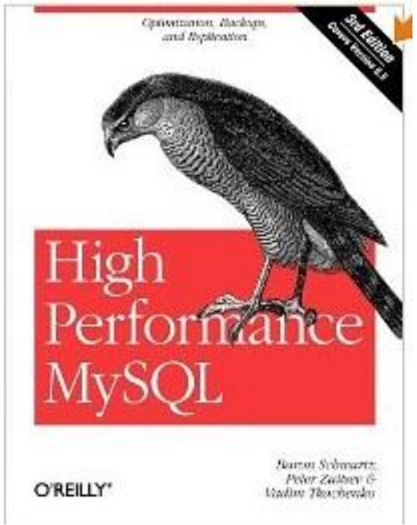
Extracting Performance and Scalability Metrics from TCP

Percona Inc
2012

About us

- Prepared by Baron Schwartz
  - Chief Performance Architect, Percona Inc
- Presented by Vadim Tkachenko
  - CTO, Percona Inc
- Percona
  - Consulting, Support, Development for MySQL
- “High Performance MySQL”, 2nd,3rd editions
- MySQLPerformanceBlog.com



Agenda

- Fundamental Metrics of Performance
- Capturing TCP Data
- Part 1: Black-Box Performance Analysis
  - Detecting Stalls and Locking
  - Detecting Performance Variations
- Part 2: Forecasting Scalability and Performance
  - A Mathematical Model of Scalability
  - Evaluating Results Against the Model
  - Real-World Applications

Why TCP/IP Headers are Great

- IP headers + TCP headers = 384 bytes
- This is usually non-privileged data, and it's easy to get
- It provides the following interesting data:
  - Origin IP address and TCP port
  - Destination IP address and TCP port
  - TCP sequence number, etc, etc
- In addition, by observing with tcpdump, we get:
  - Packet timestamp

The Fundamental Metrics

- In a protocol with call-and-response semantics, the following are enough to learn a lot:
  - Arrival time
  - Completion time
  - Session identifier

Derived Metrics

- Straightforward metrics over an observation interval
  - Queries per second (throughput)
  - Busy time
  - Total execution time
- Derived via Little's Law, the Utilization Law, etc
  - Average concurrency
  - Average response time
  - Utilization

Capturing TCP/IP Network 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
```

Capturing TCP/IP Network Traffic

- Beware of dropped packets!
- Sometimes writing to a file with -w works better.

A Sample of the Data

```
2012-02-10 10:30:57.818202 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 142
2012-02-10 10:30:57.818440 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 64
2012-02-10 10:30:57.819916 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 246
2012-02-10 10:30:57.820229 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 2896
2012-02-10 10:30:57.820239 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 1168
2012-02-10 10:30:57.822832 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 142
2012-02-10 10:30:57.823071 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 64
```

A Sample of the Data

2012-02-10 10:30:57.818202 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 142
2012-02-10 10:30:57.818440 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 64
2012-02-10 10:30:57.819916 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 246
2012-02-10 10:30:57.820229 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 2896
2012-02-10 10:30:57.820239 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 1168
2012-02-10 10:30:57.822832 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 142
2012-02-10 10:30:57.823071 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 64

2012-02-10 10:30:57.818202 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 142

Transforming the Data

