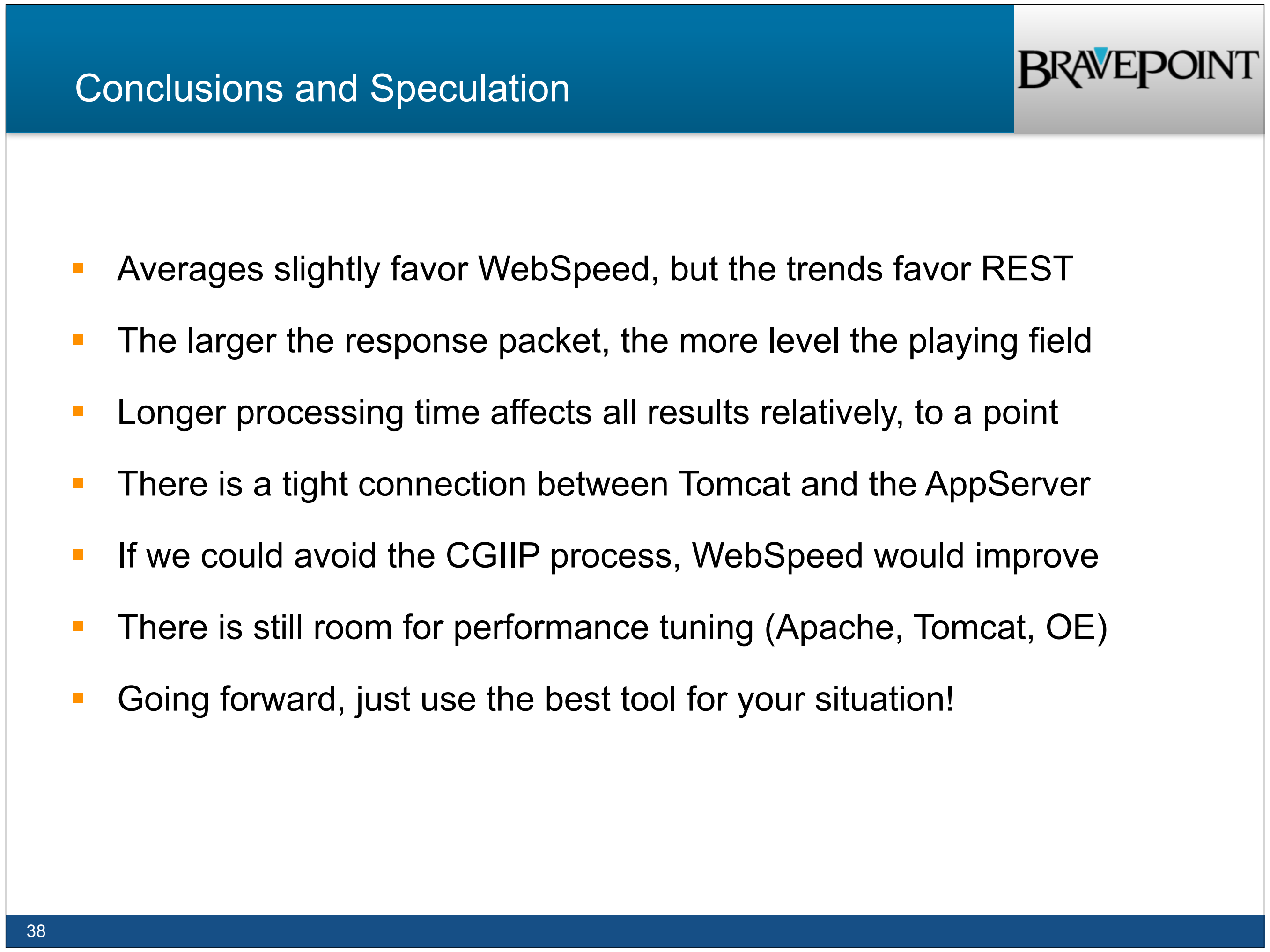
- There is a tradeoff in processing of requests
  - Tomcat: few java.exe processes, many threads
  - WebSpeed: many cgiip.exe processes, few threads
- Tomcat seems to lag with initial requests but levels off
  - Java memory increases over time, subject to GC
- WebSpeed remains relatively consistent under most loads
  - Takes time to fork and exec each CGIIP process
- Not shown: Error % increases with payload size and latency
- Requests/sec and KB/sec were very similar between servers
- Fewer requests show even less drastic results…



4kb @ 100ms (500 Users, 10 Agents)

For lower traffic, difference is only 10-20ms between technologies!

The previous tests are still relevant. They show that from a cold start there are irregularities with the results.



Conclusions and Speculation

BRAVEPOINT

- Averages slightly favor WebSpeed, but the trends favor REST
- The larger the response packet, the more level the playing field
- Longer processing time affects all results relatively, to a point
- There is a tight connection between Tomcat and the AppServer
- If we could avoid the CGIIP process, WebSpeed would improve
- There is still room for performance tuning (Apache, Tomcat, OE)
- Going forward, just use the best tool for your situation!

Configuration time for REST is significantly greater than WebSpeed.
So consider that if you want things up and running quickly.

Thank You!

Please direct any angry emails or complaints about omissions via /dev/null :)

BRAVEPOINT

Dustin Grau

dgrau@bravepoint.com