



Extracting Performance and Scalability Metrics From TCP

Baron Schwartz
Surge Conference
September 30, 2011



**Consulting
Support
Training
Development**

For MySQL

October 24-25, London

www.percona.com/live



PERCONA
LIVE

Agenda

- Capturing TCP Traffic
- Time-Series Plotting
- Stall Detection
- Detecting Performance Problems
- Modeling Scalability
- Forecasting Performance
- Validating Input
- When Is This Useful?

My Background & Perspective

- *You cannot optimize what you cannot measure.*
- I spend my life finding ways to make performance better and more predictable.
 - Improving quality of service.
 - Eliminating outliers.
 - Reducing load.
 - Lowering latency.
- I observe, explain, and solve performance problems.

Observe first.

Then explain.

Then the solution will usually be obvious.

Observing First

- Do you know everything there is to know about your system's performance?
- How can you learn more, easily and cheaply?

What's Great About TCP?

- Call/response protocol semantics
- Fundamental metrics of performance:
 - Arrival time
 - Completion time
- From this we can derive:
 - Observation interval
 - Queries per second
 - Busy time
 - Total execution time

More Metrics

- We can use Little's Law...
 - $N=XR$ (concurrency = throughput * response time)
- To further derive:
 - Concurrency
 - Utilization

Capturing TCP Traffic

```
tcpdump -s 384 -i any -nnq -tttt  
'tcp port 3306 and (((ip[2:2] - ((ip[0]&0xf)<<2))  
- ((tcp[12]&0xf0)>>2)) != 0)' > tcp-file.txt
```

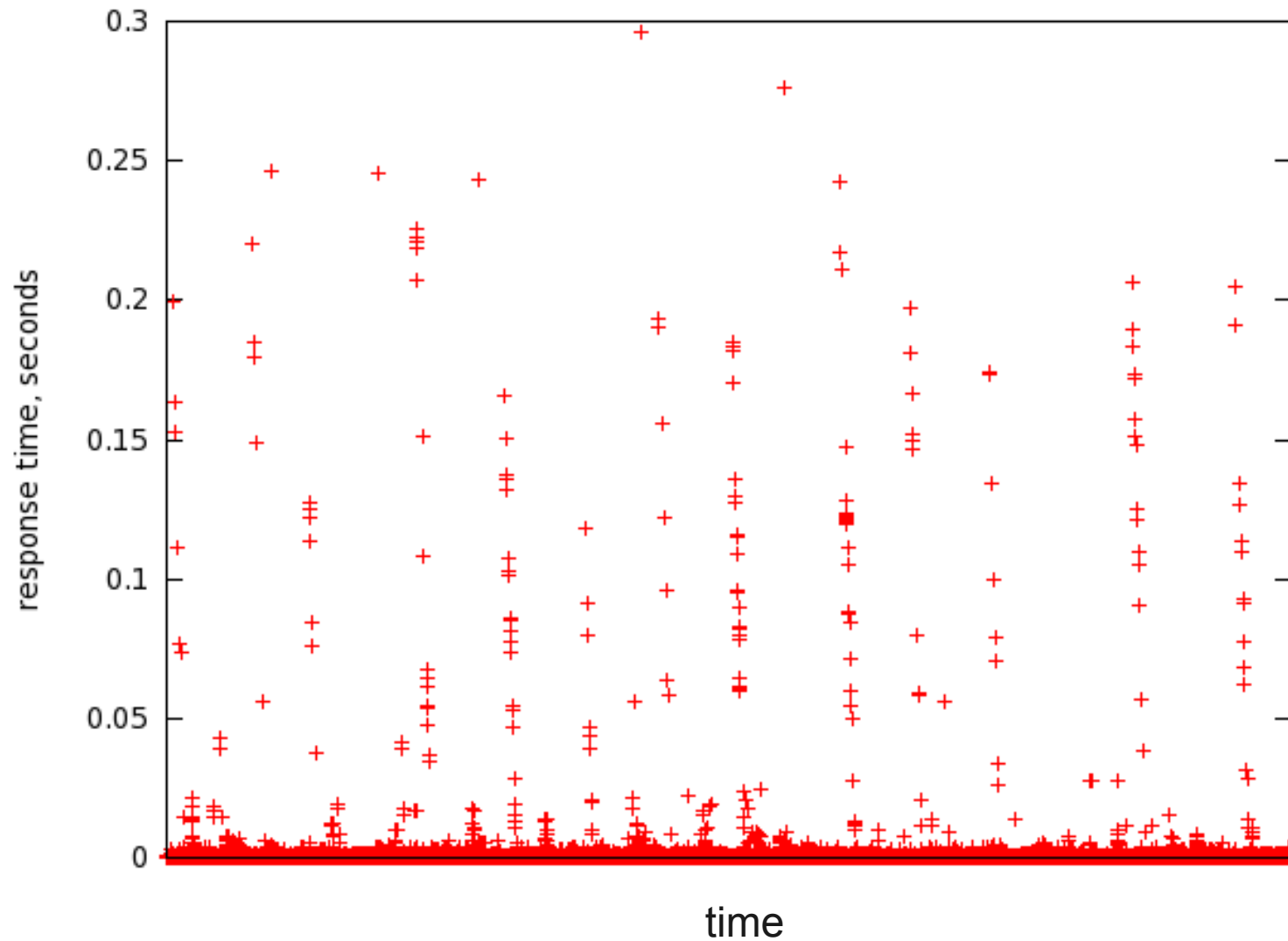
```
2011-05-05 10:47:17.810932 IP 10.220.146.79.35805 > 10.119.42.41.3306: tcp 83  
2011-05-05 10:47:17.811021 IP 10.119.42.41.3306 > 10.220.146.79.35805: tcp 64  
2011-05-05 10:47:17.811545 IP 10.250.95.31.45400 > 10.119.42.41.3306: tcp 82
```

Process with pt-tcp-model

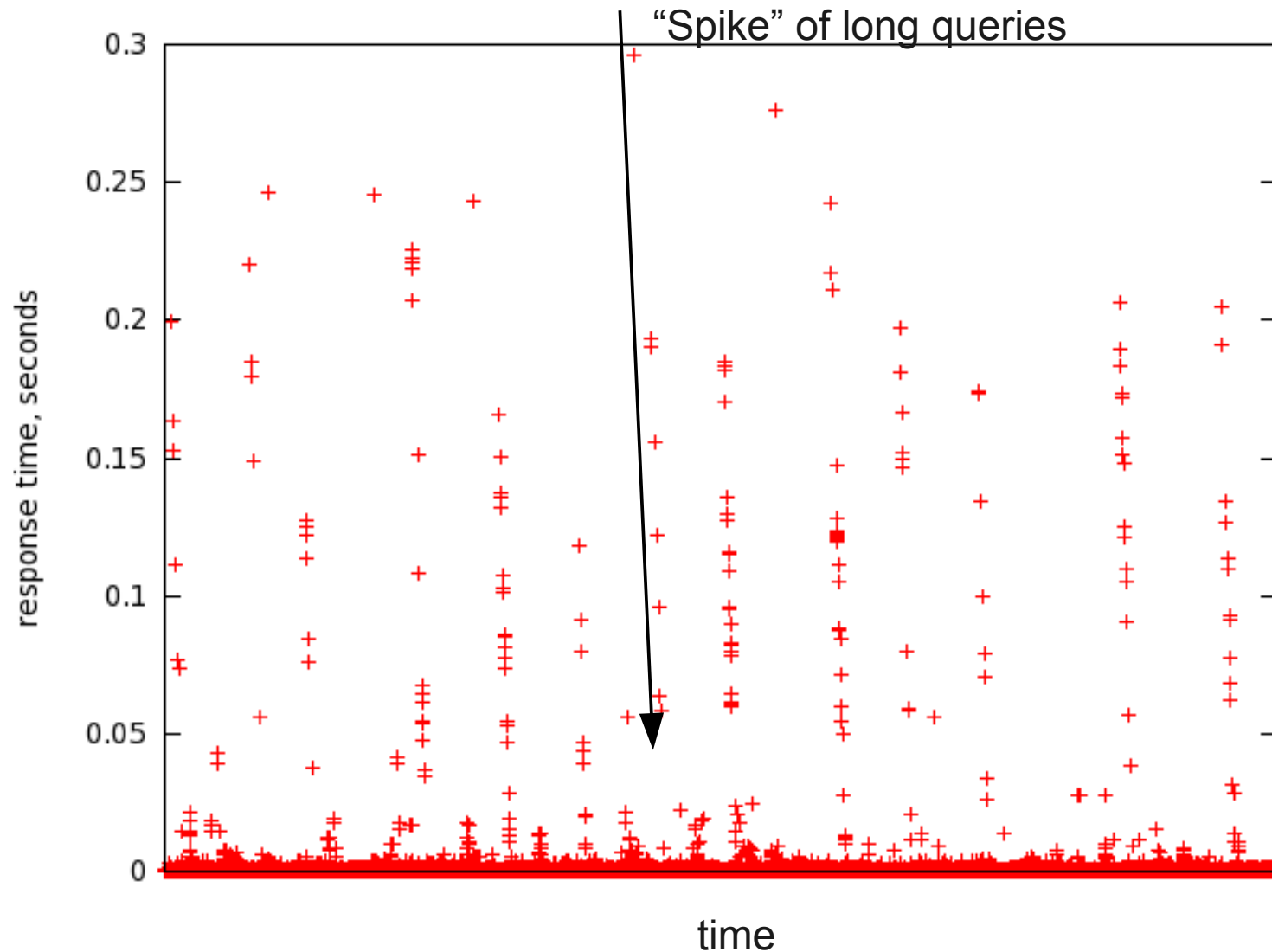
```
pt-tcp-model tcp-file.txt > requests.txt
```

#	start	end	elapsed	host:port
0	1304606837.810932	1304606837.811021	0.000089	10.220.146.79:35805
1	1304606837.811545	1304606837.811778	0.000233	10.250.95.31:45400
2	1304606837.811669	1304606837.811971	0.000302	10.243.78.239:45612
3	1304606837.811893	1304606837.812073	0.000180	10.222.110.47:44024
4	1304606837.813067	1304606837.813312	0.000245	10.220.146.79:35805

