Super Sizing Your Servers

and the Payback Trap

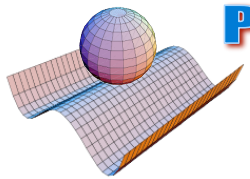
Dr. Neil Gunther

Performance Dynamics Labs

USENIX **LISA14**

Invited Talk

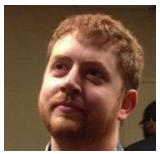
Seattle Sheraton November 13, 2014



Performance Dynamics Company℠

Acknowledgements

Special thanks to:



@patcable for inviting me to speak at LISA14



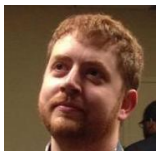
@HilaryHartman for helping a LISA first-timer



Rik Farrow for “censoring” my S4 Tutorial slides ☺

Acknowledgements

Special thanks to:

 @patcable for inviting me to speak at LISA14

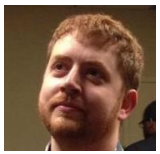
 @HilaryHartman for helping a LISA first-timer

 Rik Farrow for “censoring” my S4 Tutorial slides ☺

All the LISA14 conference organizers

Acknowledgements

Special thanks to:

 @patcable for inviting me to speak at LISA14

 @HilaryHartman for helping a LISA first-timer

 Rik Farrow for “censoring” my S4 Tutorial slides ☺

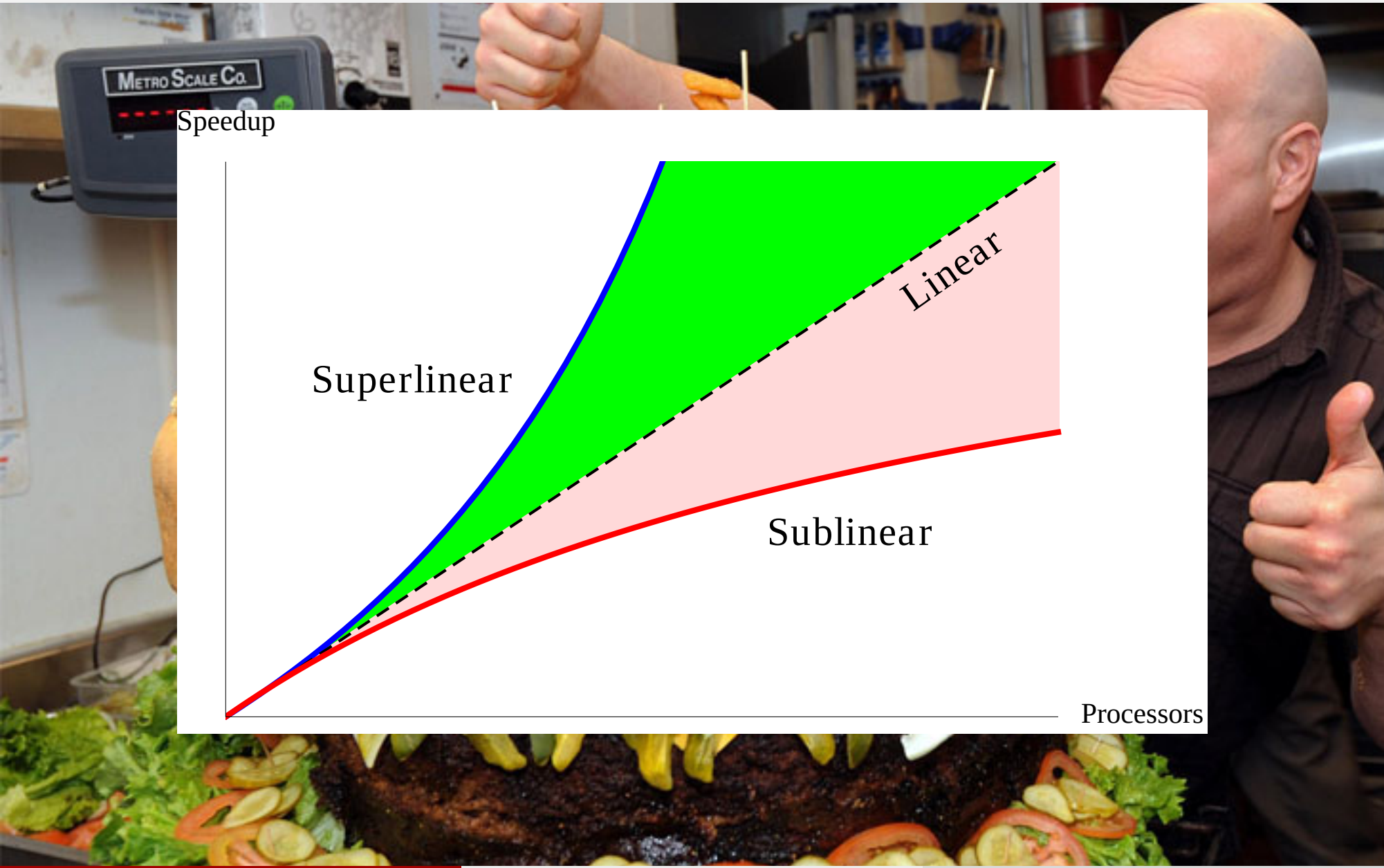
All the LISA14 conference organizers

And all the new people I’ve met at LISA14

What this talk is about



What this talk is about



Qualitative Scalability

Qualitative means:

- Scalability from an operational view
- Scalability as configuration recipes
- Lot's a words (on blogs), but no numbers

Cost-benefit analysis demands numbers, not words.
Need to measure scalability appropriately it to quantify it.

Qualitative Scalability

Qualitative means:

- Scalability from an operational view
- Scalability as configuration recipes
- Lot's a words (on blogs), but no numbers

Cost-benefit analysis demands numbers, not words.
Need to measure scalability appropriately it to quantify it.