```
2012-02-10 10:30:57.818202 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 142
2012-02-10 10:30:57.818440 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 64
2012-02-10 10:30:57.819916 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 246
2012-02-10 10:30:57.820229 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 2896
2012-02-10 10:30:57.820239 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 1168
2012-02-10 10:30:57.822832 IP 10.124.62.89.56520 > 10.124.62.75.3306: tcp 142
2012-02-10 10:30:57.823071 IP 10.124.62.75.3306 > 10.124.62.89.56520: tcp 64
```

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

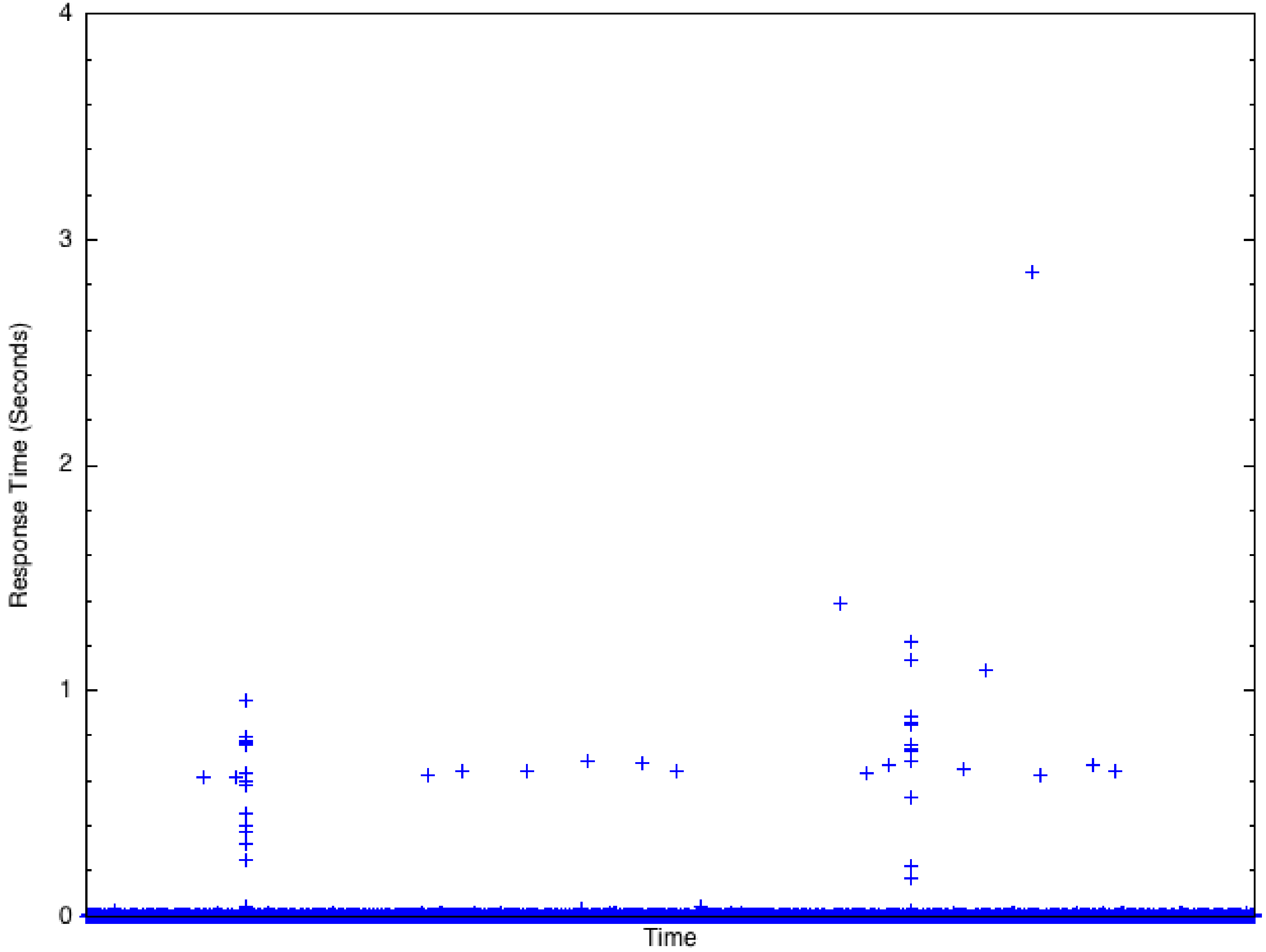
```
 #  start-timestamp          end-timestamp            elapsed  host:port
== ================= ================= ======== ====================
 7 1328887857.818202 1328887857.818440 0.000238 10.124.62.89:56520
10 1328887857.819916 1328887857.820229 0.000313 10.124.62.89:56520
14 1328887857.822832 1328887857.823071 0.000239 10.124.62.89:56520