Plot as Time-Series and Inspect



Plot as Time-Series and Inspect



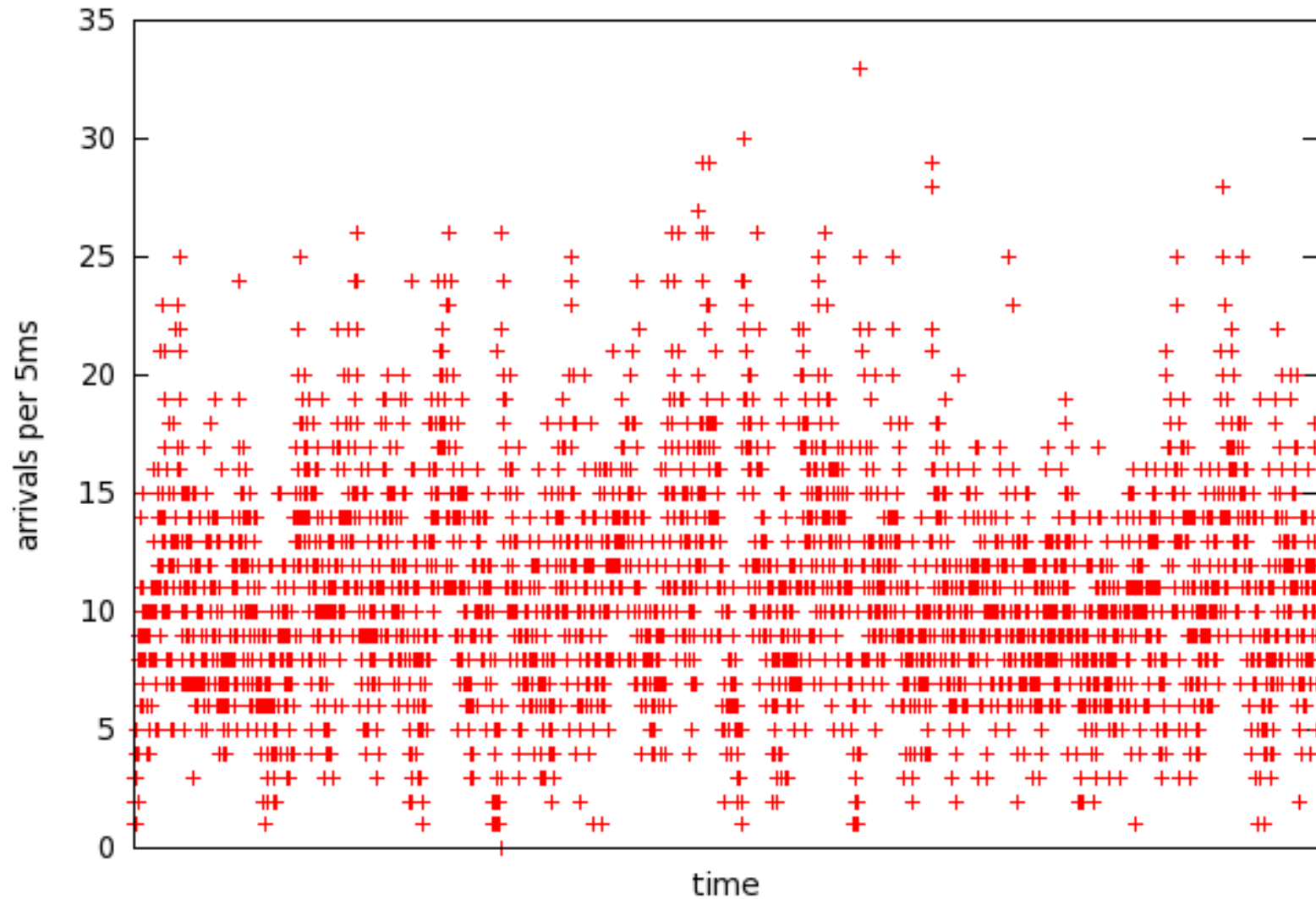
Stall Detection

- What's happening there?
- Each query completes when the previous one releases the resource.
- A long query makes incoming queries queue.

(I happen to know it's `SELECT FOR UPDATE`).

If completions are what's affected,
maybe we can see the evidence.

Arrivals per 5ms



Hmmmm

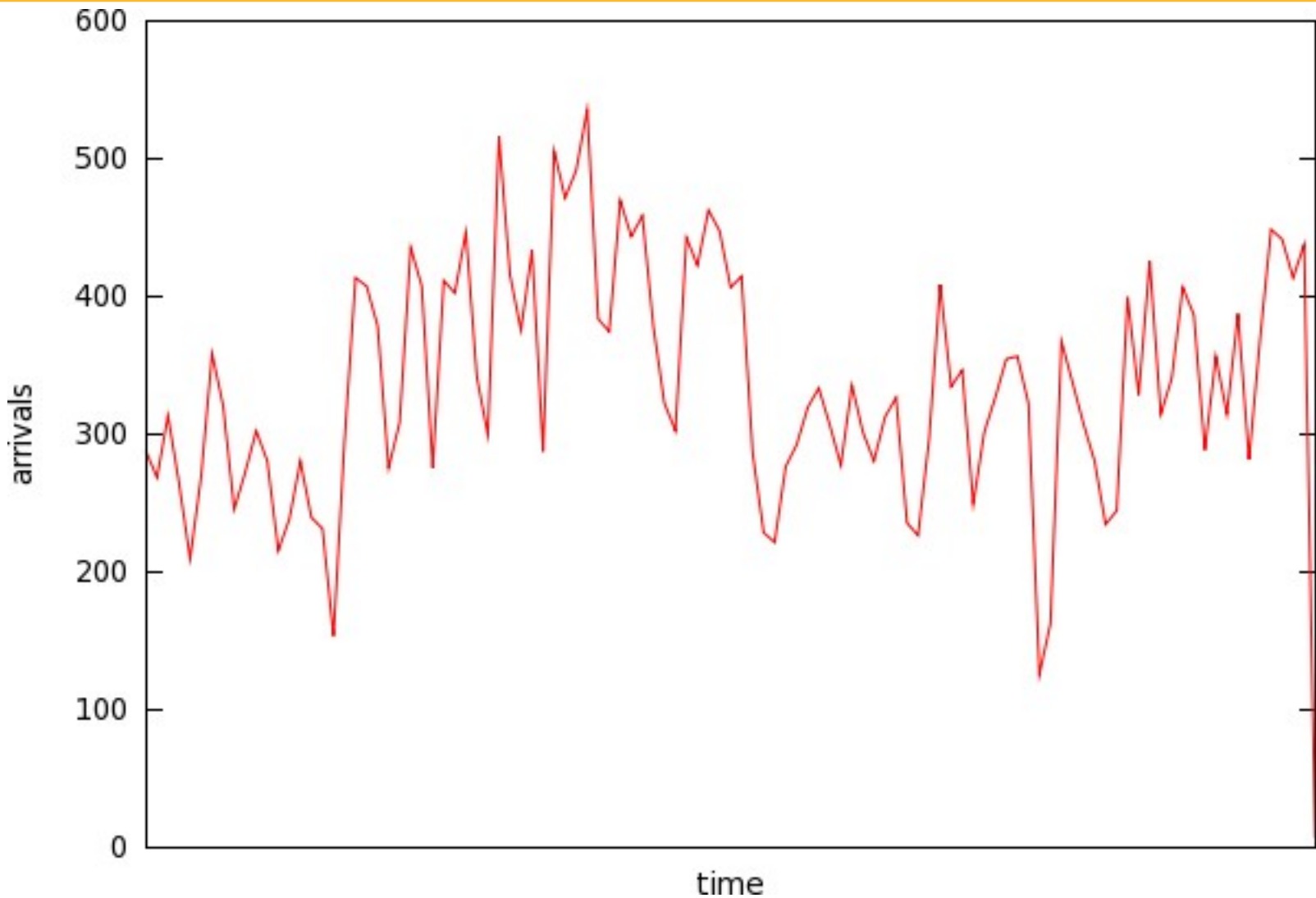
It's kind of hard to see what's going on in that graph. Is 5 ms too fine granularity?

Is there really a problem?

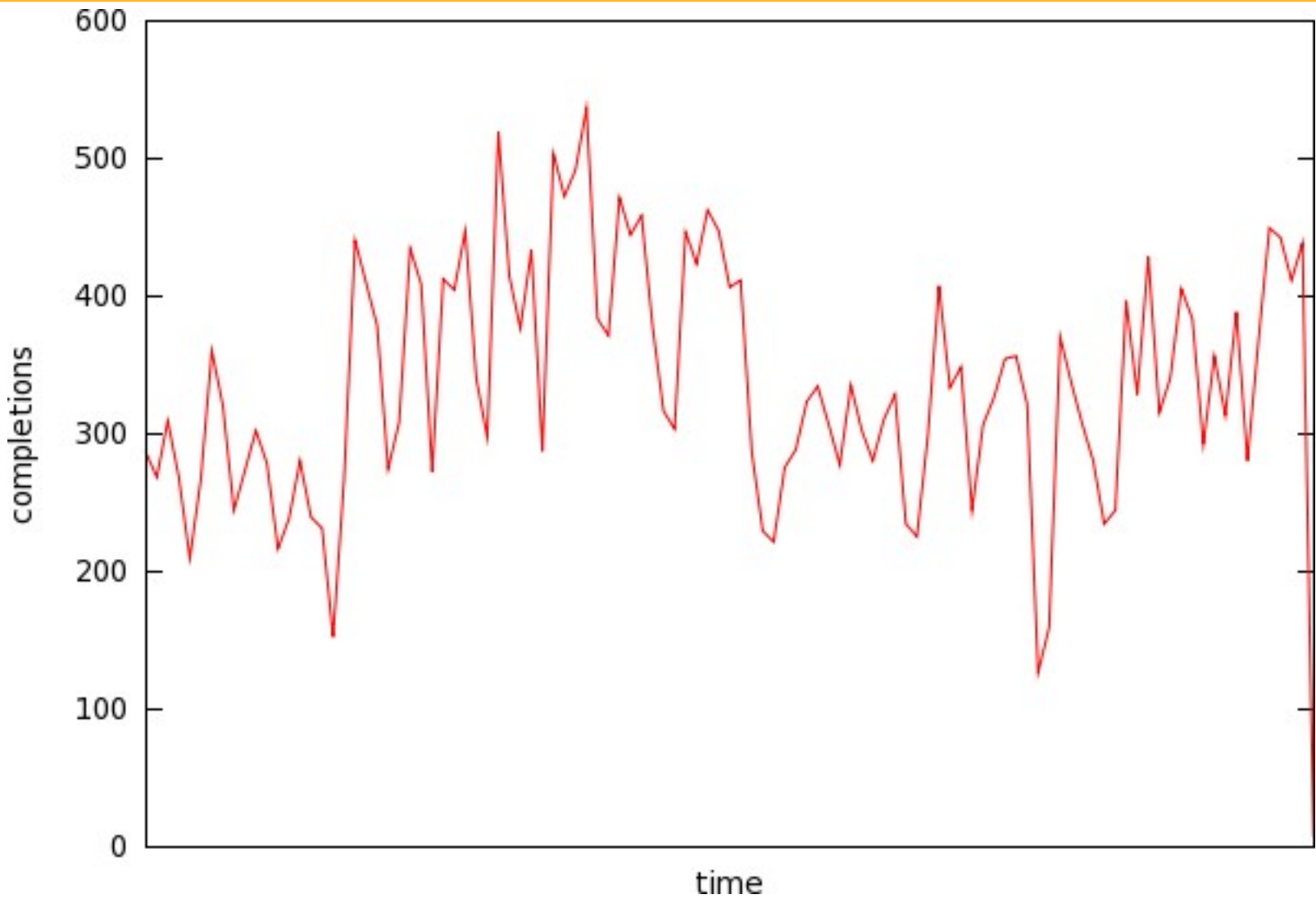
If arrivals are fairly uniform, but completions cluster together, can we see it?