But how?

Qualitative Scalability

Qualitative means:

- Scalability from an operational view
- Scalability as configuration recipes
- Lot's a words (on blogs), but no numbers

Cost-benefit analysis demands numbers, not words.
Need to measure scalability appropriately it to quantify it.

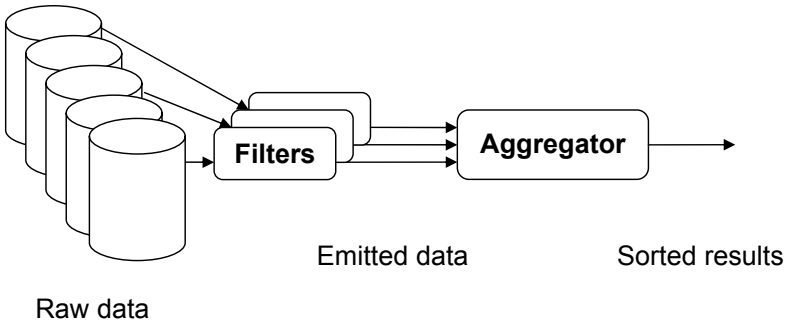
But how?

Need controlled measurements (e.g., Jmeter)

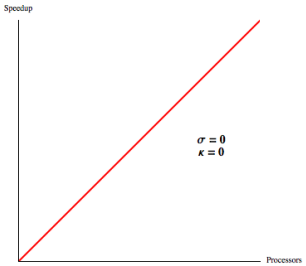
Cannot understand scalability by monitoring Prod systems. The human brain is not built for that. Need to transform time-series **data** to informational **metrics**.

Quantitative Scalability

- Google 2005 on MapReduce: *"If scaling were perfect, performance would be proportional to the number of machines. In our test, it was 0.98 of the machines."*



Translation: MR scalability is 98% of ideal linear scaling

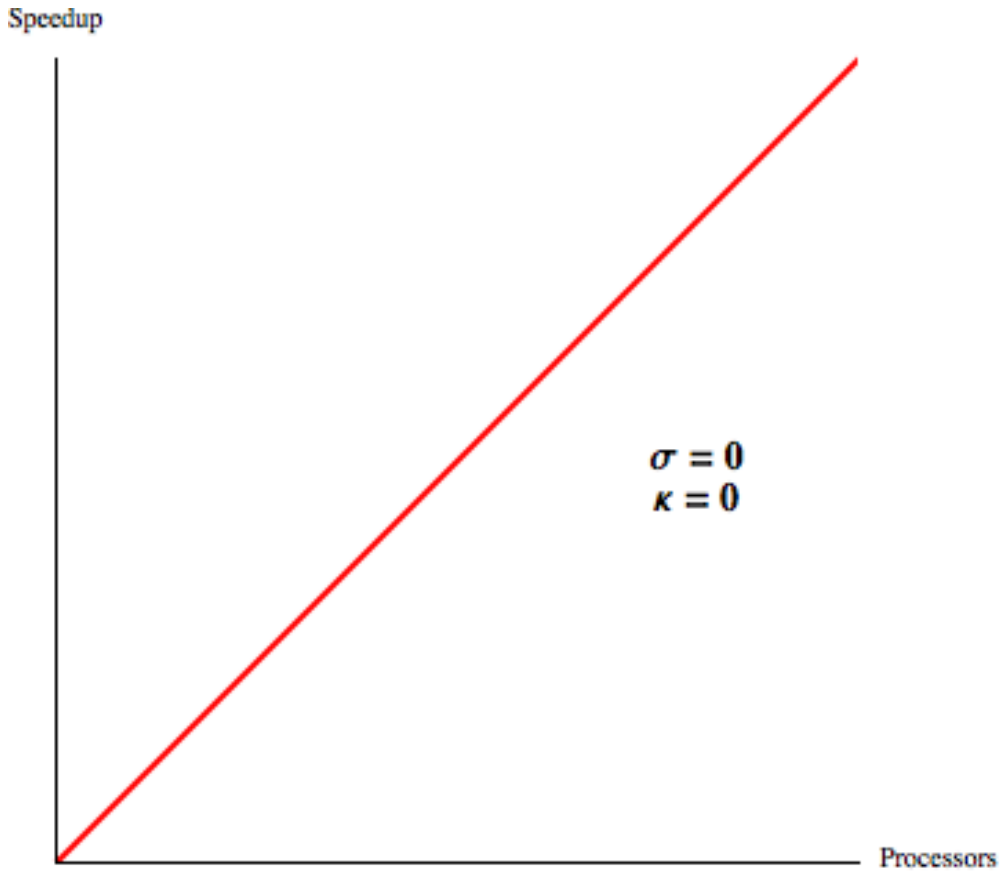


- Scalability is a function, not a single number
  - Diminishing returns due to increasing overhead
  - Want to express overhead loss quantitatively
  - But what (mathematical) function?