15 1328887857.824518 1328887857.824828 0.000310 10.124.62.89:56520
13 1328887857.822784 1328887857.823108 0.000324 10.124.62.89:56523
16 1328887857.826182 1328887857.826419 0.000237 10.124.62.89:56520
19 1328887857.827202 1328887857.827438 0.000236 10.124.62.101:57780
20 1328887857.827348 1328887857.827661 0.000313 10.124.62.106:54368
12 1328887857.821355 1328887857.821611 0.000256 10.124.62.101:57779
```

About The Following Graphs

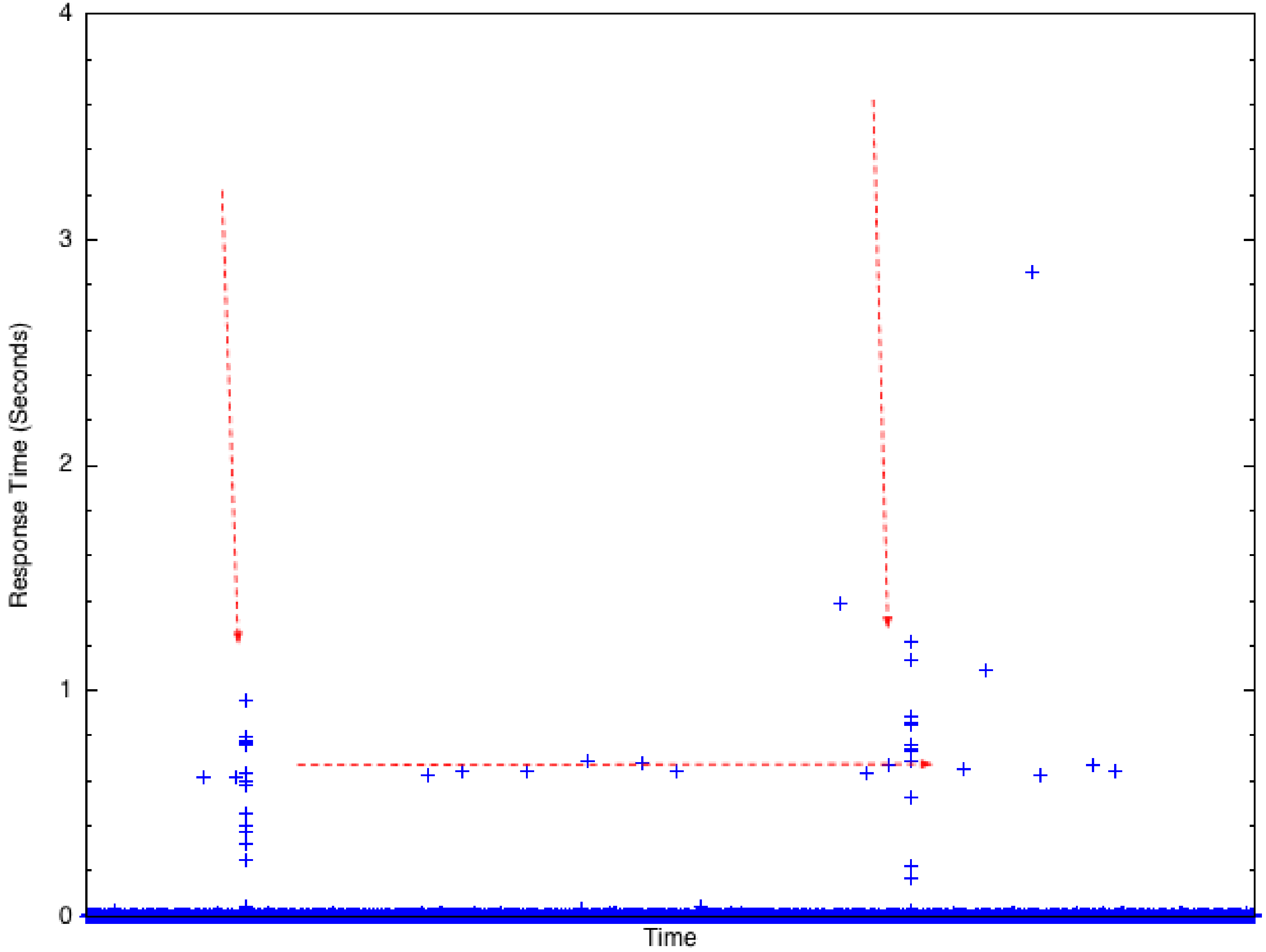
- The following plots are from several samples
- They range from ~10s to ~2m in duration
- Application load was low to moderate
- The application is a Ruby On Rails e-commerce site
- The database has a mixed workload (not just RoR)

Black-Box Performance Analysis

Step 1: Plot on a Time-Series Chart

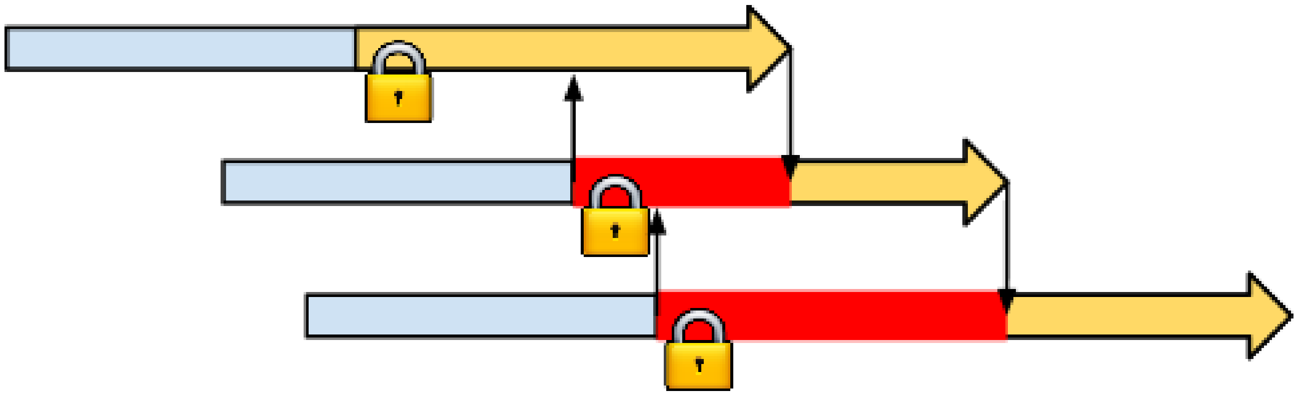


What do the Anomalies Mean?



Stalls Explained

- The points are plotted in order of *completion*.
- They complete in the order their dependencies are met.
- That's why the spikes slope to the right slightly.