The following graphs are per 2-tenths of a sec.

Arrivals



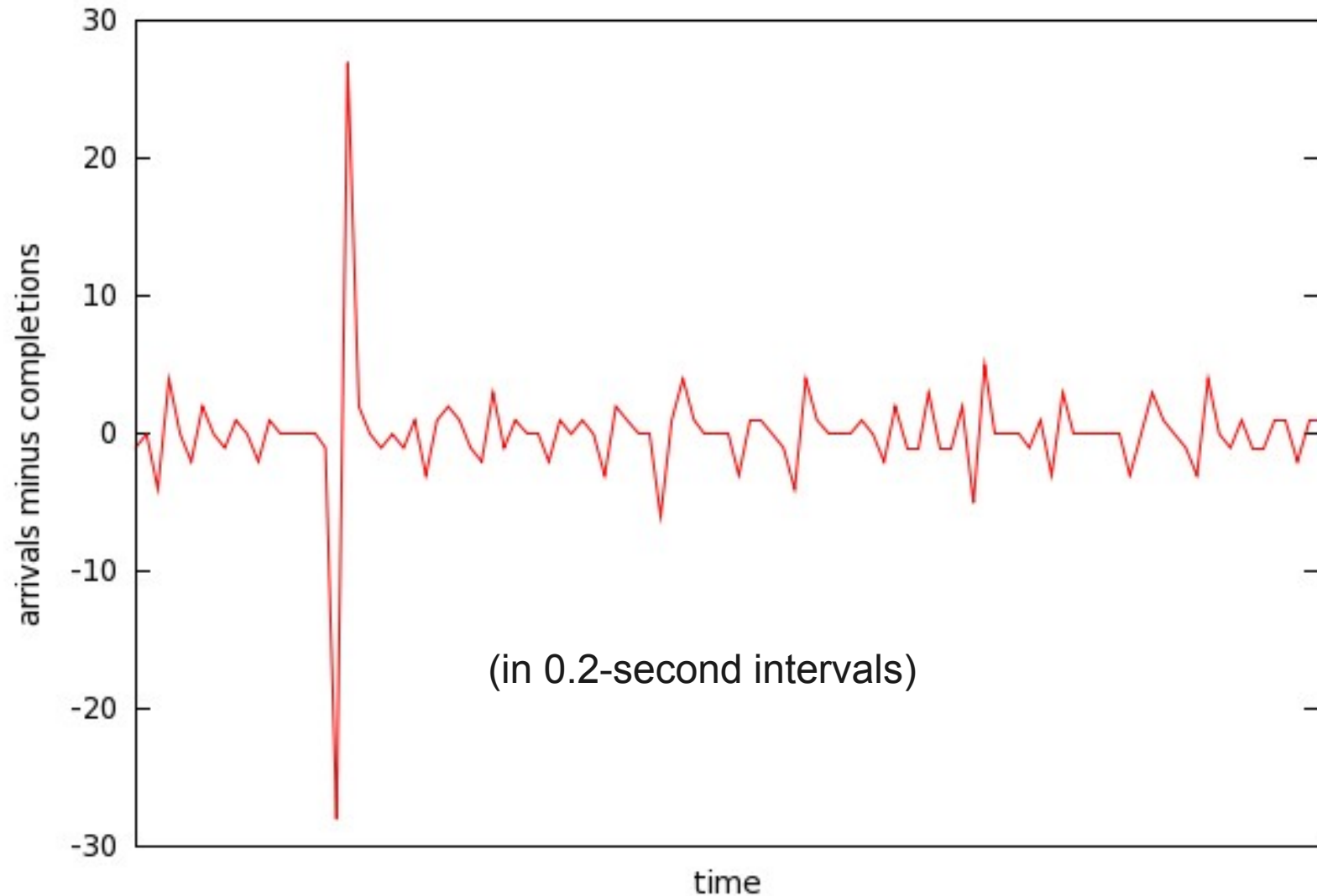
Completions



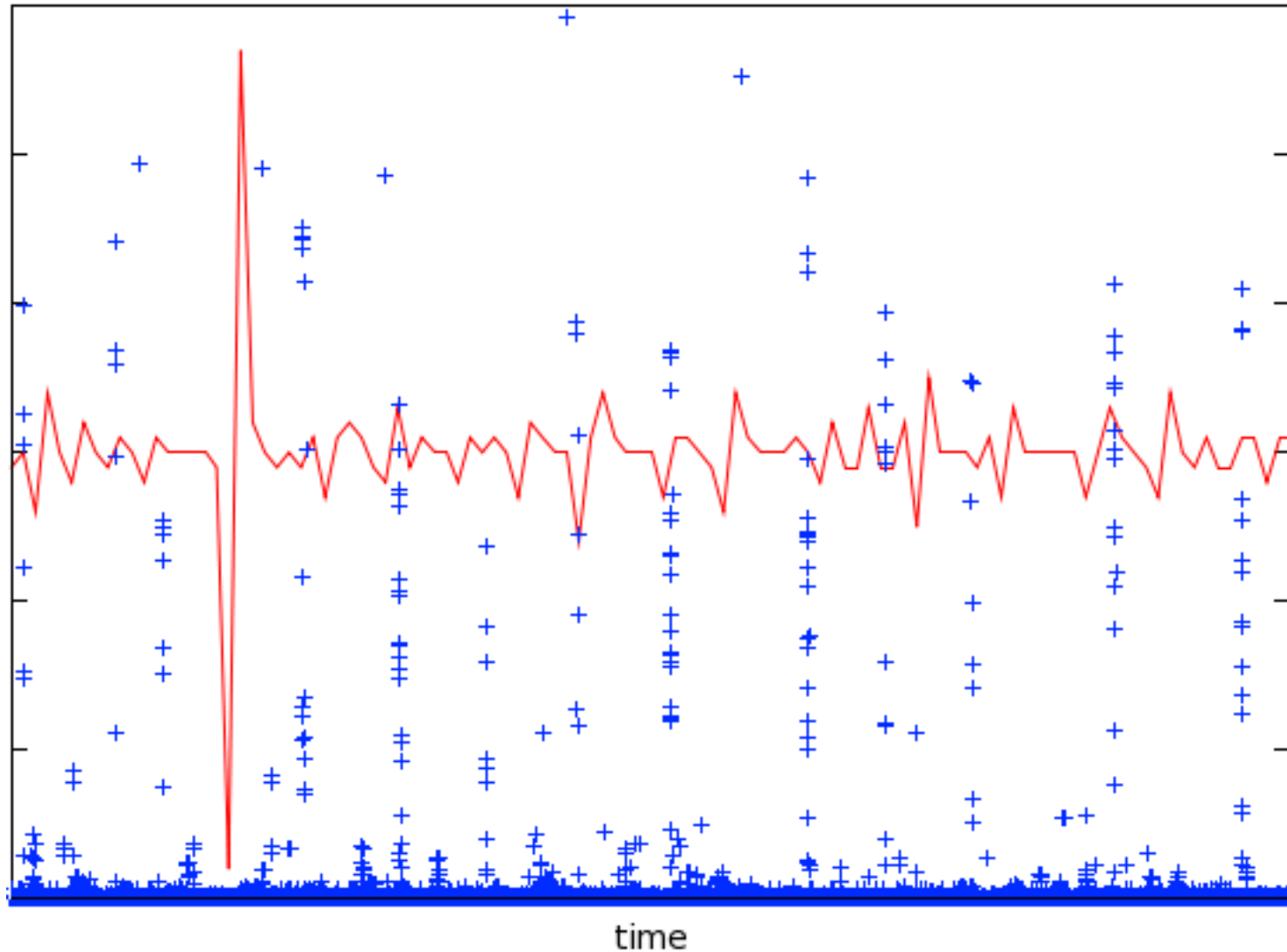
Hmmmmmmmmmmmmmmmmmm

- Looks almost identical. Is my theory wrong?
- What if we plot completions minus arrivals?

Completions Minus Arrivals



Seeing Hidden Patterns



Pile-Up Detection Algorithm

- Plot completions minus arrivals
- Try finer and finer granularities
- Is a 0.2-second pile-up acceptable?