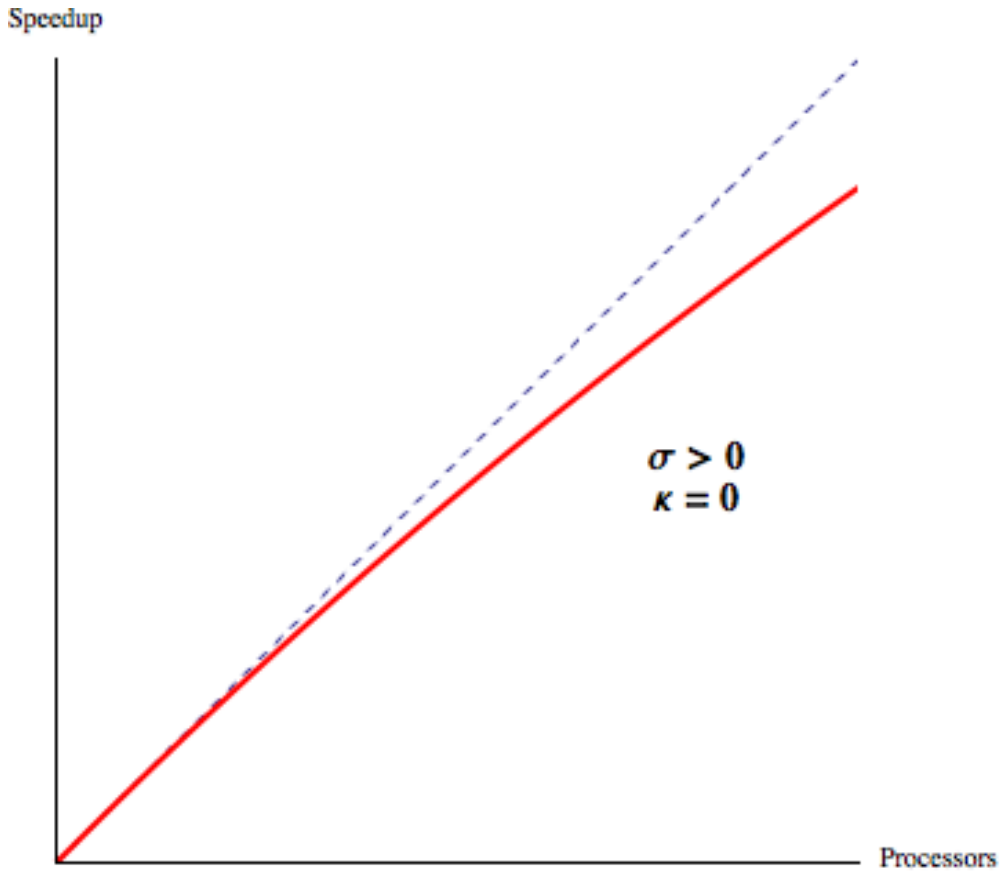
Outline

1. How to Quantify Scalability
   - Components of Scalability
   - Universal Scalability Law
2. Application of USL
   - Varnish
   - Memcached
   - Sirius
3. Superlinear Sizing
   - What it looks like
   - Perpetual motion
   - Hunting the Superlinear Snark
4. Summary