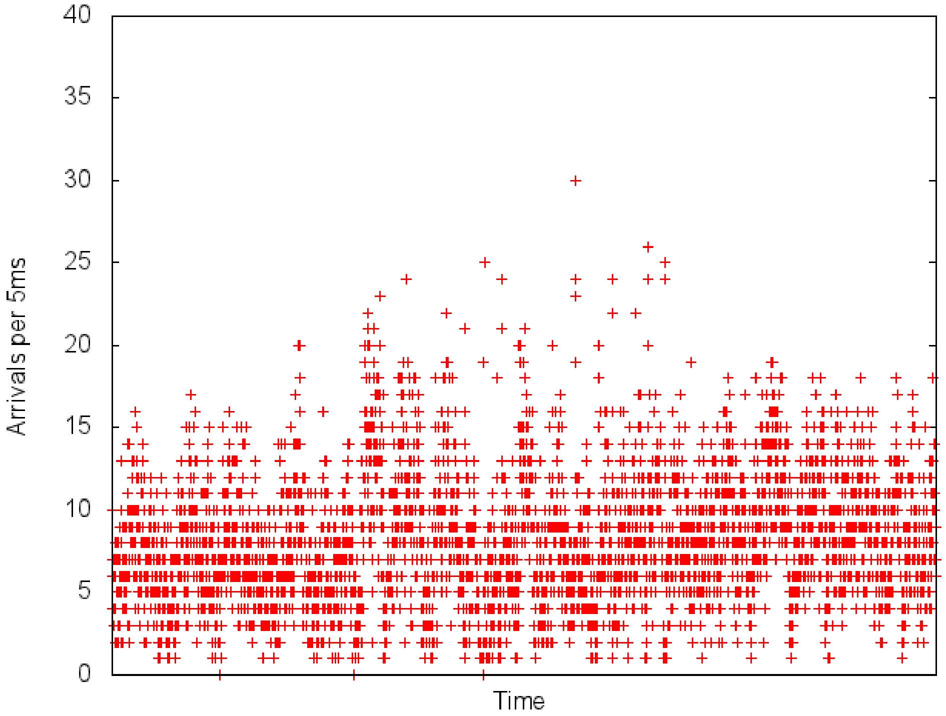


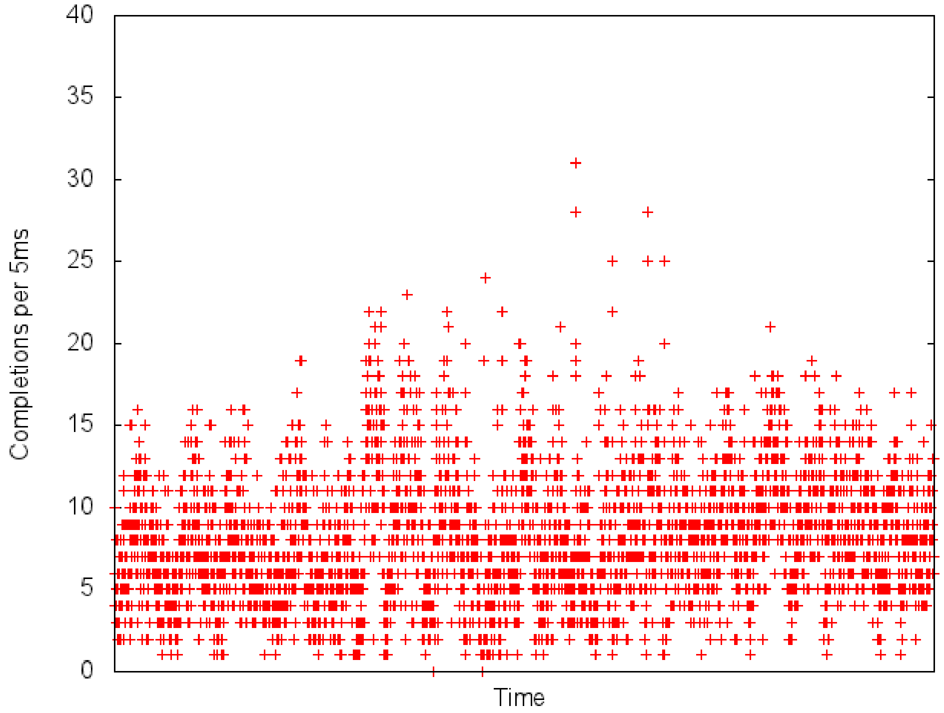
The Stalls are SELECT FOR UPDATE.

- I actually captured 4096 bytes of the packet, not 384
- I used pt-query-digest to inspect the queries in the protocol
- The dependencies are caused by explicit locking
- Completions cluster together when they are all waiting for the same lock

Can Completion Times Reveal More?

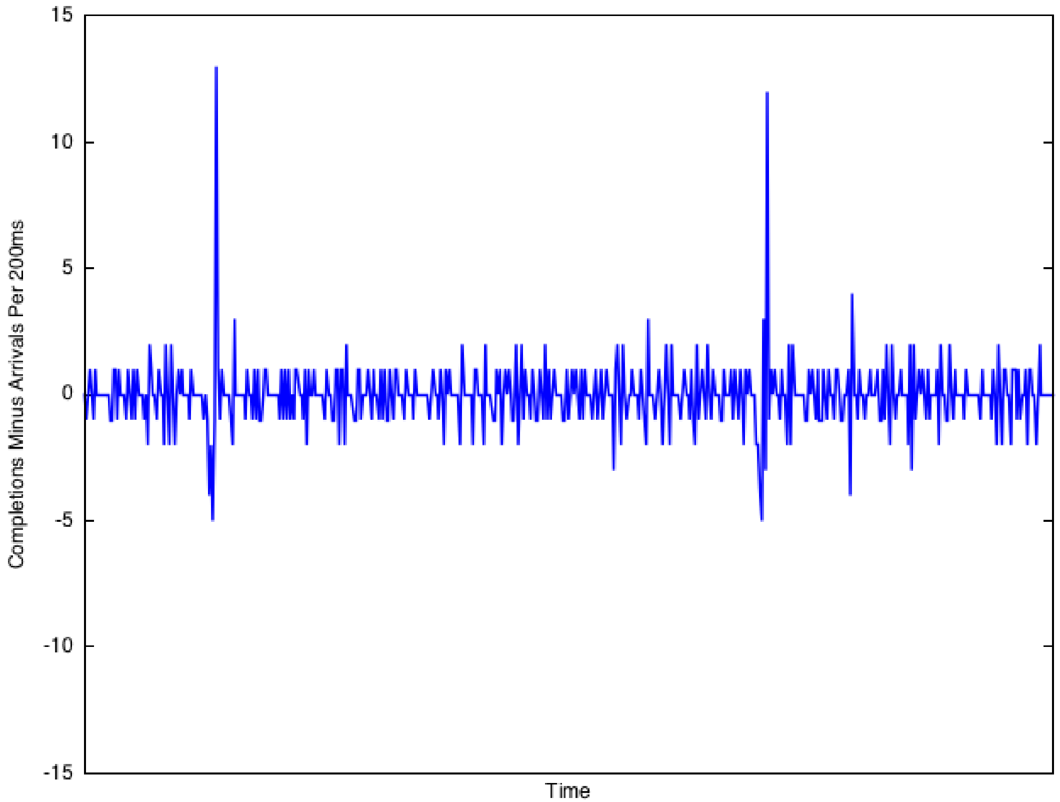
- Maybe we can compare completion counts -vs- arrivals?
- The following charts show counts per 5ms.





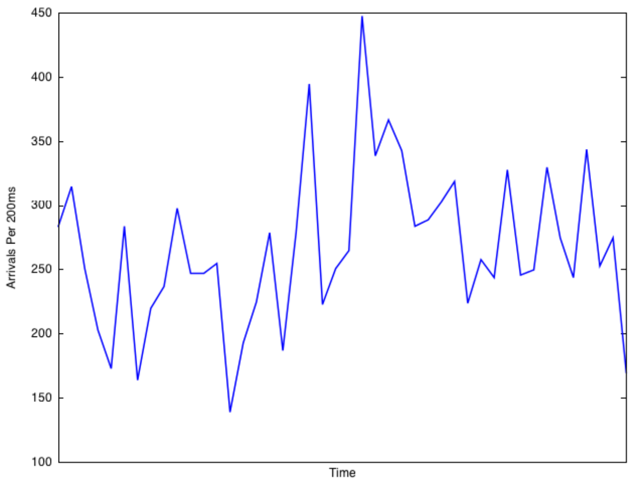
Subtraction and Coarser Aggregation

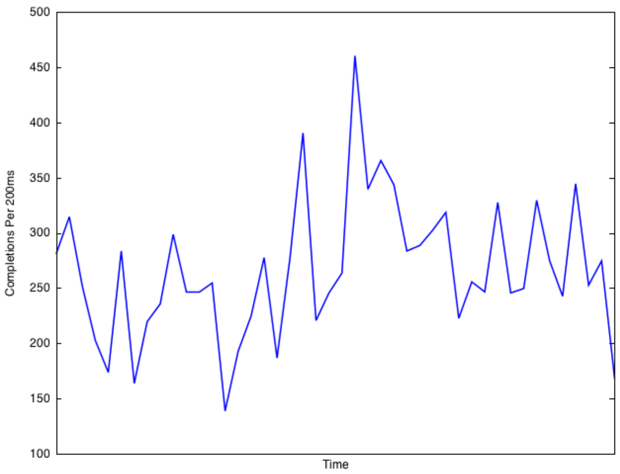
- 5ms is too fine-grained
- It's too hard to compare scatter plots
- Subtract arrivals from completions, 200ms at a time

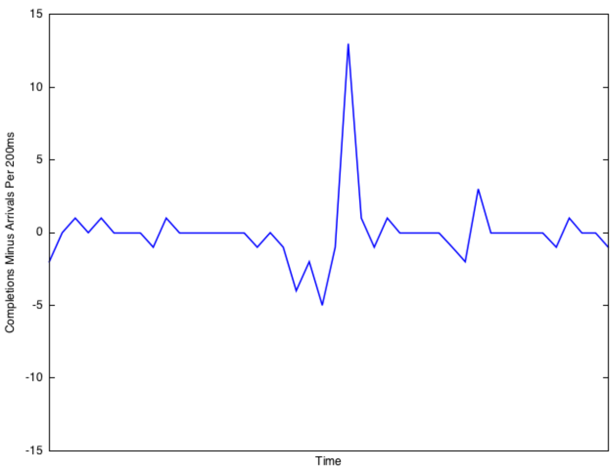


Why Does This Work?

- On average, arrivals ~= completions in any interval
- When a stall occurs on an interval boundary,
  - The first interval gets many arrivals that don't complete