Detecting Performance Problems

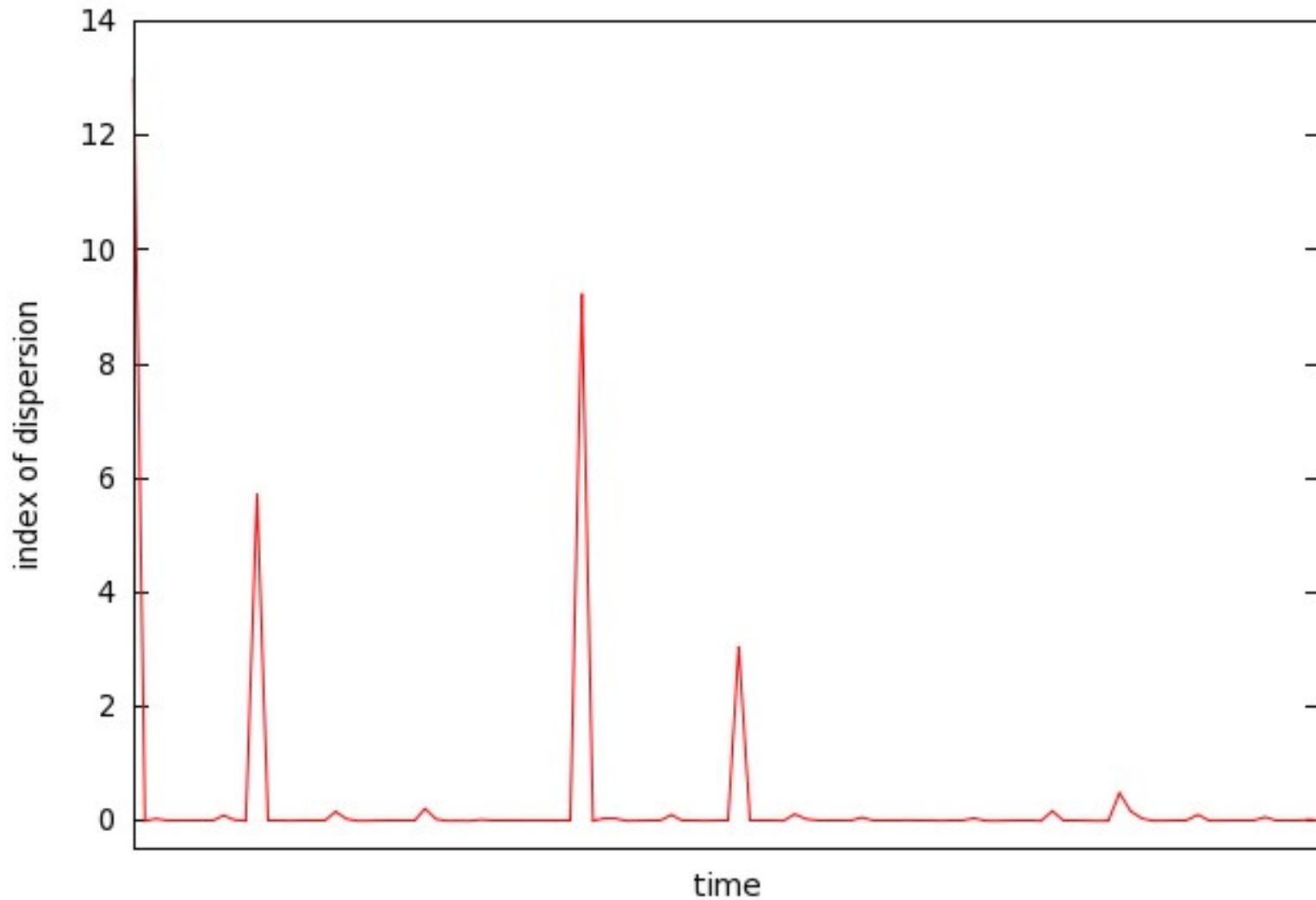
Another good statistic is the index of dispersion of response times.

$$\frac{\text{Variance}}{\text{Mean}}$$

Index of Dispersion

- Normalized relative to average response time.
- Lets you compare different workloads.

Index of Dispersion

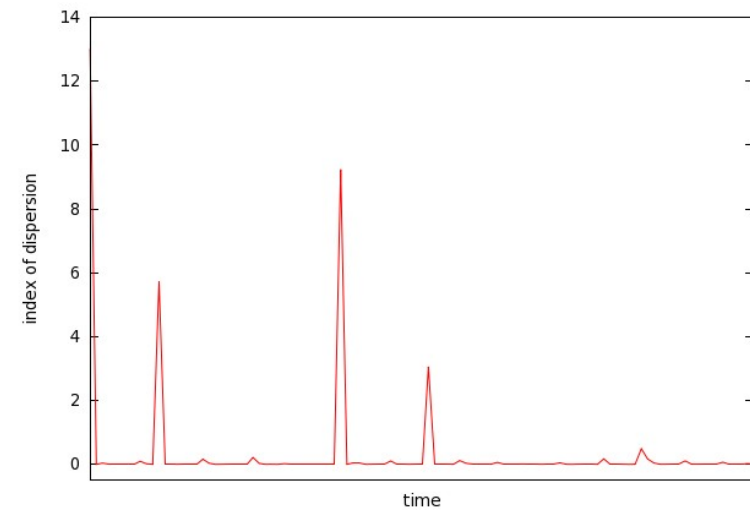
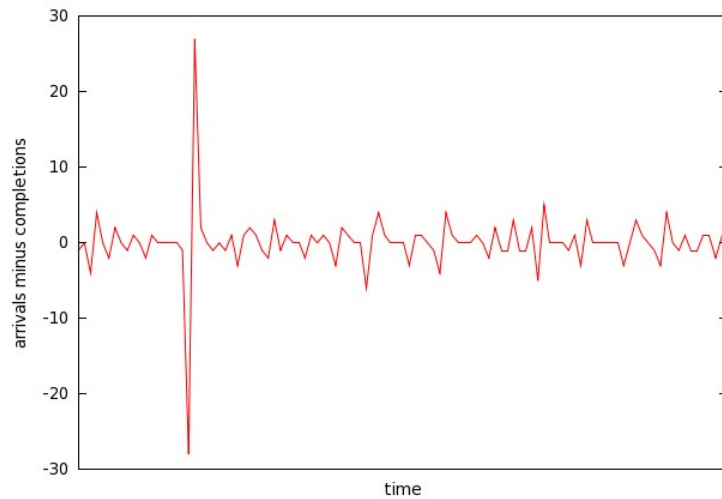
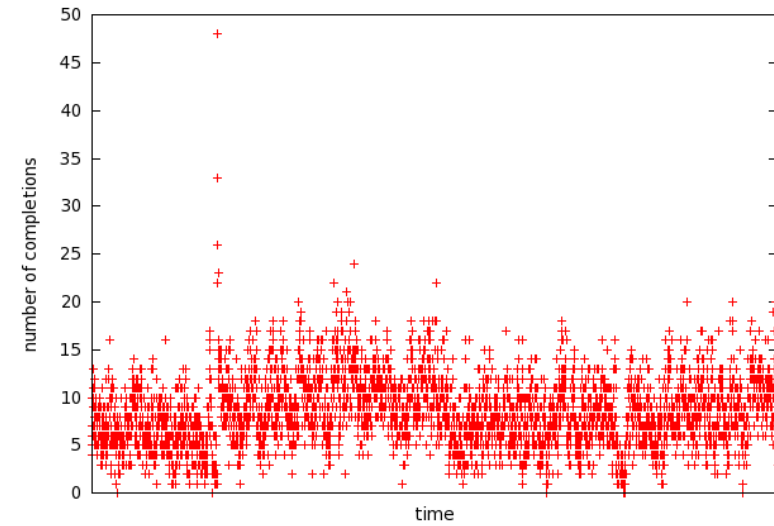
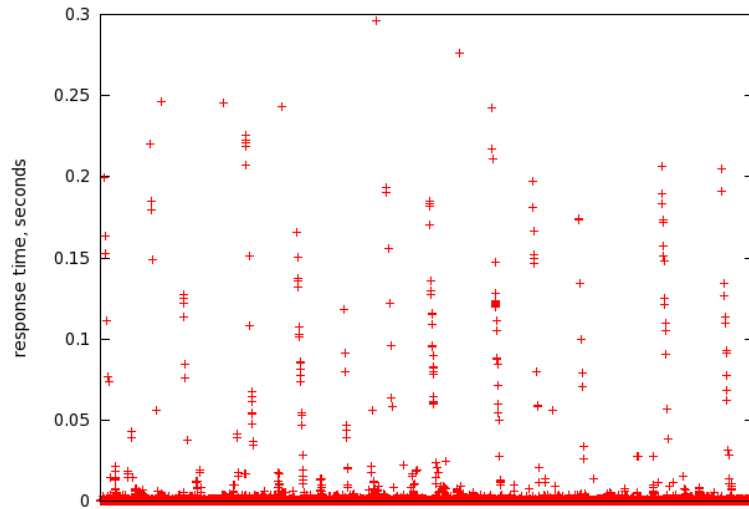


Interpreting Index of Dispersion

- What does it MEAN when there's a spike?
- “This time period is highly variable.”
- “Highly variable performance is bad.”

Highly variable == highly optimizable.

Time-Series, All Together Now



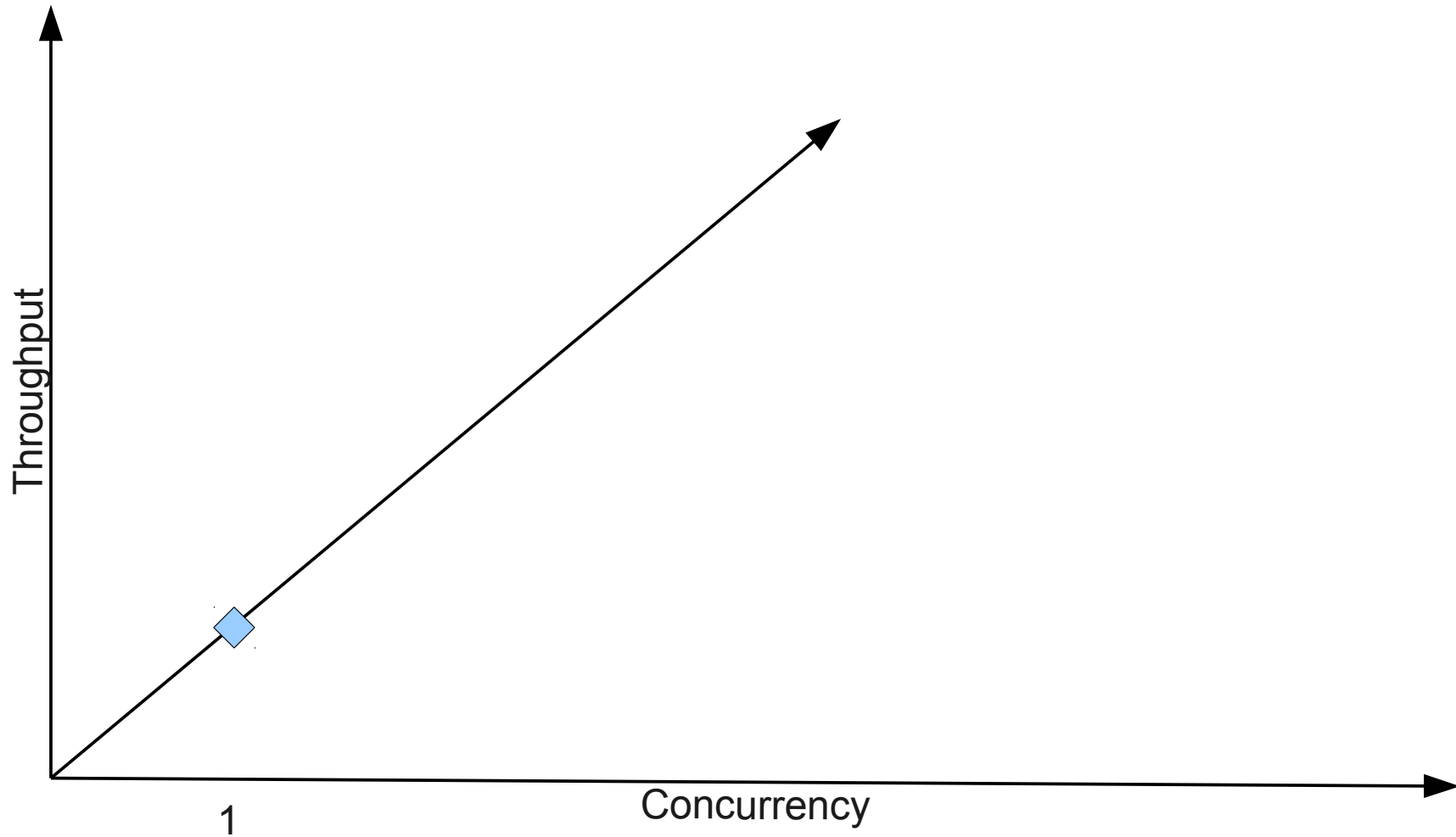
Scalability and Performance (the mathematical version)

What is scalability?