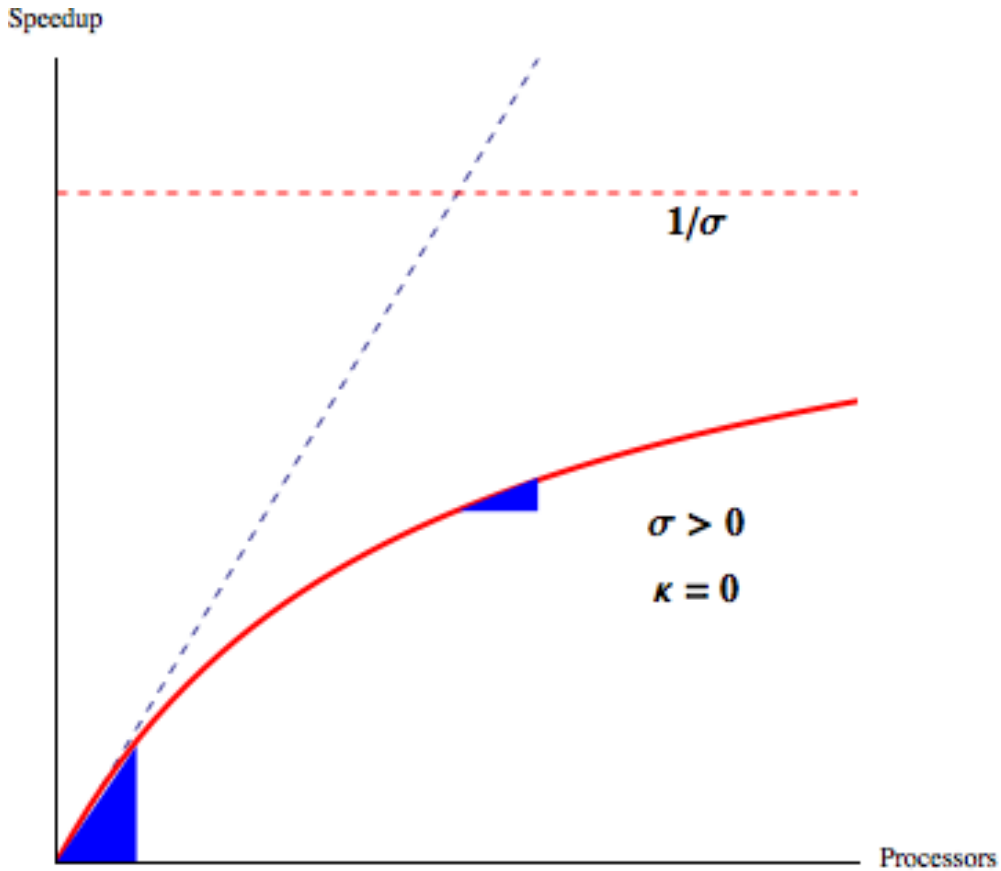
Equal bang for your buck: Concurrency



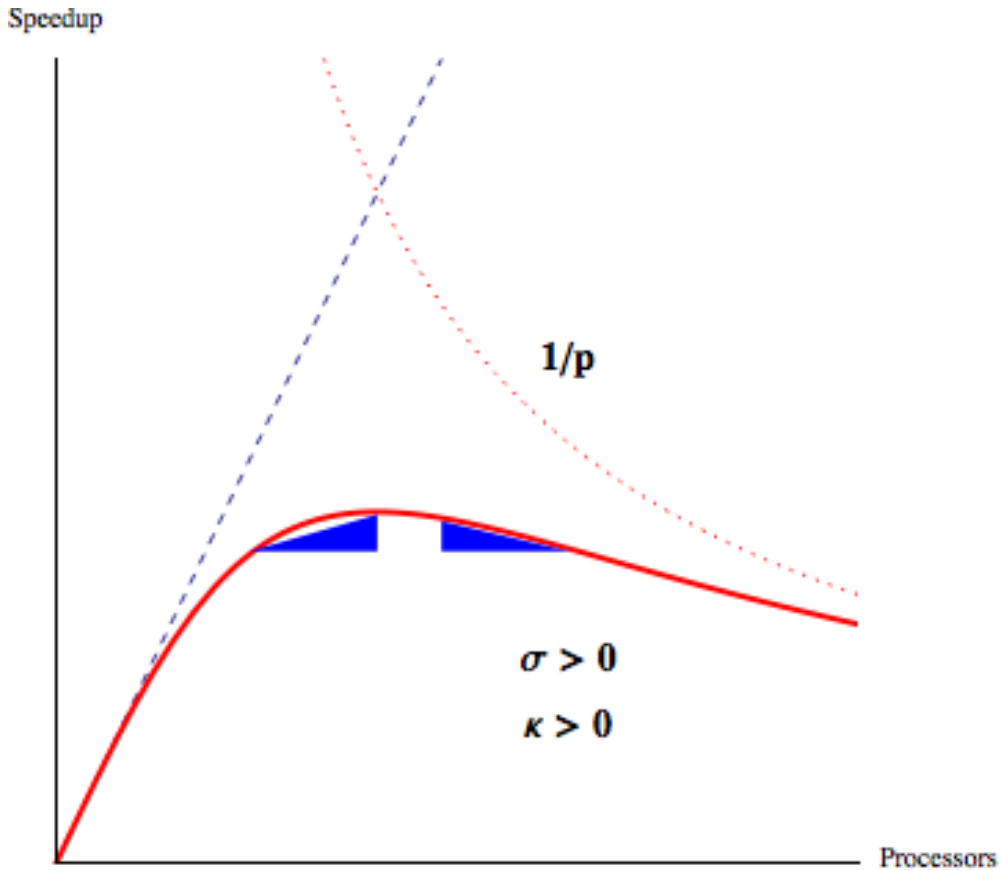
Diminishing returns: Contention cost



Resource saturation: More contention cost



Negative returns: Coherency of non-local data



Universal Scalability Law (USL)

p: processes or processors provide system **load**

[1] NJG. "A Simple Capacity Model of Massively Parallel Transaction Systems," *Computer Measurement Group Conf.* 1993

Universal Scalability Law (USL)

p: processes or processors provide system **load**

S_p: **speedup** response function $\equiv$ normalized thruput

[1] NJG. "A Simple Capacity Model of Massively Parallel Transaction Systems," *Computer Measurement Group Conf.* 1993

Universal Scalability Law (USL)

p: processes or processors provide system **load**

S_p: **speedup** response function $\equiv$ normalized thruput

Q: What kind of function?

[1] NJG. "A Simple Capacity Model of Massively Parallel Transaction Systems," *Computer Measurement Group Conf.* 1993

Universal Scalability Law (USL)

p: processes or processors provide system **load**

S_p: **speedup** response function $\equiv$ normalized thruput

Q: What kind of function?

A: Rational function [1]

[1] NJG. "A Simple Capacity Model of Massively Parallel Transaction Systems," *Computer Measurement Group Conf.* 1993

Universal Scalability Law (USL)

p: processes or processors provide system **load**

S_p: **speedup** response function $\equiv$ normalized thruput

Q: What kind of function?

A: Rational function [1]

$$S_p(\sigma, \kappa) = \frac{p}{1 + \sigma\,(p-1) + \kappa\,p(p-1)}$$

[1] NJG. "A Simple Capacity Model of Massively Parallel Transaction Systems," *Computer Measurement Group Conf.* 1993

Universal Scalability Law (USL)

p: processes or processors provide system **load**

S_p: **speedup** response function $\equiv$ normalized thruput

Q: What kind of function?

A: Rational function [1]

$$S_p(\sigma, \kappa) = \frac{p}{1 + \sigma\,(p-1) + \kappa\, p(p-1)}$$

The three Cs:

[1] NJG. "A Simple Capacity Model of Massively Parallel Transaction Systems," *Computer Measurement Group Conf.* 1993

Universal Scalability Law (USL)

p: processes or processors provide system **load**

S_p: **speedup** response function $\equiv$ normalized thruput

Q: What kind of function?

A: Rational function [1]

$$S_p(\sigma, \kappa) = \frac{p}{1 + \sigma\,(p-1) + \kappa\,p(p-1)}$$

The three Cs:

1. **C**oncurrency

[1] NJG. "A Simple Capacity Model of Massively Parallel Transaction Systems," *Computer Measurement Group Conf.* 1993

Universal Scalability Law (USL)

p: processes or processors provide system **load**

S_p: **speedup** response function $\equiv$ normalized thruput

Q: What kind of function?

A: Rational function [1]

$$S_p(\sigma, \kappa) = \frac{p}{1 + \sigma\,(p-1) + \kappa\, p(p-1)}$$

The three Cs:

1. **C**oncurrency
2. **C**ontention ($0 < \sigma < 1$)

[1] NJG. "A Simple Capacity Model of Massively Parallel Transaction Systems," *Computer Measurement Group Conf.* 1993

Universal Scalability Law (USL)

p: processes or processors provide system **load**

S_p: **speedup** response function $\equiv$ normalized thruput

Q: What kind of function?