  - The second interval gets more completions
  - The graph dips, then spikes







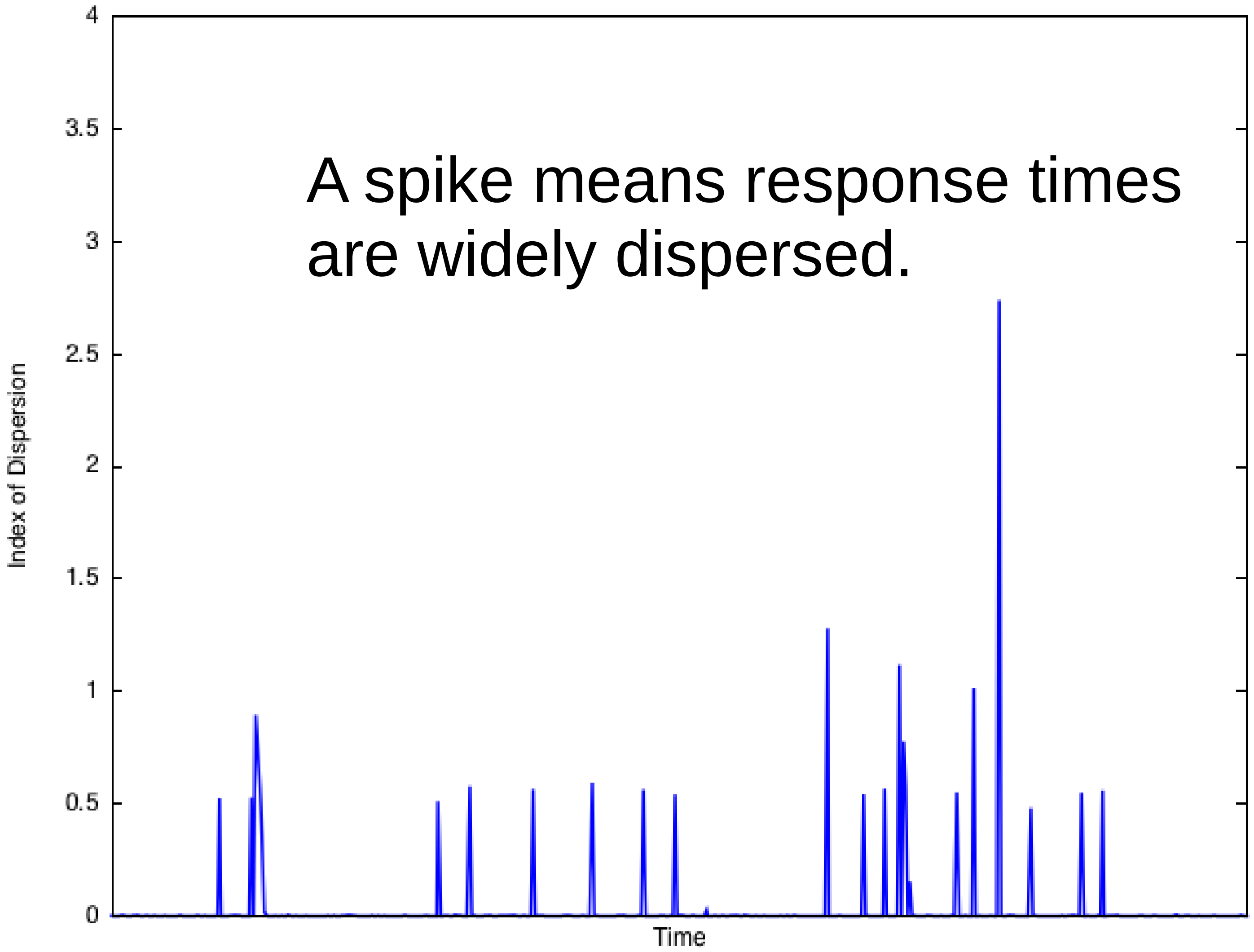
Detecting Performance Variations

- Most statistics (max, quantile, avg, stdev) are unhelpful
- Variance-to-mean ratio (index of dispersion) is very useful.

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

- Normalized measure of the dispersion of response times.

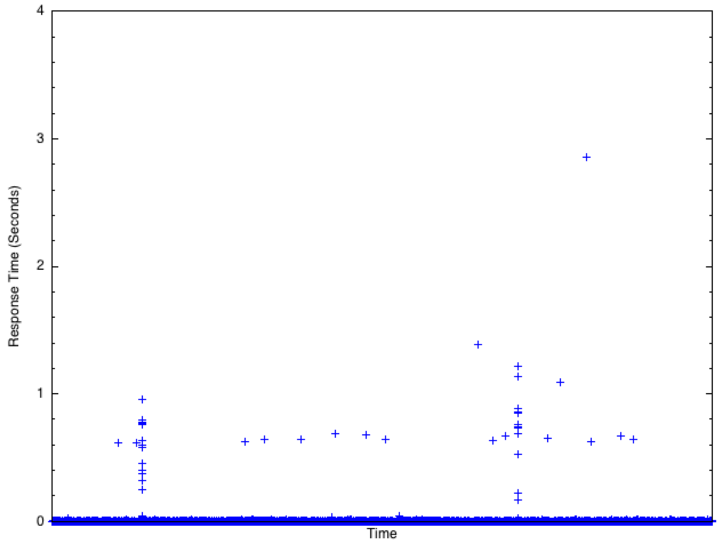
Plotting the Index of Dispersion

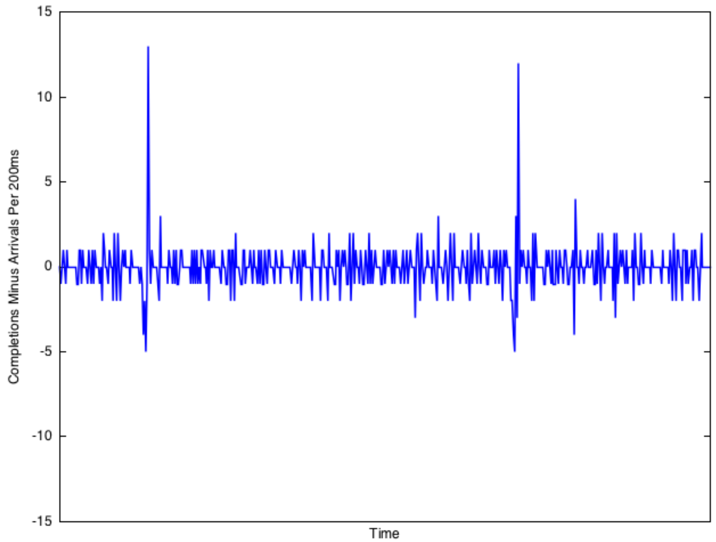


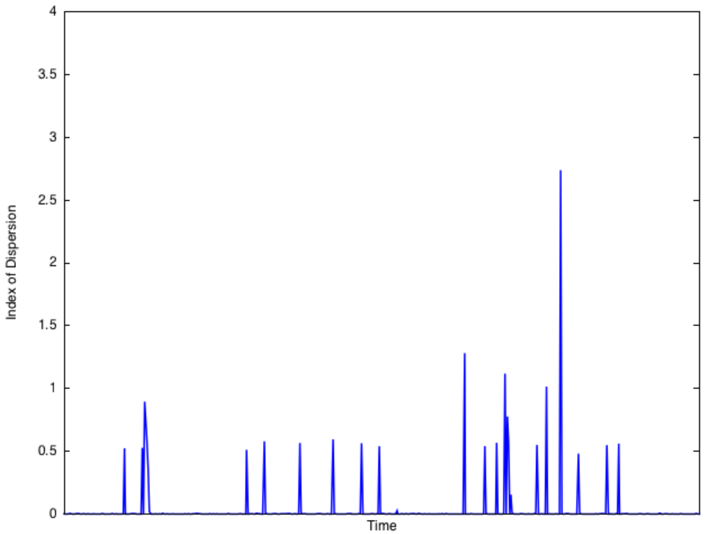
Interpreting Index of Dispersion

- Highly variable == highly optimizable
- Uniform, consistent performance is preferable

All the Plots Together



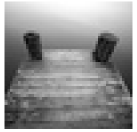




In the Real World

https://twitter.com/#!/aaronbbrown777/status/123469227985354752

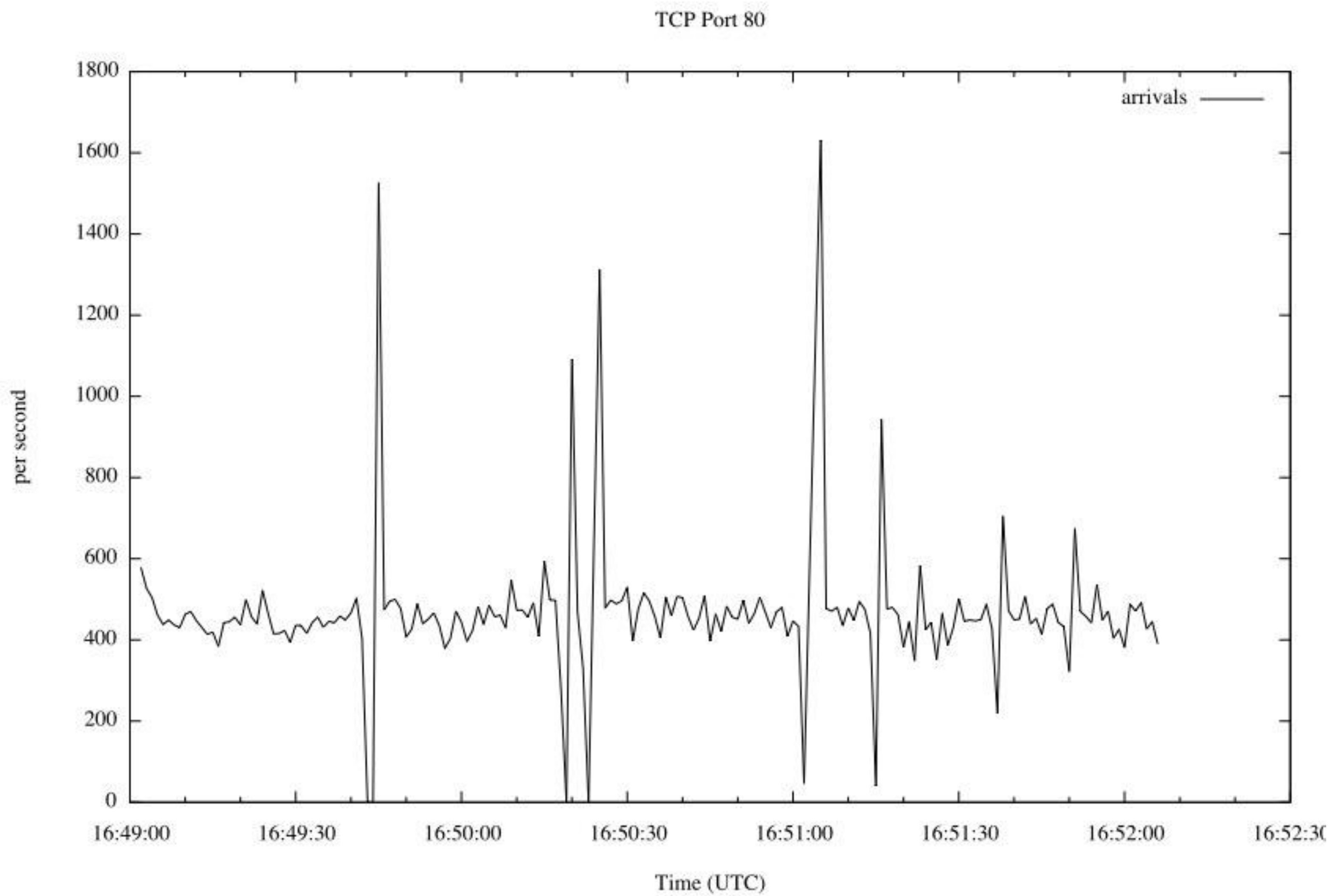
Search | Home | Profile | Messages | Who To Follow



@AaronBBrown777
Aaron Brown

Non-ideal TCP traffic entering the load balancer skitch.com/aaronbbrown/f9... (created with help from pt-tcp-model by @xaprb & @percona)