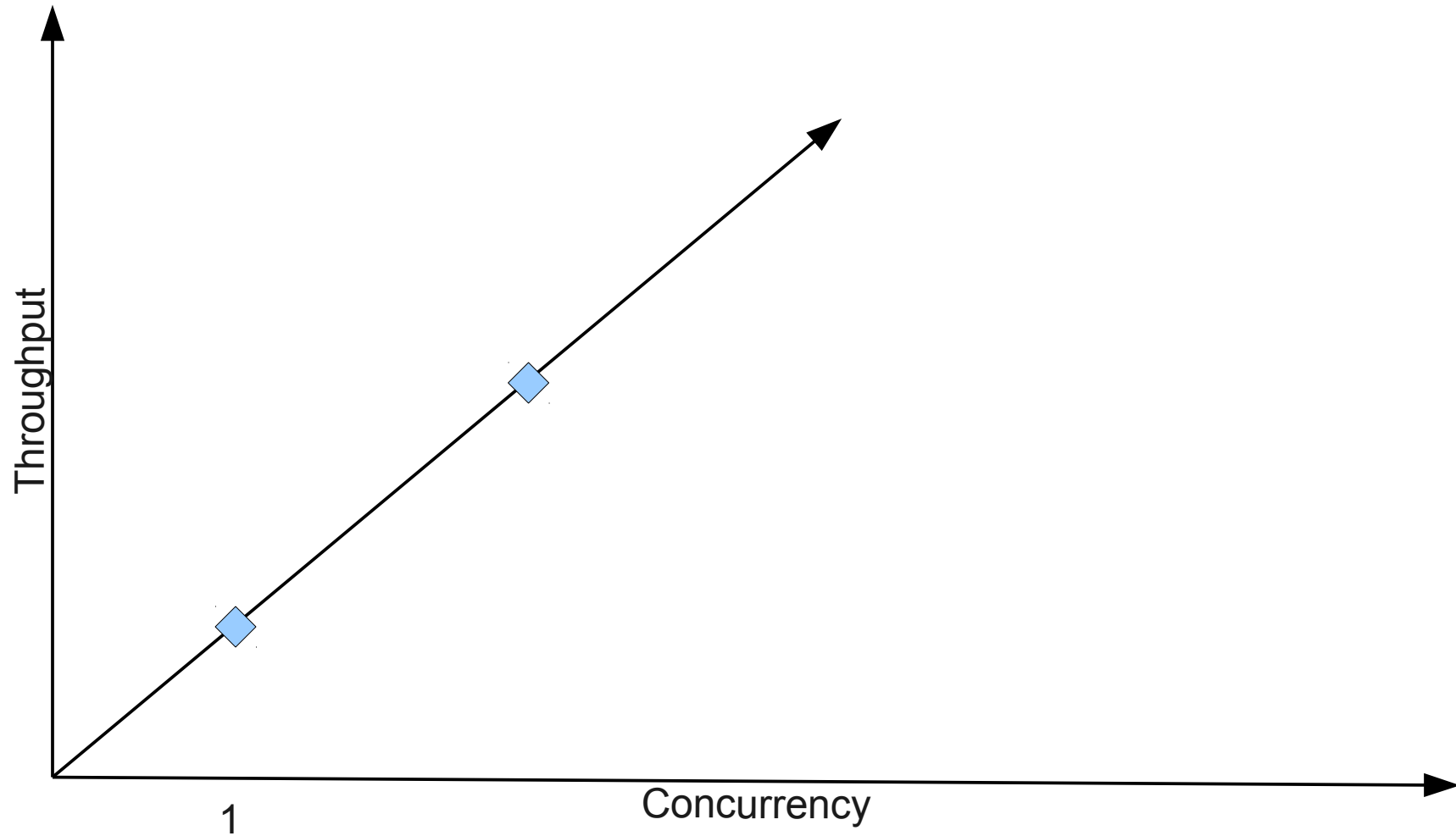
Modeling Scalability

- Scalability is a function (equation)
- The X-axis is Concurrency
- The Y-axis is Throughput