A: Rational function [1]

$$S_p(\sigma, \kappa) = \frac{p}{1 + \sigma\,(p-1) + \kappa\,p(p-1)}$$

The three Cs:

1. **C**oncurrency
2. **C**ontention $(0 < \sigma < 1)$
3. **C**oherency $(0 < \kappa < 1)$

[1] NJG. "A Simple Capacity Model of Massively Parallel Transaction Systems," *Computer Measurement Group Conf.* 1993

Measurement Meets Model

$$\underbrace{\frac{X(p)}{X(1)}}_{\text{Thruput data}}$$

Measurement Meets Model

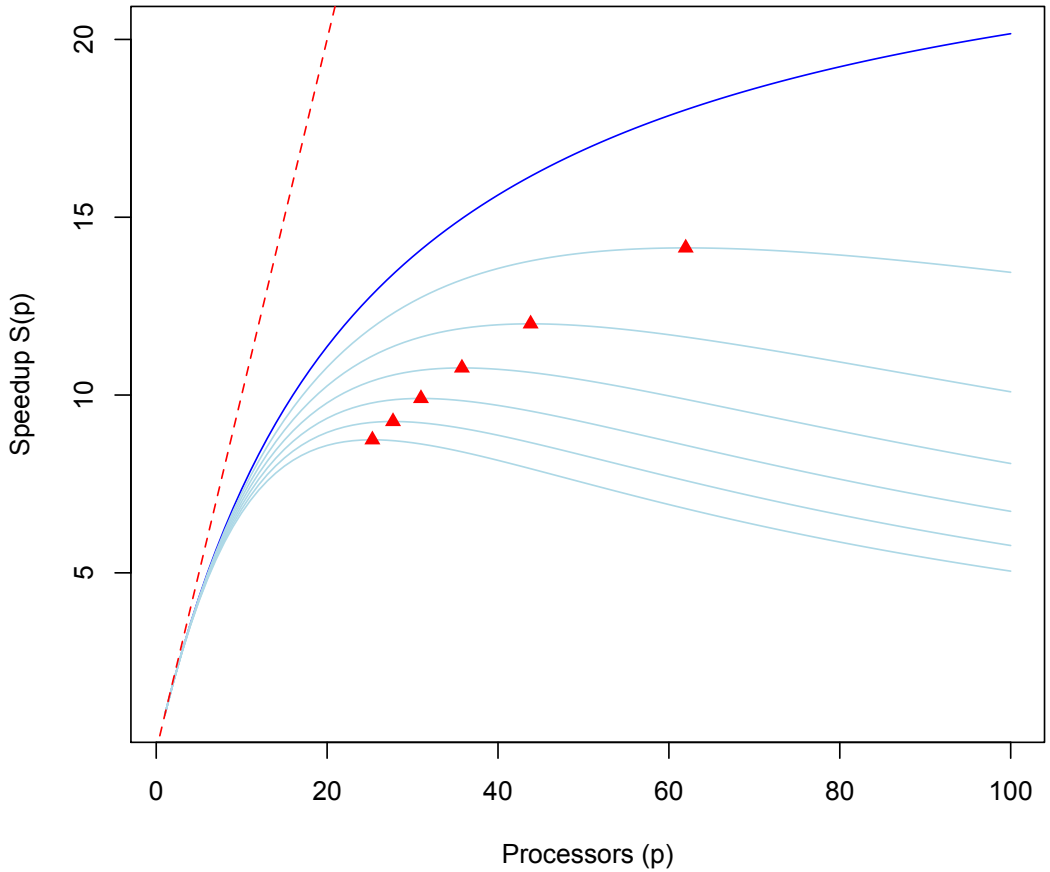
$$\underbrace{\frac{X(p)}{X(1)}}_{\text{Thruput data}} \longrightarrow \underbrace{S_p(\sigma, \kappa)}_{\text{Speedup}}$$

Measurement Meets Model

$$\underbrace{\frac{X(p)}{X(1)}}_{\text{Thruput data}} \longrightarrow \underbrace{S_p(\sigma, \kappa)}_{\text{Speedup}} \longleftarrow \underbrace{\frac{p}{1 + \sigma\,(p-1) + \kappa\,p(p-1)}}_{\text{USL model}}$$

Measurement Meets Model

$$\underbrace{\frac{X(p)}{X(1)}}_{\text{Thruput data}} \longrightarrow \underbrace{S_p(\sigma,\kappa)}_{\text{Speedup}} \longleftarrow \underbrace{\frac{p}{1+\sigma\,(p-1)+\kappa\,p(p-1)}}_{\text{USL model}}$$



How do we determine σ and κ?

$$S(p, \sigma, \kappa) = \frac{p}{1 + \sigma\,(p-1) + \kappa\,p(p-1)}$$

- Brute force measurements (good luck!)
- Data from controlled measurements, e.g., JMeter

How do we determine σ and κ?

$$S(p, \sigma, \kappa) = \frac{p}{1 + \sigma\,(p-1) + \kappa\,p(p-1)}$$

- Brute force measurements (good luck!)
- Data from controlled measurements, e.g., JMeter
- **Clever way:** Apply statistical regression

How do we determine σ and κ?

$$S(p, \sigma, \kappa) = \frac{p}{1 + \sigma\,(p-1) + \kappa\,p(p-1)}$$

- Brute force measurements (good luck!)
- Data from controlled measurements, e.g., JMeter
- **Clever way:** Apply statistical regression
- I'll use R stats tools throughout this talk:
  - FOSS codes with 40 yr history since **S** at Bell Labs
  - See **LISA14 Tutorial S6** *"R for Sysadmins"*

How do we determine σ and κ?