In the Real World

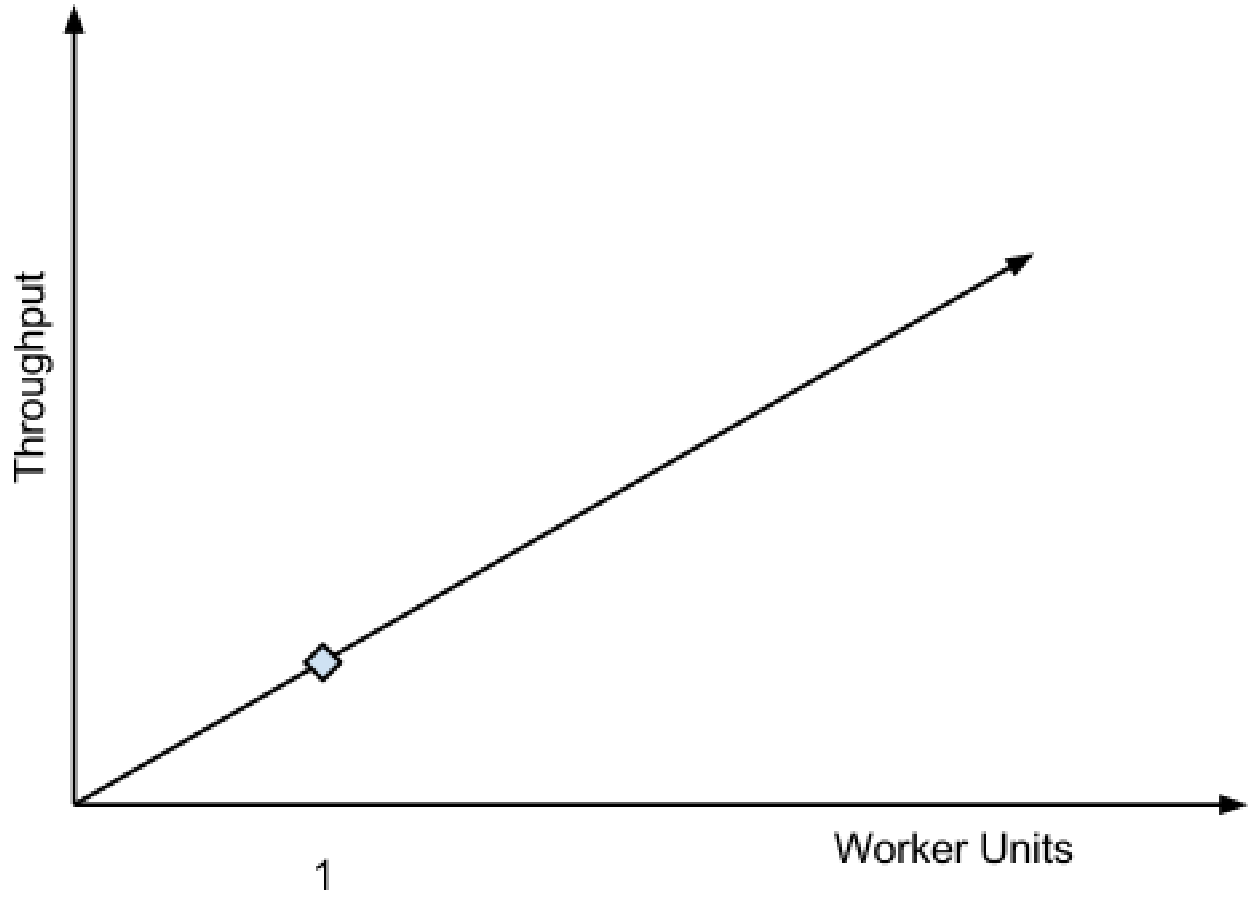


Part 2:
Forecasting Scalability and Performance

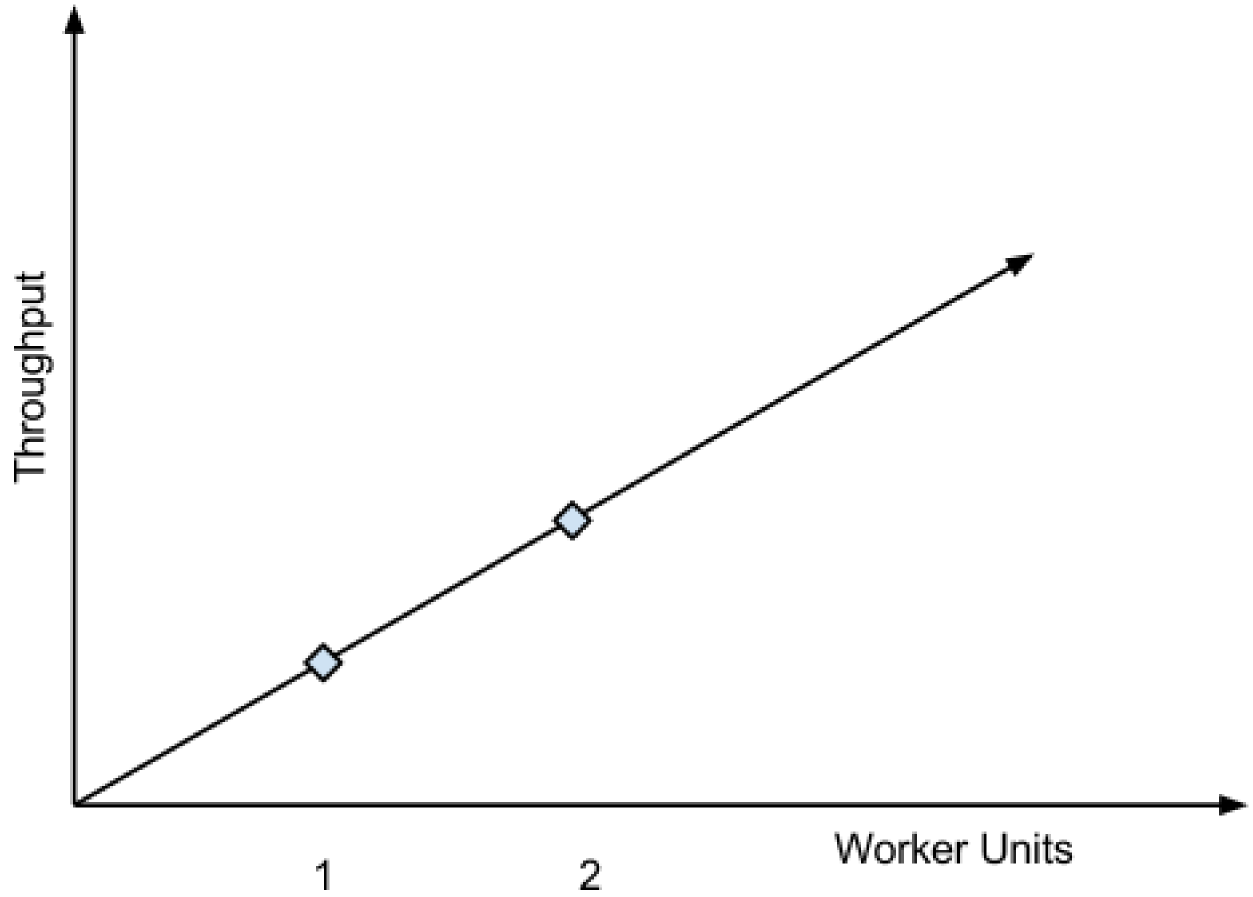
Defining Scalability

- Scalability is a mathematical function (equation)
- The X-axis is the number of worker units
- The Y-axis is throughput

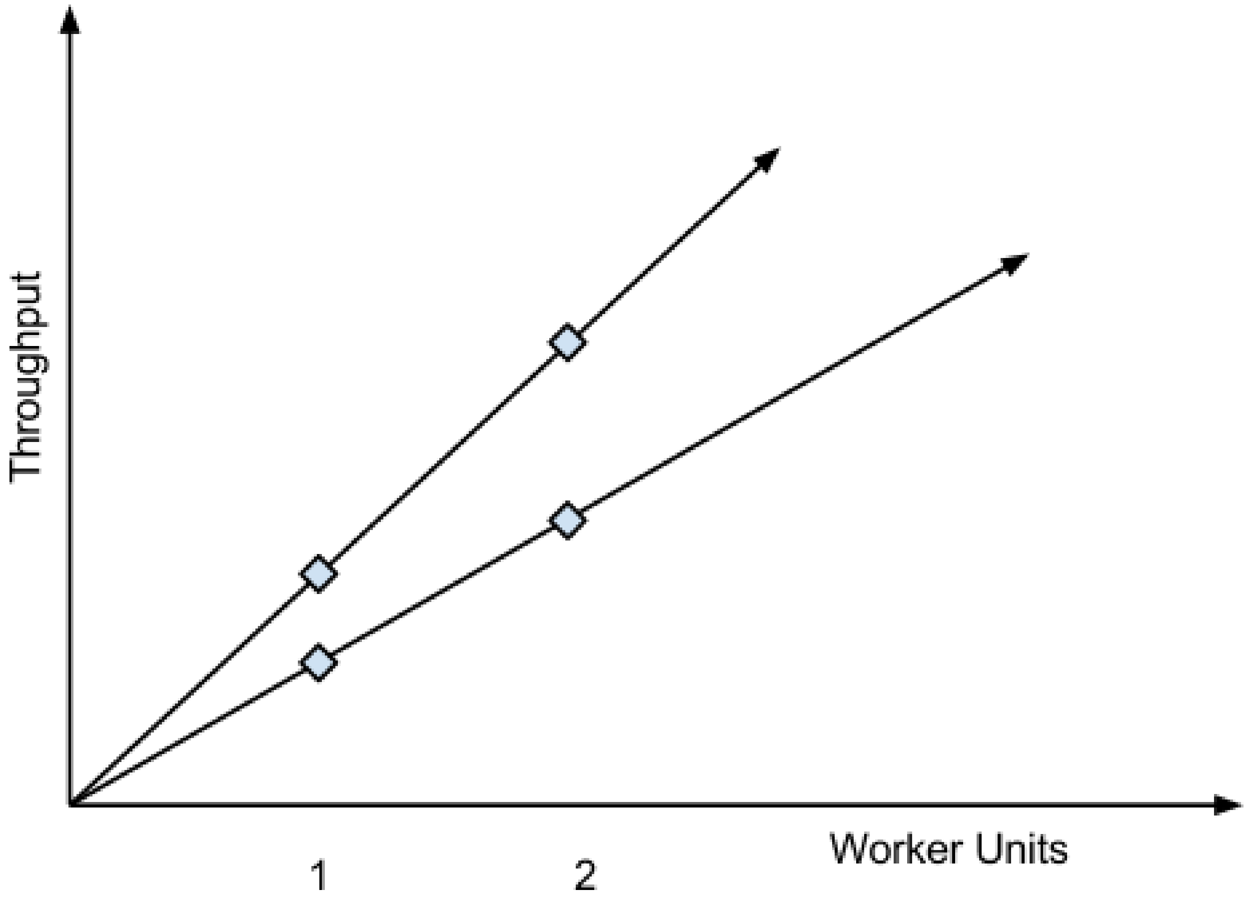
The Scalability Function



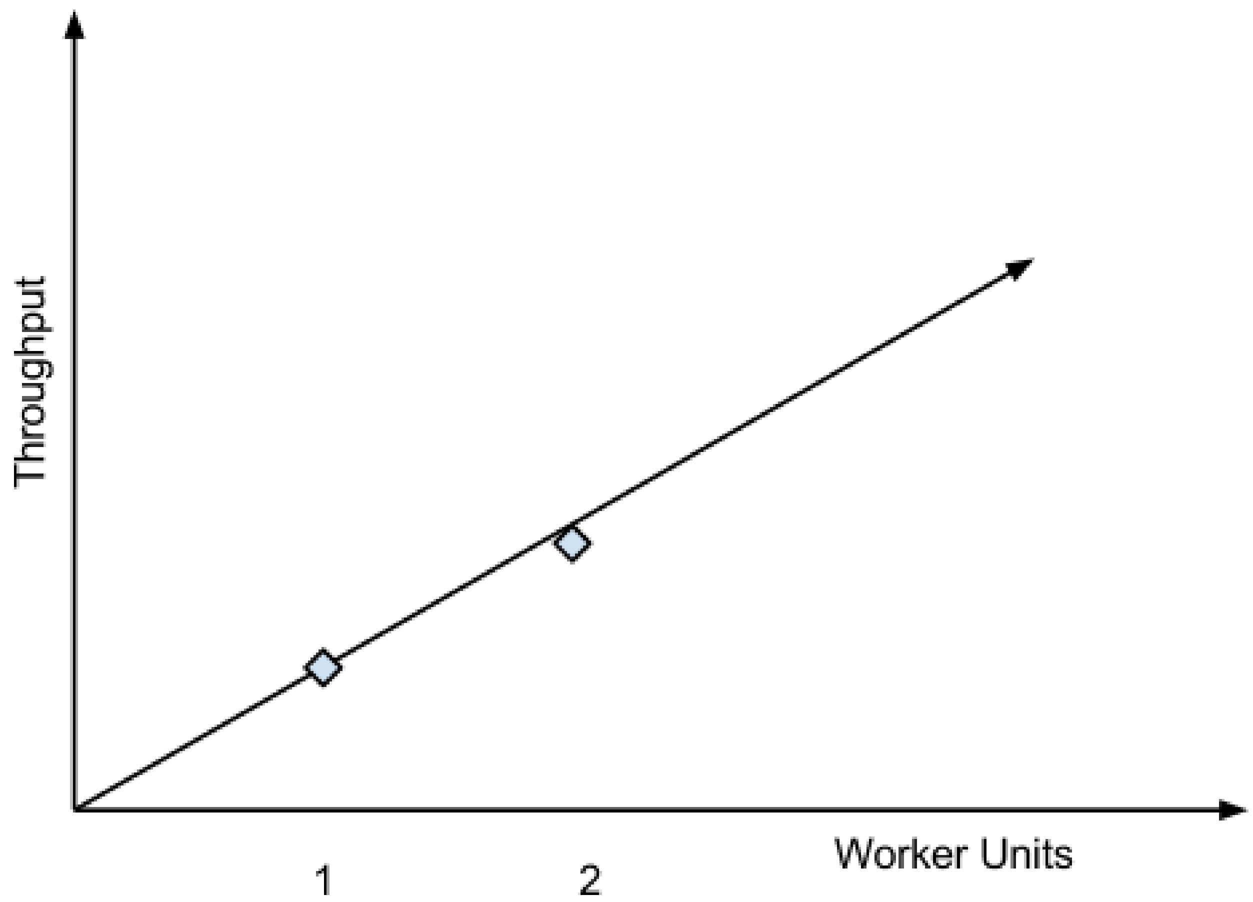
Linear Scalability



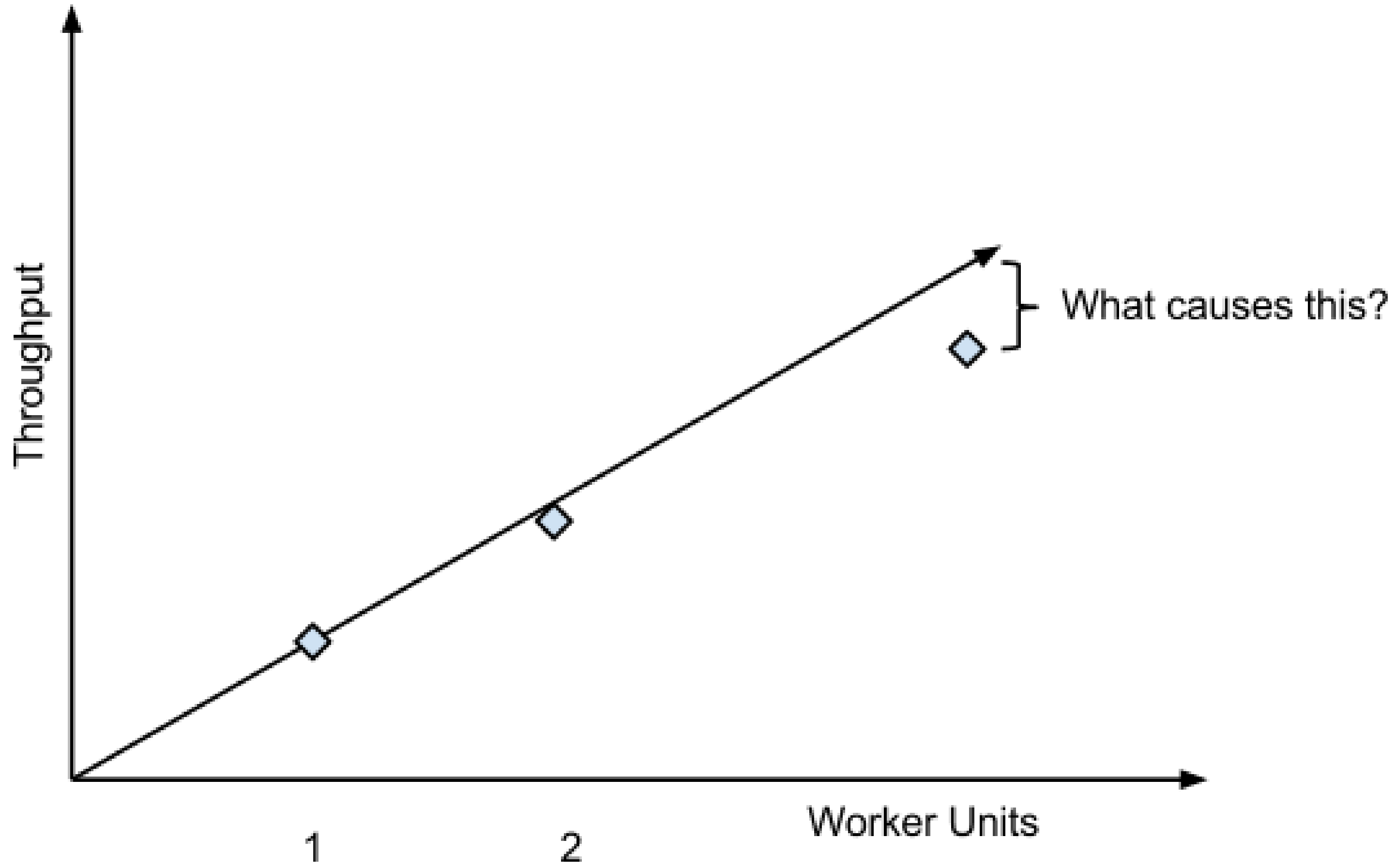
Also Linear Scalability



Not Linear Scalability

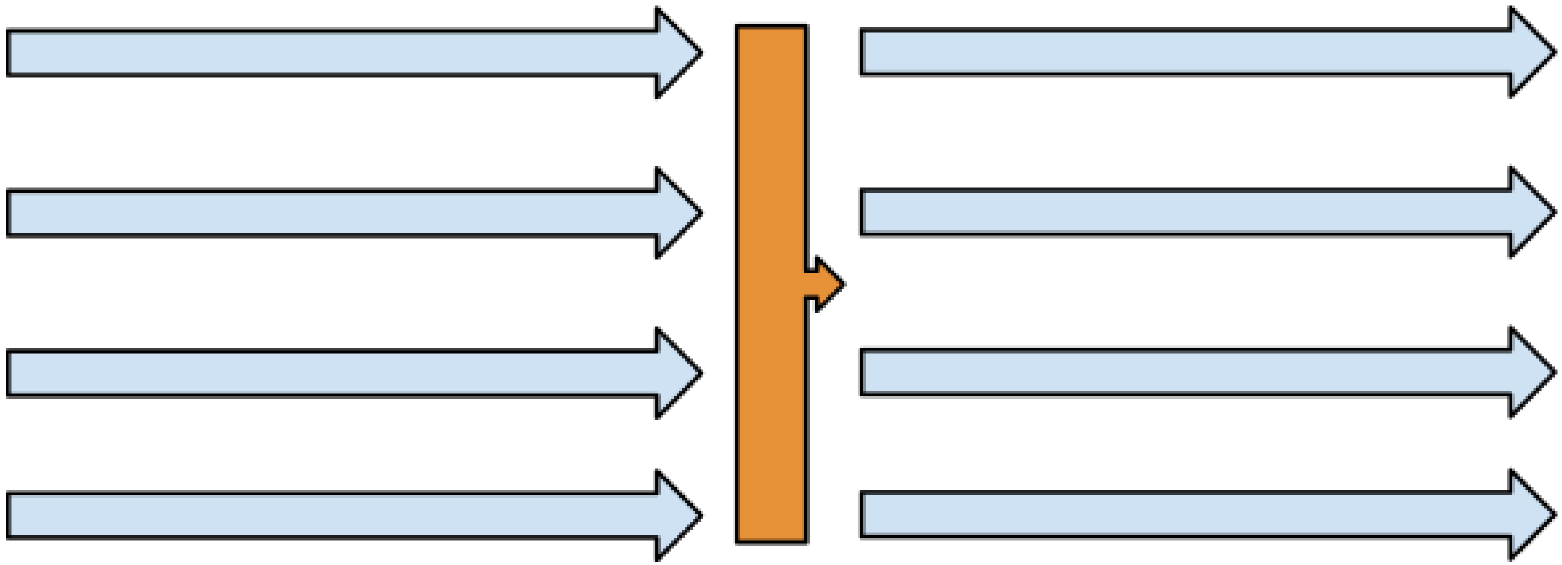


What Causes Non-Linearity?



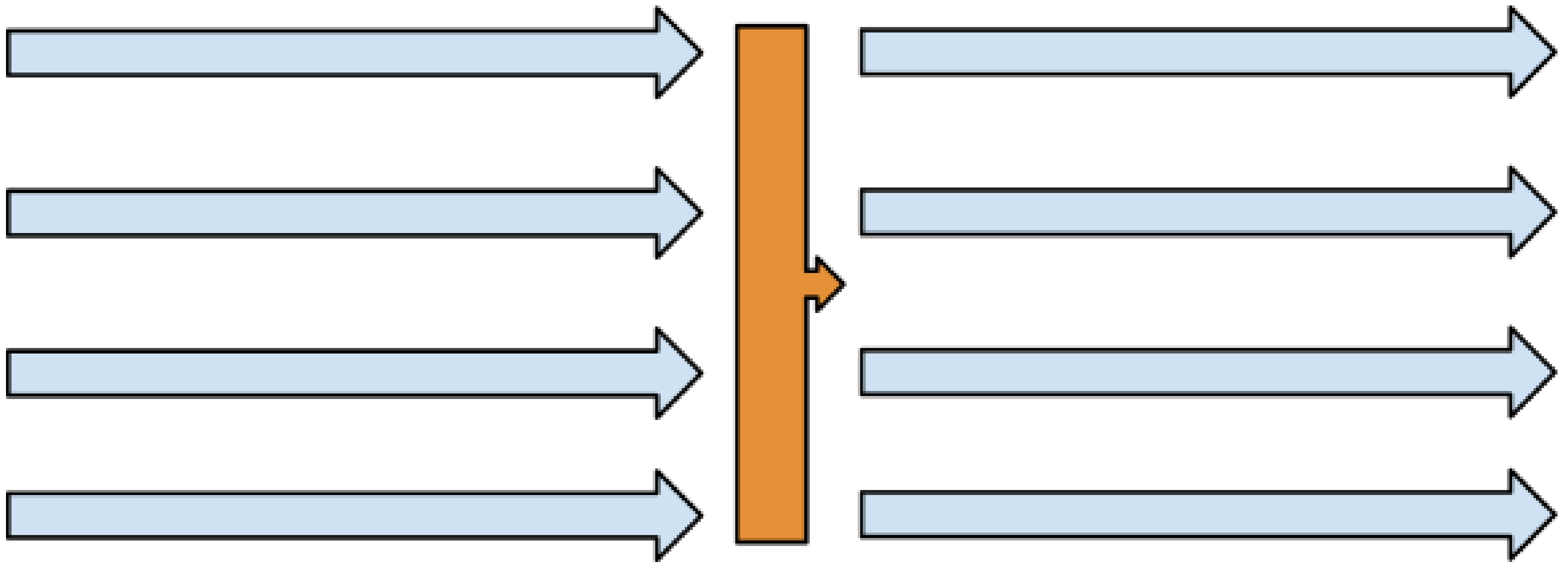
Factor #1: Serialization

- Amdahl's Law: if not all work can be parallelized, speedup is limited to the reciprocal of the serialized portion.



Factor #1: Serialization

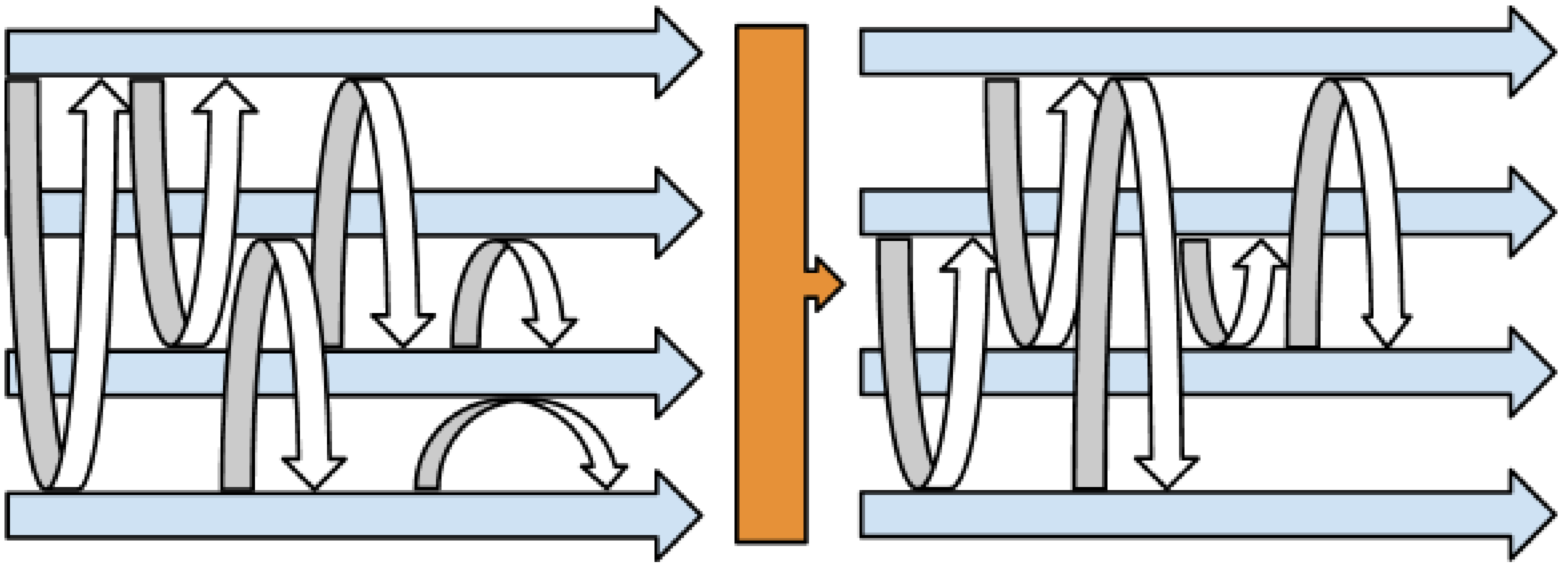
- Amdahl's Law: if not all work can be parallelized, speedup is limited to the reciprocal of the serialized portion.