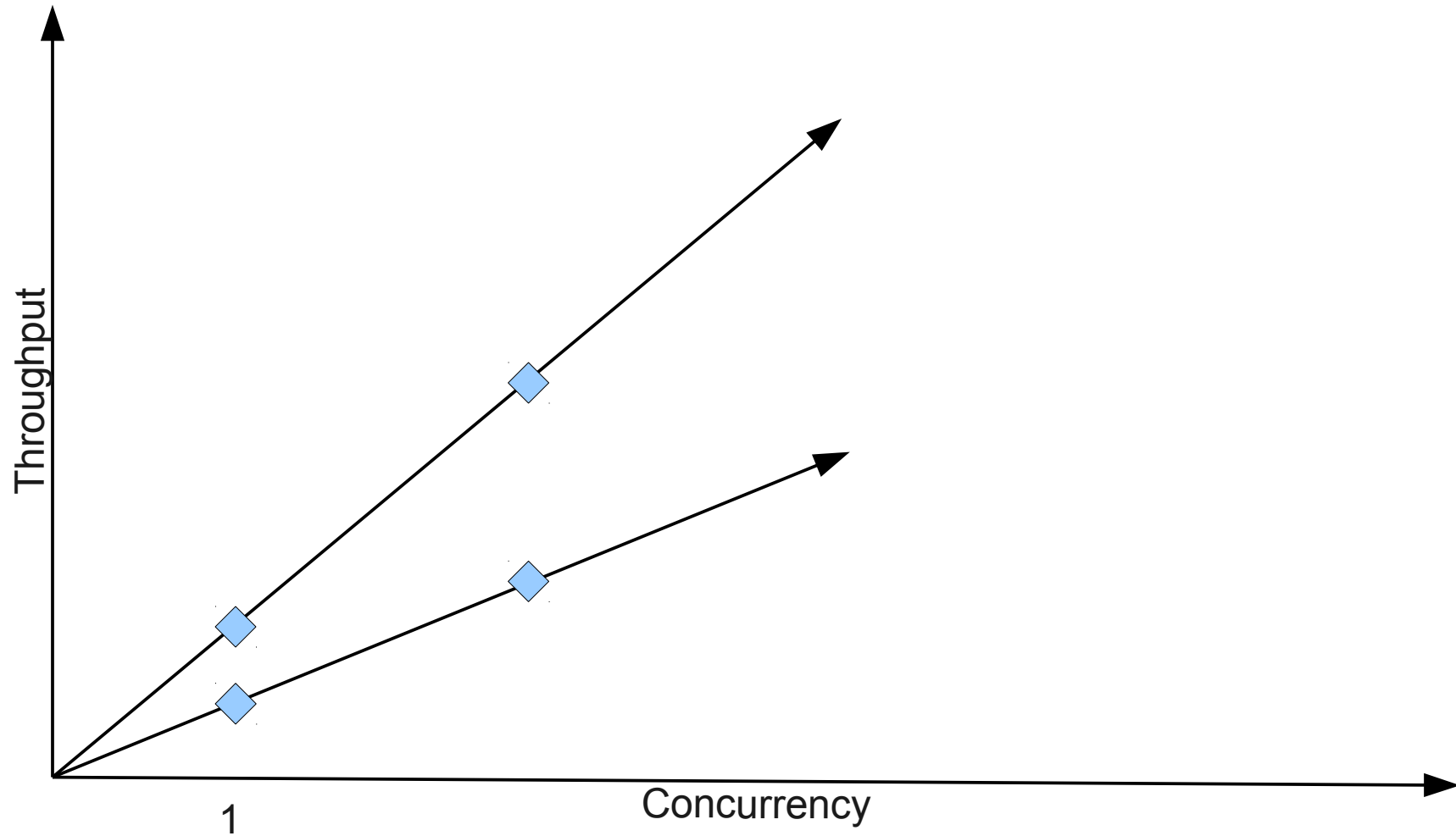
The Scalability Function



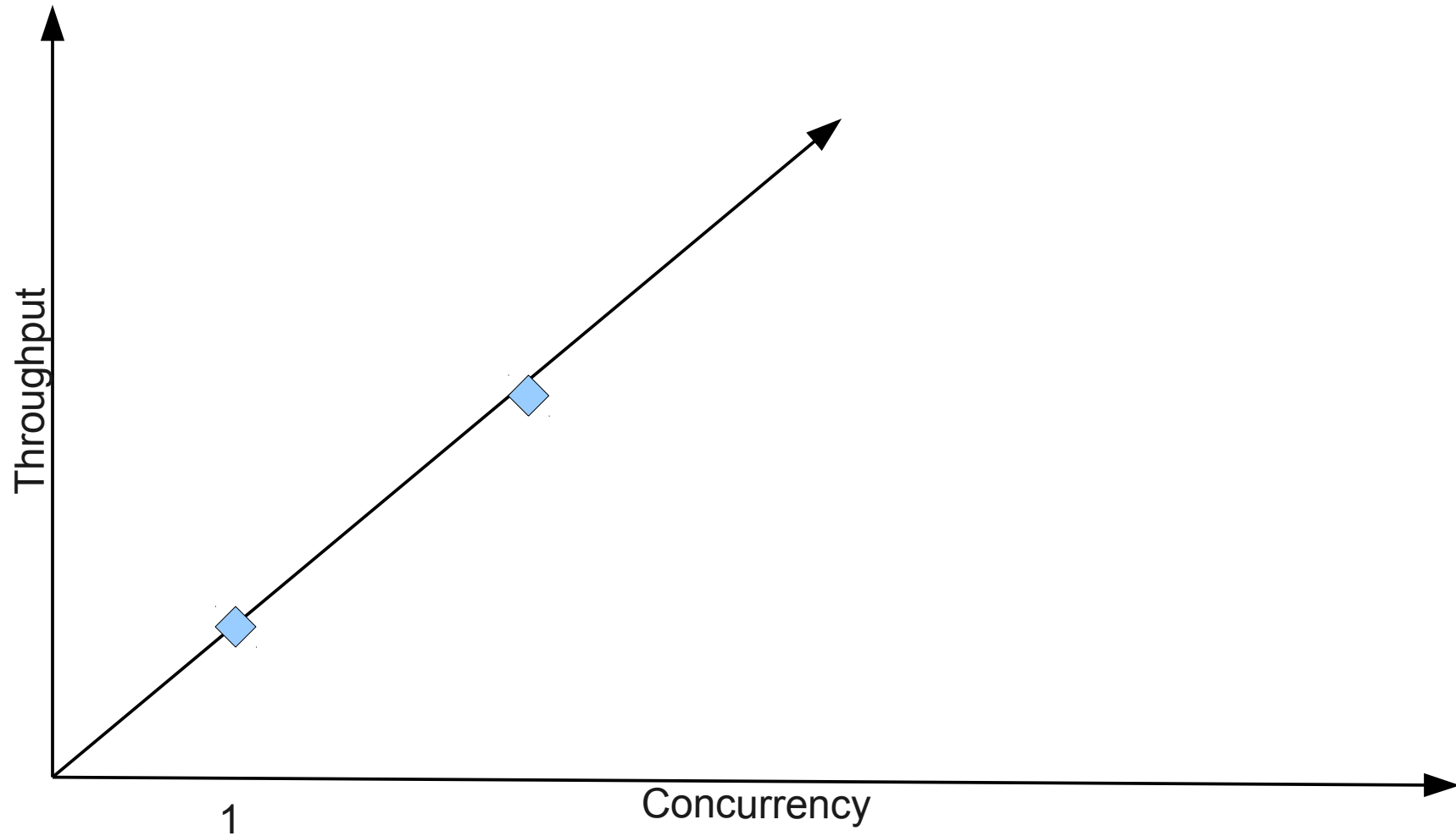
This is Linear Scalability



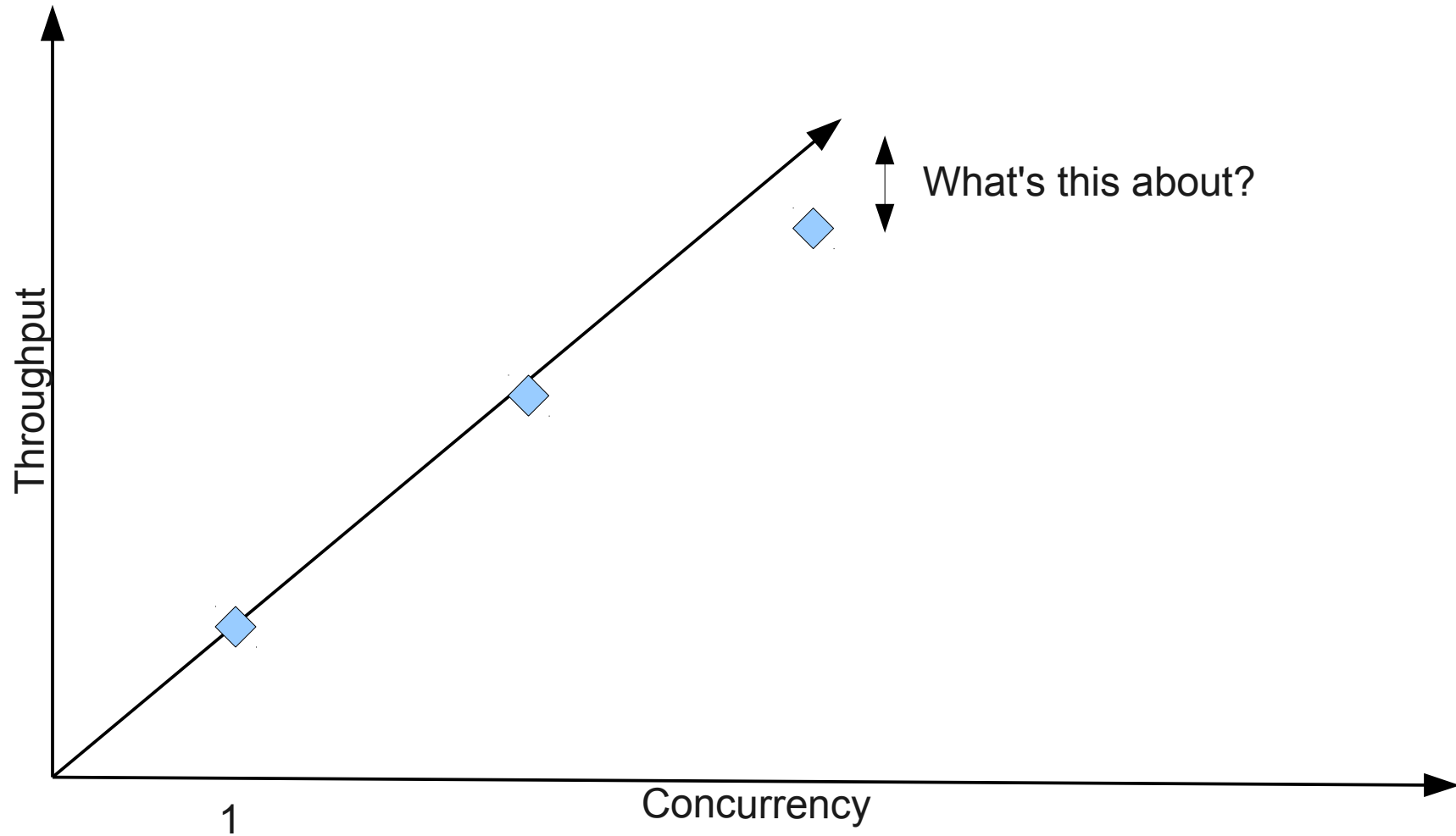
This is Also Linear Scalability



This is Not Linear Scalability



What Causes Non-Linearity?



Factor #1: Serialization

- Some portion of the work cannot be done in parallel
- This is Amdahl's Law

$$C(N) = \frac{N}{1 + \sigma(N - 1)}$$

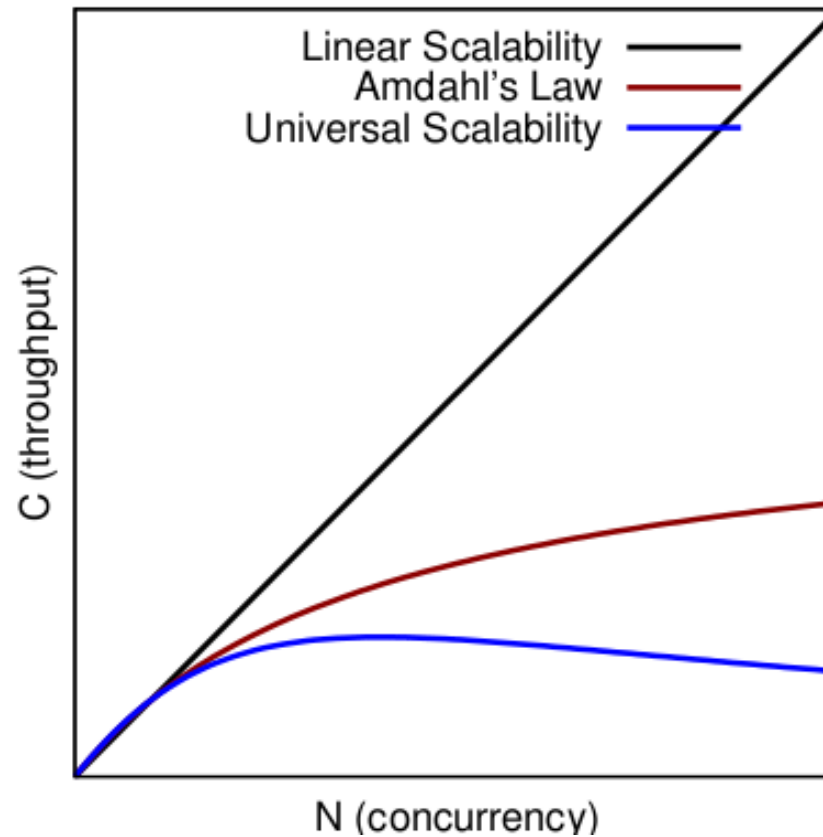
Factor #2: Coherency

- Some portion of the work relies on IPC, cross-node communication, etc
- Dr. Neil Gunther's University Scalability Law:

$$C(N) = \frac{N}{1 + \sigma(N - 1) + \kappa N(N - 1)}$$

Loss of Scalability

Most systems have serialization & coherency.