$$S(p, \sigma, \kappa) = \frac{p}{1 + \sigma\,(p-1) + \kappa\,p(p-1)}$$

- Brute force measurements (good luck!)
- Data from controlled measurements, e.g., JMeter
- **Clever way:** Apply statistical regression
- I'll use R stats tools throughout this talk:
  - FOSS codes with 40 yr history since **S** at Bell Labs
  - See **LISA14 Tutorial S6** *"R for Sysadmins"*
- Magic functions in **R**:
  - `nls()` nonlinear regression $\rightarrow \sigma, \kappa$ in one swell foop
  - `optimize()` to estimate $X_{data}(1)$ if missing
  - `predict()` smooth interpolation/extrapolation from data
  - `plot()` with its bells & whistles

Outline

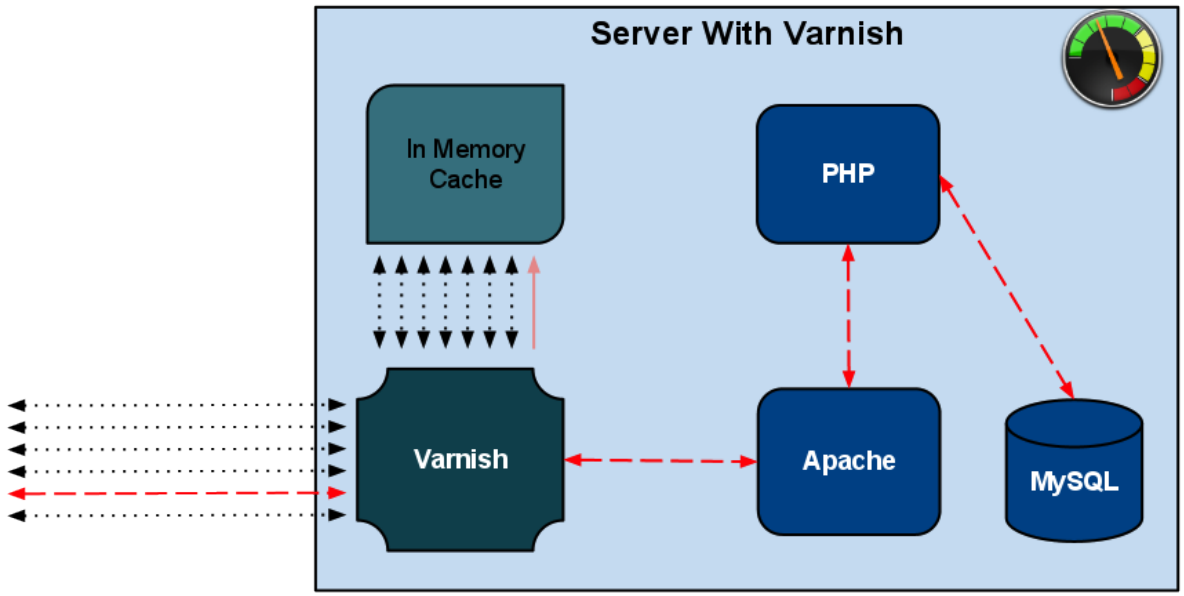
1. How to Quantify Scalability
   - Components of Scalability
   - Universal Scalability Law
2. Application of USL
   - Varnish
   - Memcached
   - Sirius
3. Superlinear Sizing
   - What it looks like
   - Perpetual motion
   - Hunting the Superlinear Snark
4. Summary

Varnish

Data by D. Popa *(DigitAir, RO)* via S. Parvu *(Nokia, FI)*

Varnish Architecture

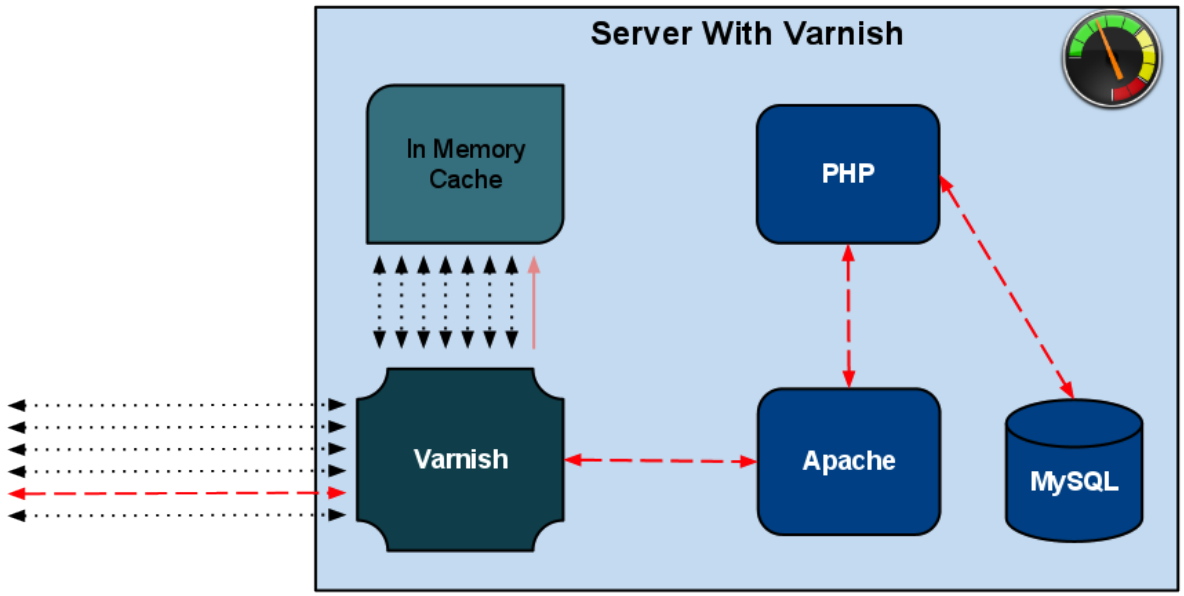
- HTTP accelerator
- Reverse web proxy caching system
- Sits in front of classic web server