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

Factor #2: Crosstalk

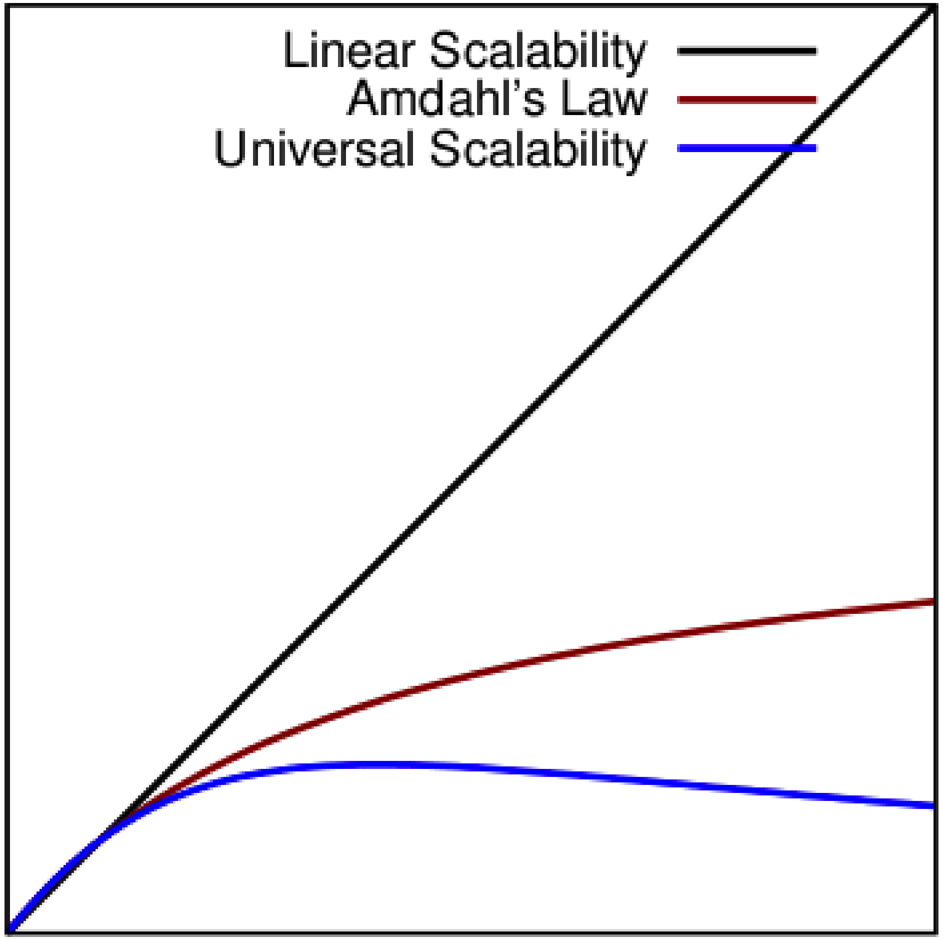
- Universal Scalability Law: scalability degrades in proportion to the number of crosstalk channels, which is O(n^2).



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

Degradation of Throughput

- Most systems have both serialization and crosstalk.



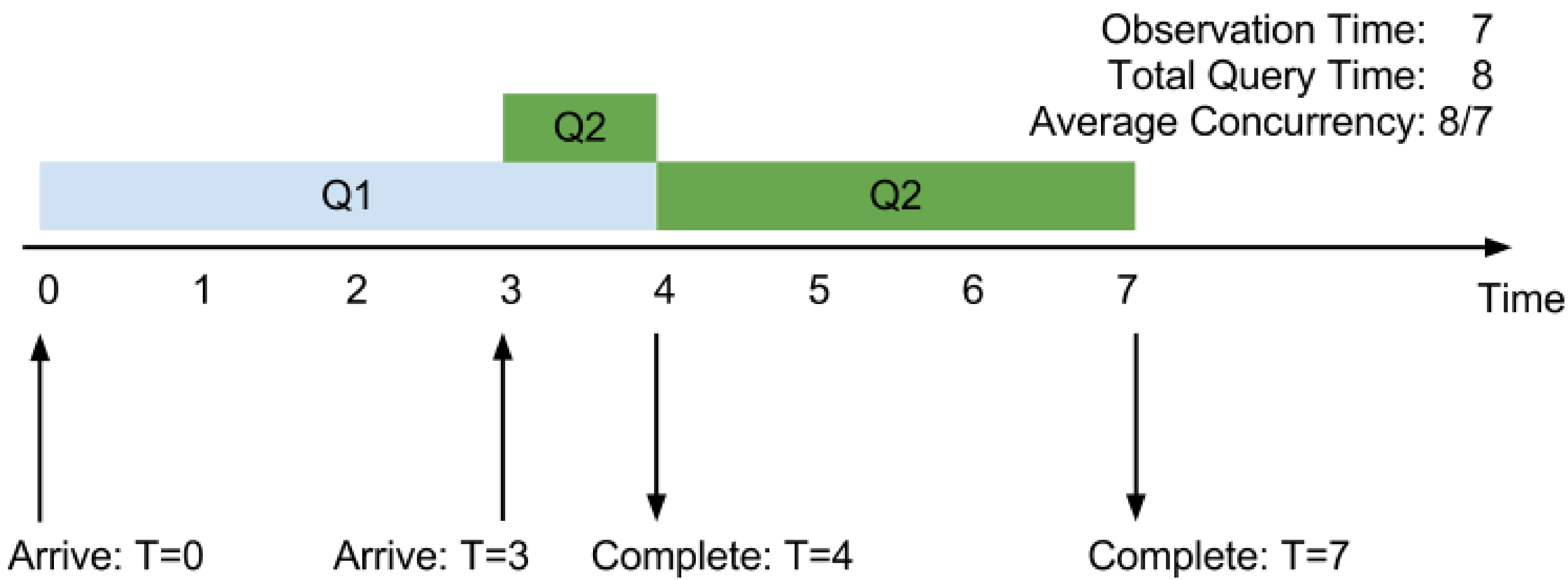
Scalability Modeling Algorithm

- Measure throughput and concurrency
- Perform a regression against the Universal Scalability Law
  - This determines the sigma and kappa coefficients
- ????
- Profit!

What Inputs Do We Need?

- Throughput is easy (queries per second)
- Concurrency is a little more subtle:
  - Sort the arrivals and departures by timestamp
  - Each arrival increments concurrency
  - Each departure decrements it
- Compute the average concurrency per time interval

The Concurrency Calculation



Using pt-tcp-model

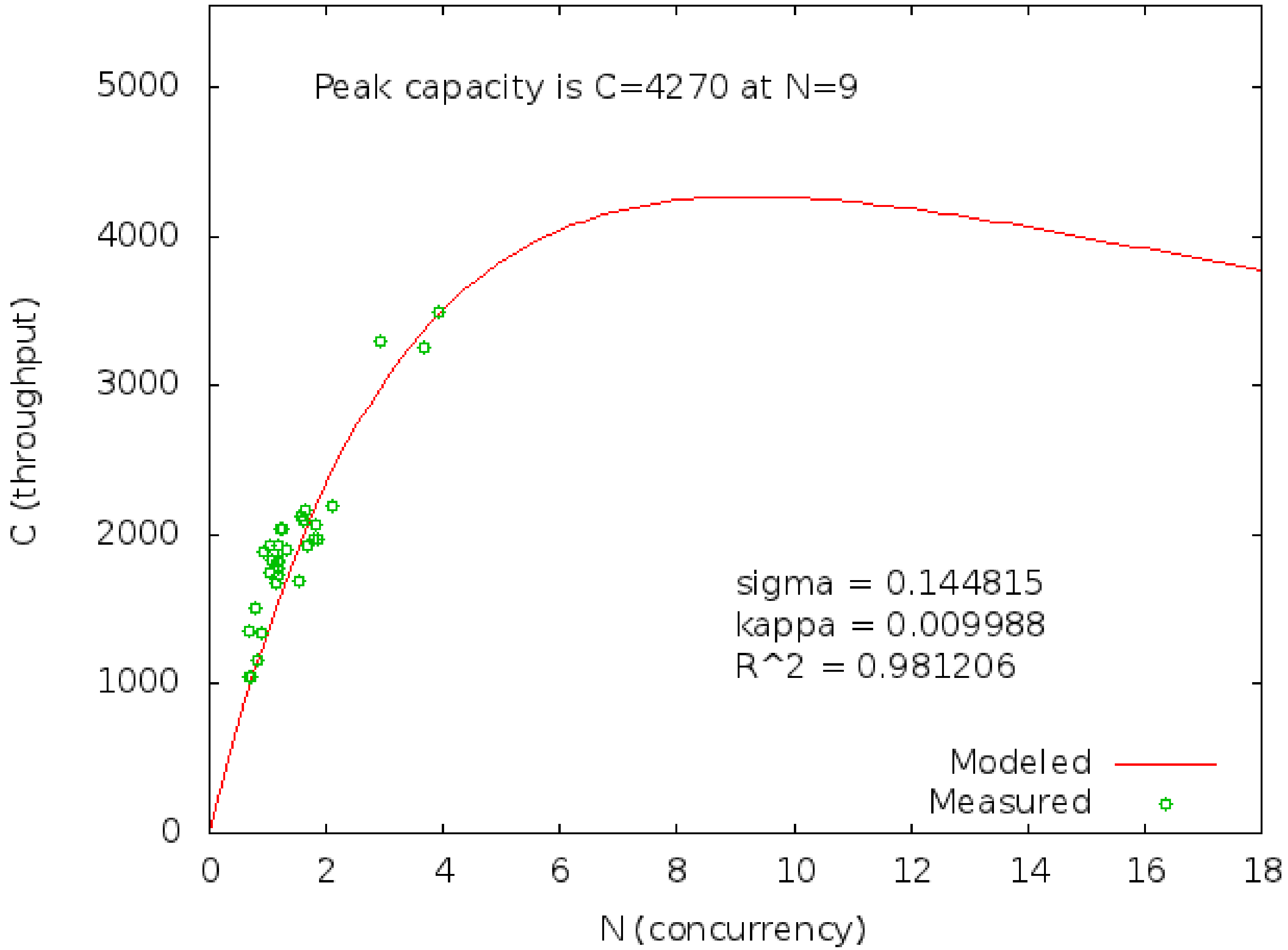
You can compute these metrics with pt-tcp-model.