Scalability Modeling Method

- Measure Throughput and Concurrency.
- Perform a regression against the USL.
(Determine sigma and kappa coefficients)
- ?????
- Profit!

$$C(N) = \frac{N}{1 + \sigma(N - 1) + \kappa N(N - 1)}$$

Concurrency from TCP

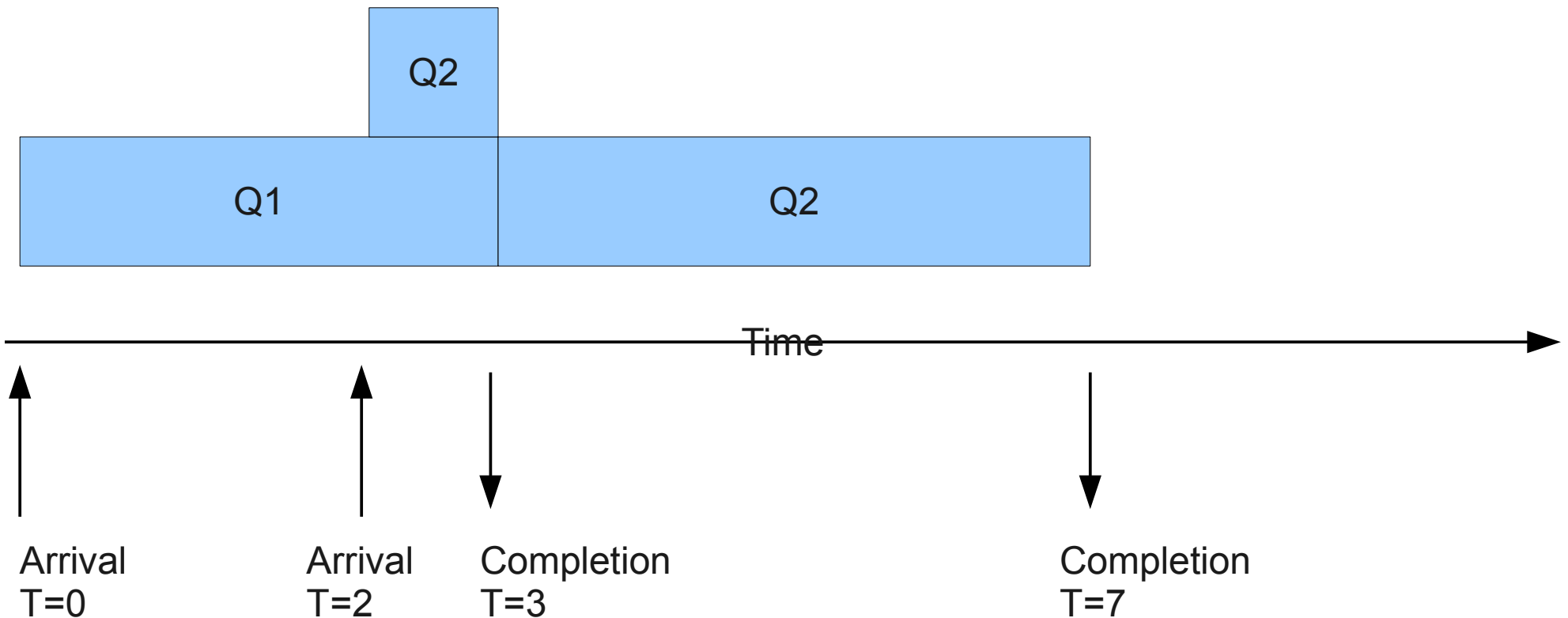
- Throughput is easy (queries per second)
- Concurrency is harder.

Concurrency from TCP

- Sort the arrivals and departures in order.
 - Each arrival increments concurrency.
 - Each departure decrements it.
- Calculate a moving weighted average.

Visually...

Observation Time: 7
Total Query Time: 8
Average Concurrency: 8/7



Using pt-tcp-model

- pt-tcp-model knows how to compute this.

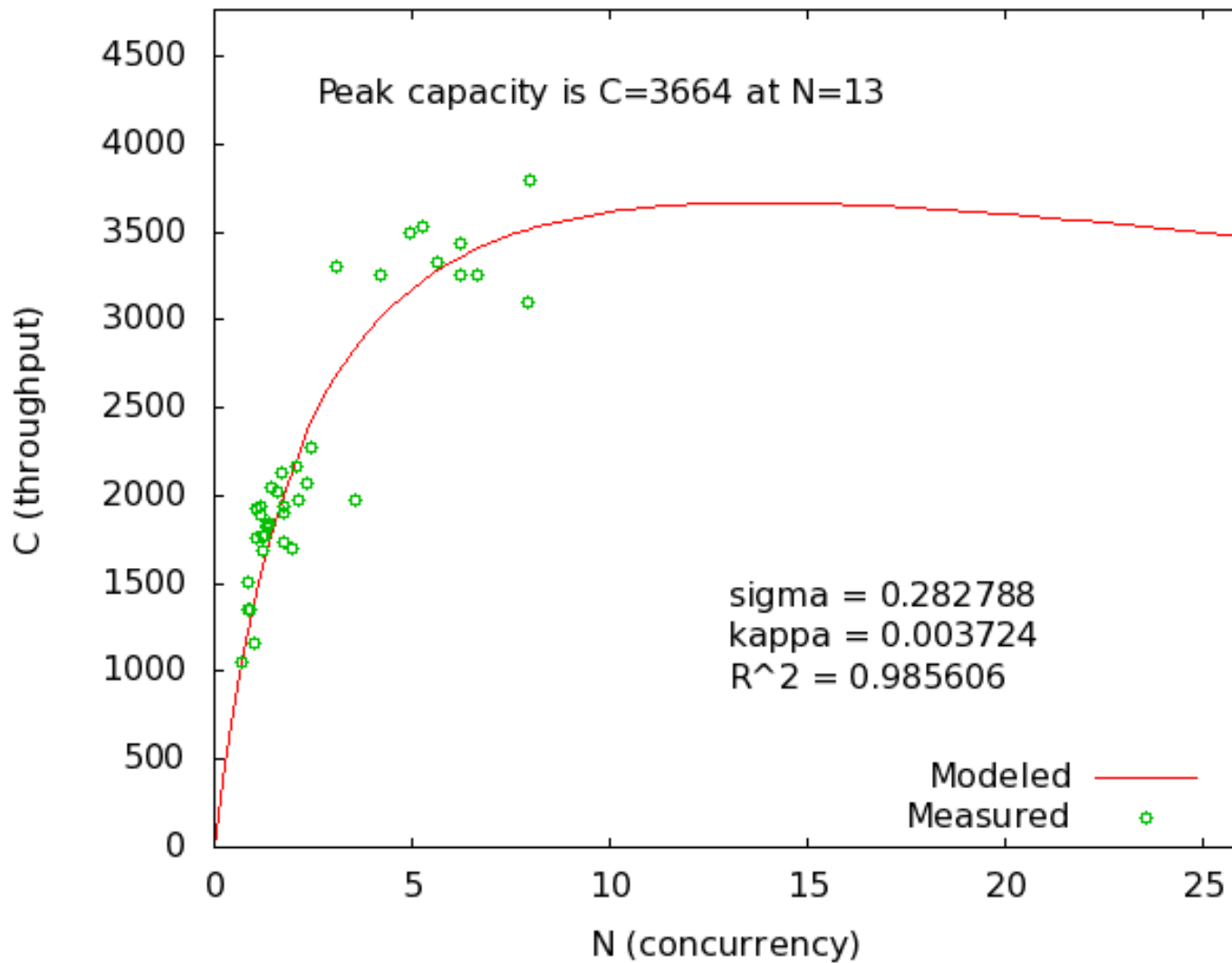
```
sort -n -k1,1 requests.txt > sorted.txt
```

```
pt-tcp-model --type=requests sorted.txt > sliced.txt
```

Determine the Coefficients

- Use *R*, *gnuplot* or other statistical tools to fit the model to the data and derive:
 - The coefficient of serialization (σ)
 - The coefficient of coherency (κ)

Results

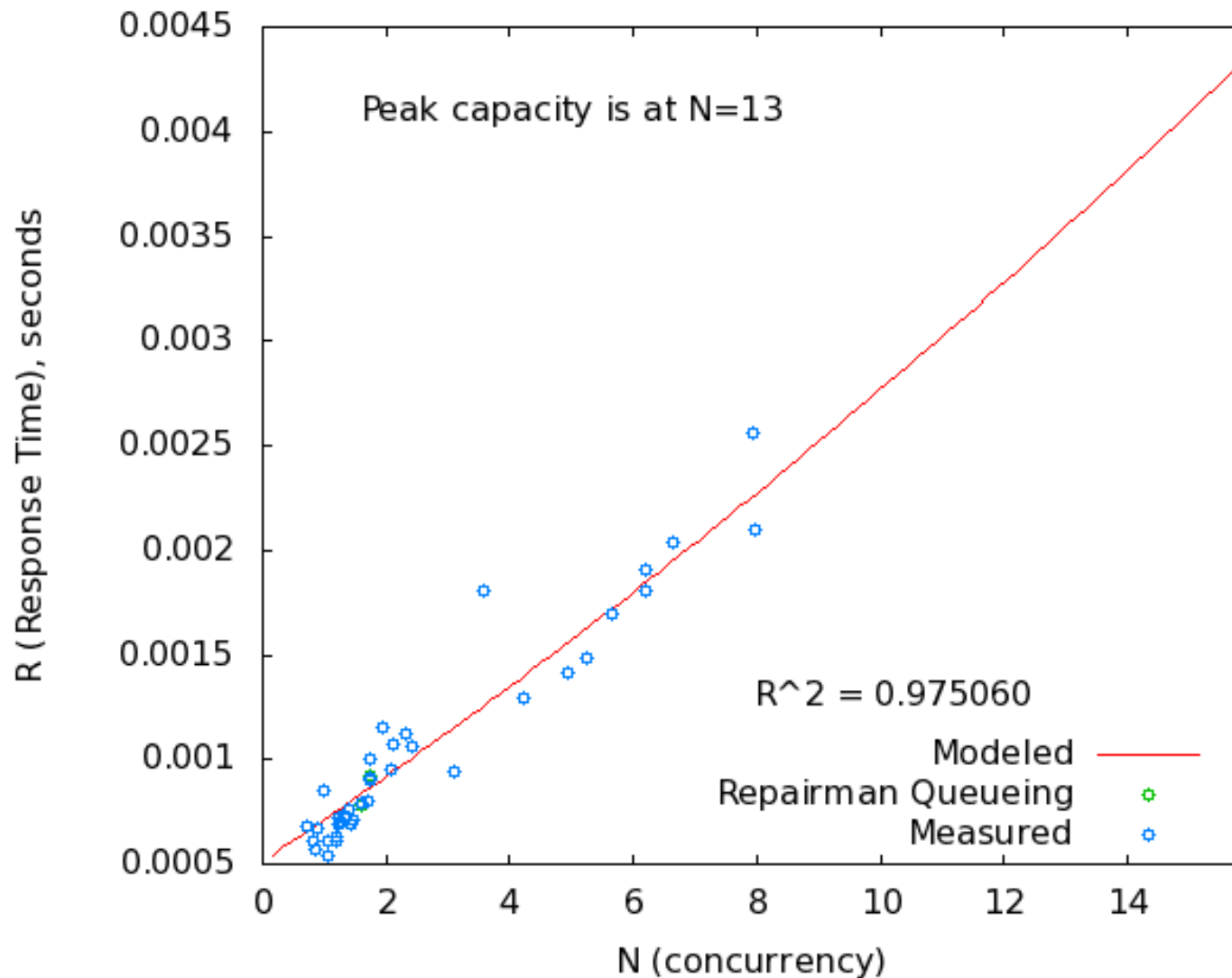


What is performance?

Forecasting Performance

- Performance = Response Time
- Little's Law: $N = XR$
(Concurrency = Throughput * Response Time)

Forecasting Performance



Validating Input

Garbage in, garbage out.