- Caching handled by virtual memory
- **Claim:** Highly scalable (read: linear)

Varnish Architecture

- HTTP accelerator
- Reverse web proxy caching system
- Sits in front of classic web server

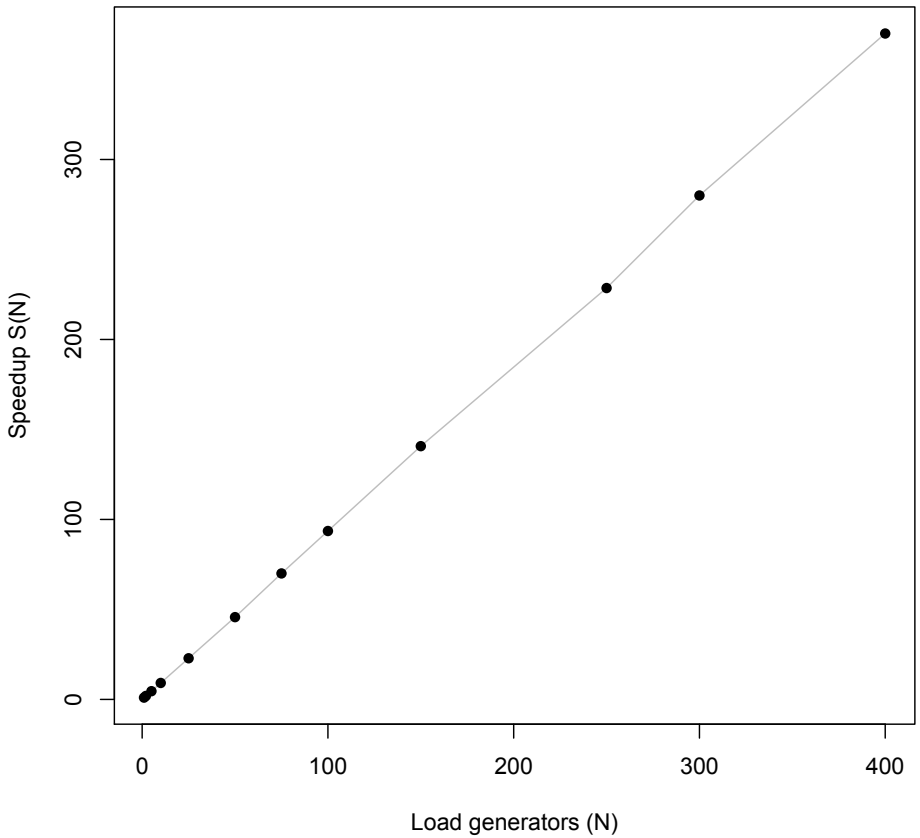


- Caching handled by virtual memory
- **Claim:** Highly scalable (read: linear) ... but is it?

Varnish Jmeter Measurements

Example (Read in raw data and plot it in R)

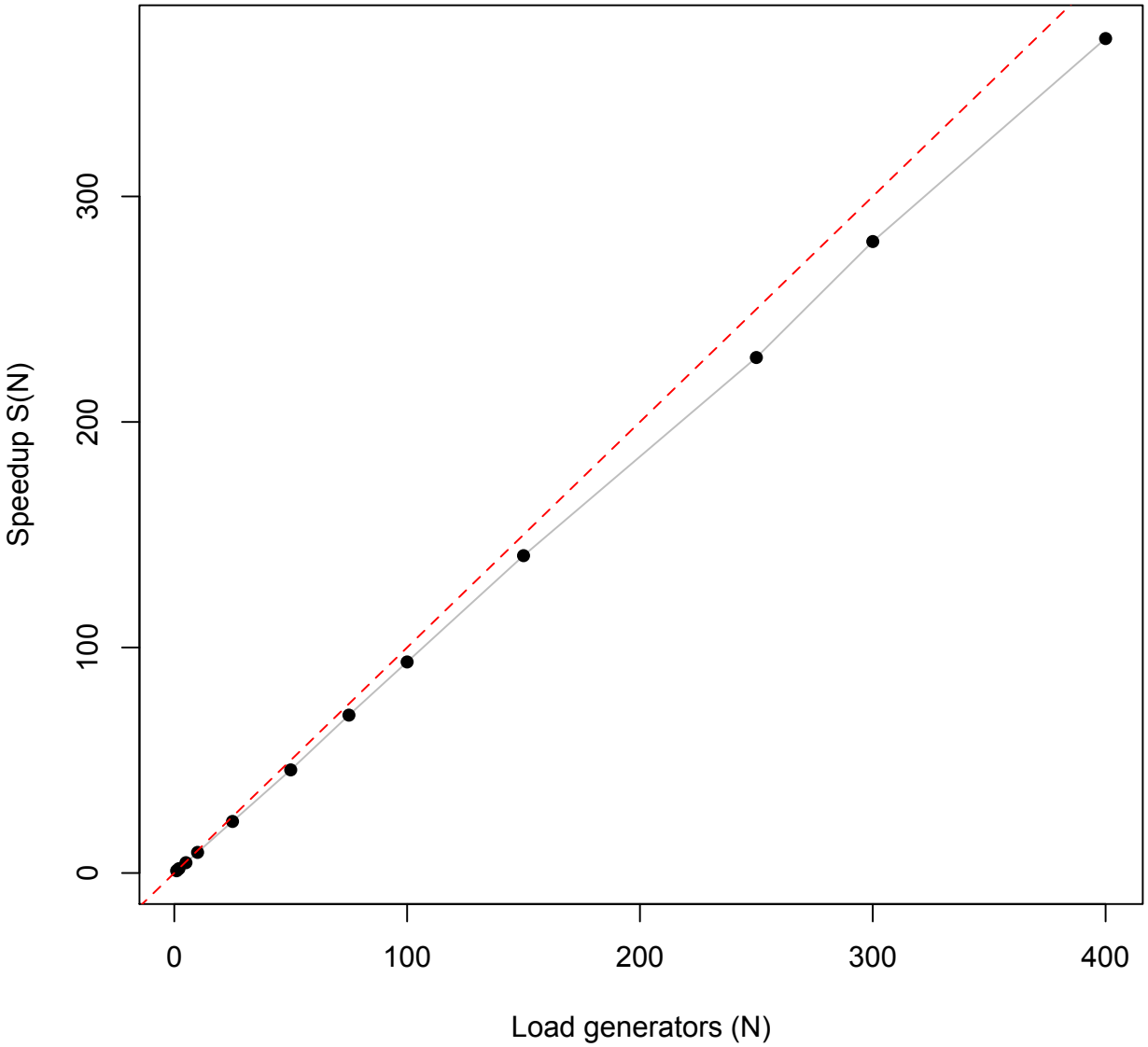
```
data <- read.table(fname,header=TRUE,sep="\t")
print(data)
plot(data$N,data$X_N,type="b")
```