```
sort -n -k1,1 requests.txt > sorted.txt
pt-tcp-model --type=requests sorted.txt > sliced.txt
```

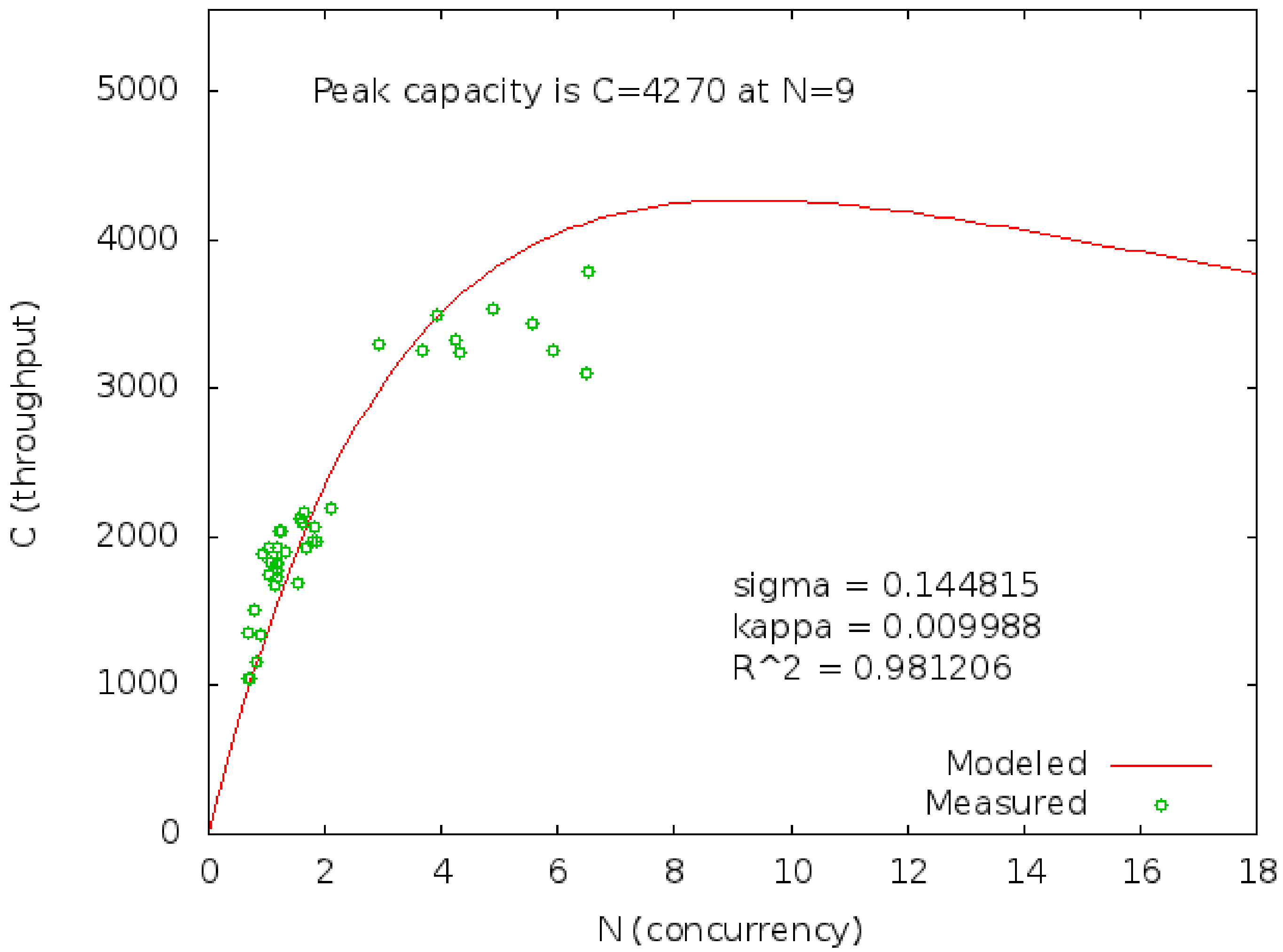
Determine Kappa and Sigma

- Use R, gnuplot or other tools to fit the model to the data and derive:
  - Coefficient of serialization (sigma)
  - Coefficient of crosstalk (kappa)

Results on a Partial Dataset



Results on the Full Dataset



How to Approach the USL

- The USL can be useful as a best-case or worst-case model

How to Approach the USL

- The USL can be useful as a best-case or worst-case model
- Worst-Case Bounds
  - The USL models worst-case scalability
  - Your system should scale *better* than that
  - Use it as a point of reference for "we can improve this"

How to Approach the USL

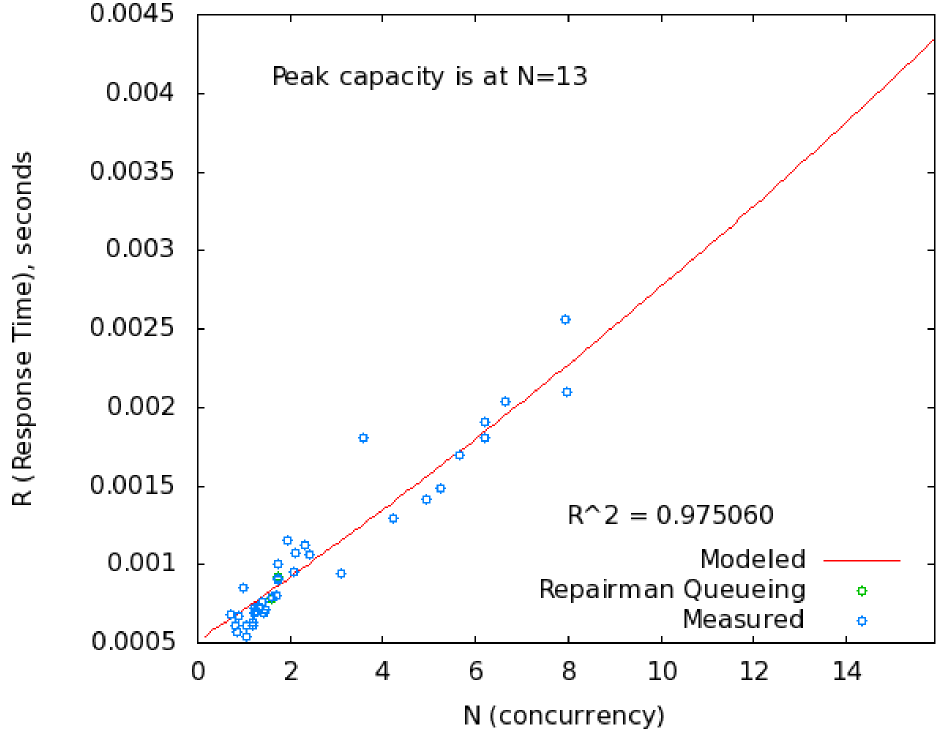
- The USL can be useful as a best-case or worst-case model
- Worst-Case Bounds
  - The USL models worst-case scalability
  - Your system should scale *better* than that
  - Use it as a point of reference for "we can improve this"
- Best-Case Bounds
  - Many systems don't scale as well as they should
  - When forecasting past observable limits, be pessimistic
  - "I expect this system to scale worse than predicted"

How to Approach the USL

- The USL can be useful as a best-case or worst-case model
- Worst-Case Bounds
  - The USL models worst-case scalability
  - Your system should scale *better* than that
  - Use it as a point of reference for "we can improve this"
- Best-Case Bounds
  - Many systems don't scale as well as they should
  - When forecasting past observable limits, be pessimistic
  - "I expect this system to scale worse than predicted"
- The USL is a *model*.

Forecasting Performance

- Performance = Response Time
- Little's Law: N = XR
  - concurrency = throughput * response time
- Thus R = N/X. You can model this just like scalability, with the same caveats.



Validate Your Input

- The USL works best on a well-behaved data set
- You may need to remove outliers
- You may need to select well-behaved windows of time
- Beware of mixed or variable workloads
- "Black box" plotting is a good place to start

Resources

- Percona Toolkit
  - http://www.percona.com/software/
- Neil J. Gunther's book
  - *Guerrilla Capacity Planning*
- Percona White Papers
  - "MySQL Performance Analysis..."
  - "Forecasting MySQL Scalability..."
  - http://www.percona.com/about-us/mysql-white-papers
- These slides
  - http://goo.gl/kUQNz

baron@percona.com
@xaprb

PERCONA LIVE

www.percona.com/live