Validating Input

- Do not just throw data at the model.
- Beware of dropped packets in tcpdump.
- You may need to remove outliers.
- Beware of highly mixed / changing workloads.

When Is This Useful?

- Worst-case bound
- Best-case bound

Be Vewwy Vewwy Quiet

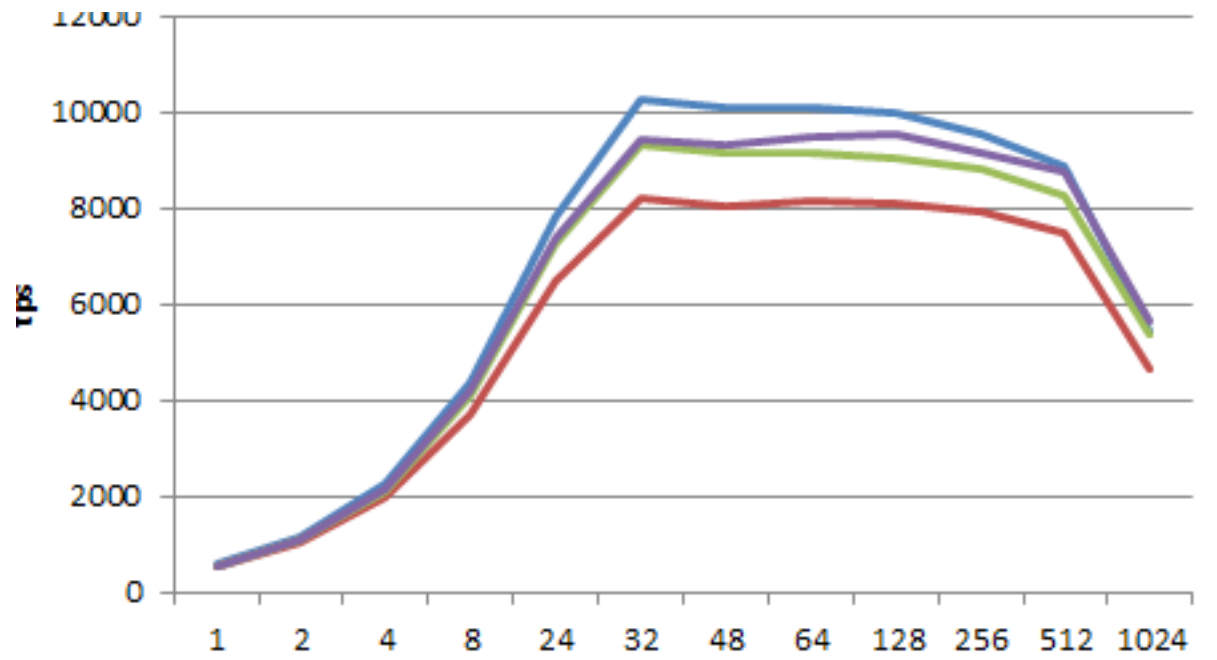


I'm wooking for bottwenecks!

Worst-Case Scaling

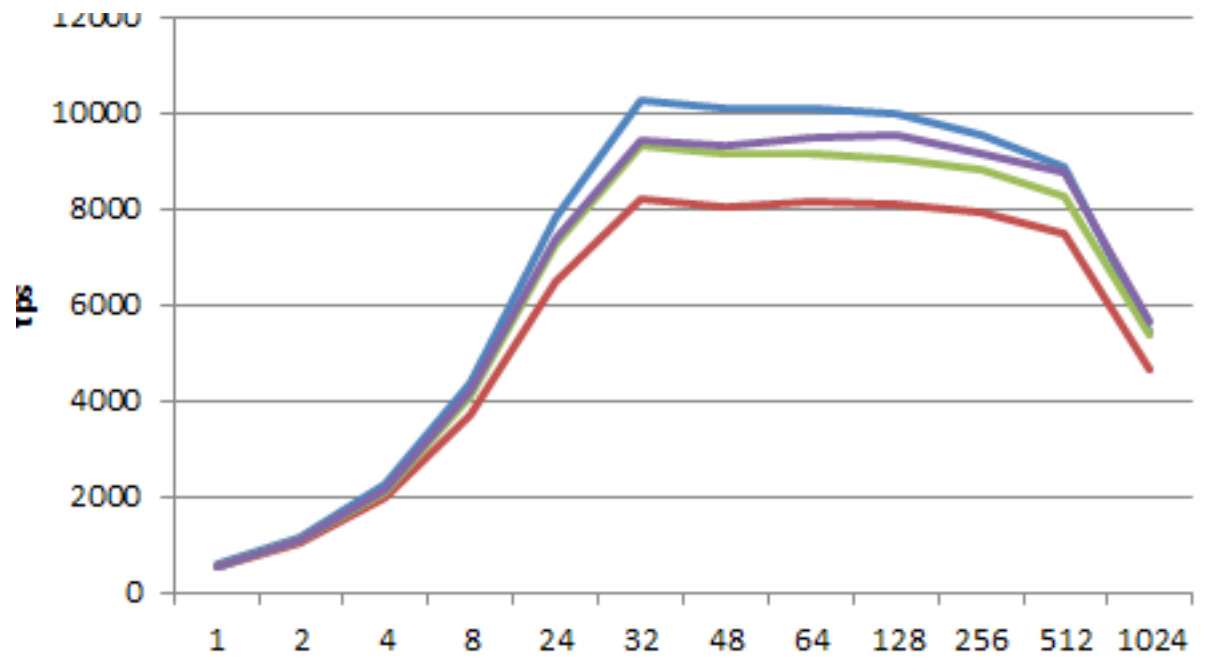
- The USL models synchronous repairman queuing behavior.
- This is worst-case, and your system should scale *better* than this.
- If not, you have a bottleneck. Go hunting.

Double Rainbow!

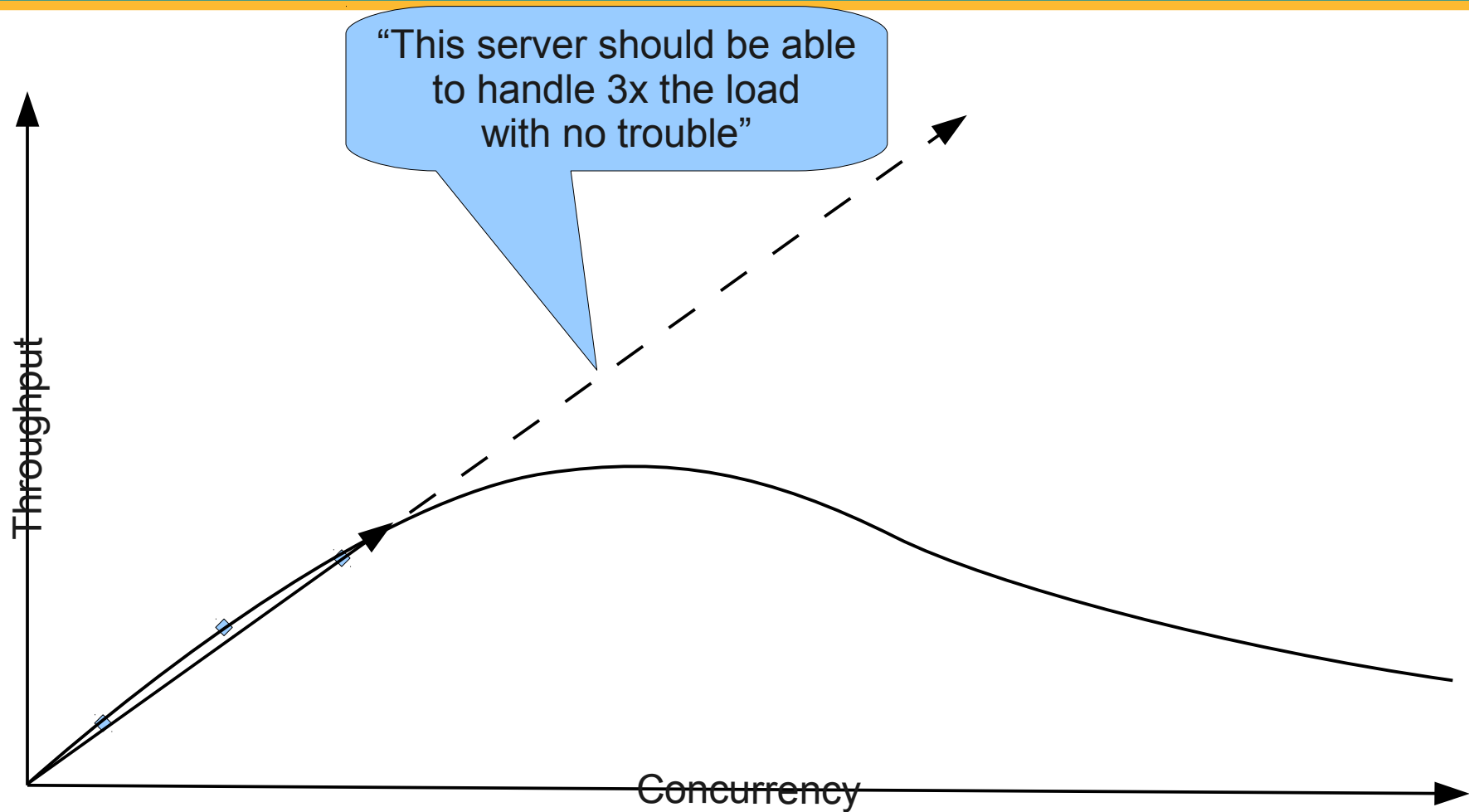


What's Wrong With That Plot?

- Why is it concave upwards till 24 threads?
 - It's because the X-axis scale is not linear.



“Assuming Linear Scalability...”



Best-Case Bounds

- We know most systems don't scale as well as they're supposed to.
- Therefore, the USL can be used for capacity planning purposes as a *best-case* bound.
- “I expect this system not to scale as well as predicted, so I'd better not count on getting any more performance than the model indicates.”

Summary

- We can get arrivals & completions from TCP
- And measure query performance easily
- And find problems the eye can't see
- And forecast beyond what we can observe
- For nearly any system
- Including ones that have no instrumentation!
- Forecasting/modeling requires experience and judgment, and does not give exact answers.

Resources

- Percona Toolkit: percona.com/software

Further Research

- Neil J. Gunther's book
 - *Guerrilla Capacity Planning*
- “Forecasting MySQL Scalability” white paper
 - <http://percona.com/about-us/mysql-white-papers>

Contact information: baron at percona.com