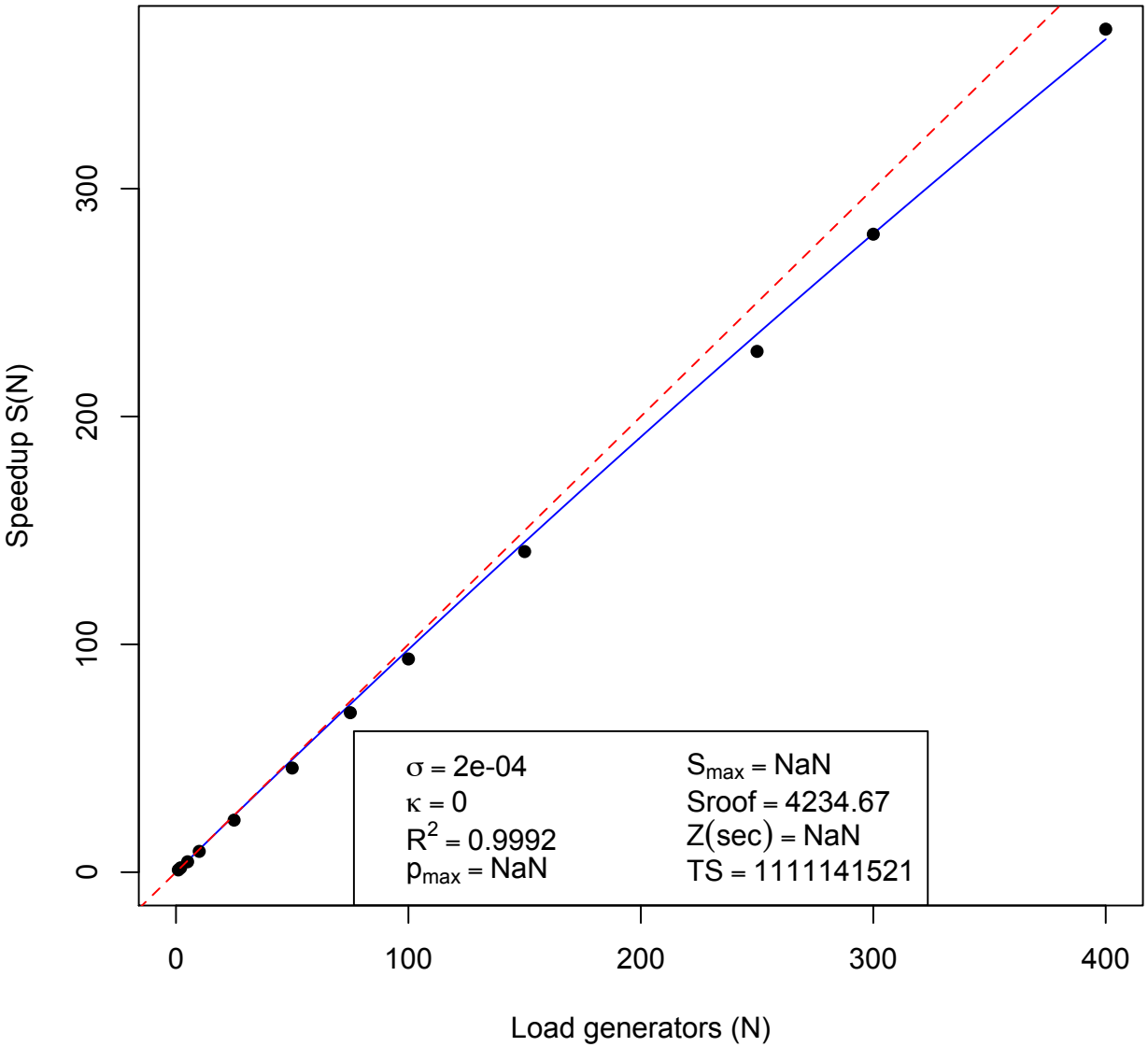
By typing `data` into R console:

```
> data
     N    X_N       Speed      Effcy
1    1    1.4    1.000000 1.0000000
2    2    2.7    1.928571 0.9642857
3    5    6.4    4.571429 0.9142857
4   10   12.8    9.142857 0.9142857
5   25   32.0   22.857143 0.9142857
6   50   64.0   45.714286 0.9142857
7   75   98.0   70.000000 0.9333333
8  100  131.0   93.571429 0.9357143
9  150  197.0  140.714286 0.9380952
10 250  320.0  228.571429 0.9142857
11 300  392.0  280.000000 0.9333333
12 400  518.0  370.000000 0.9250000
```

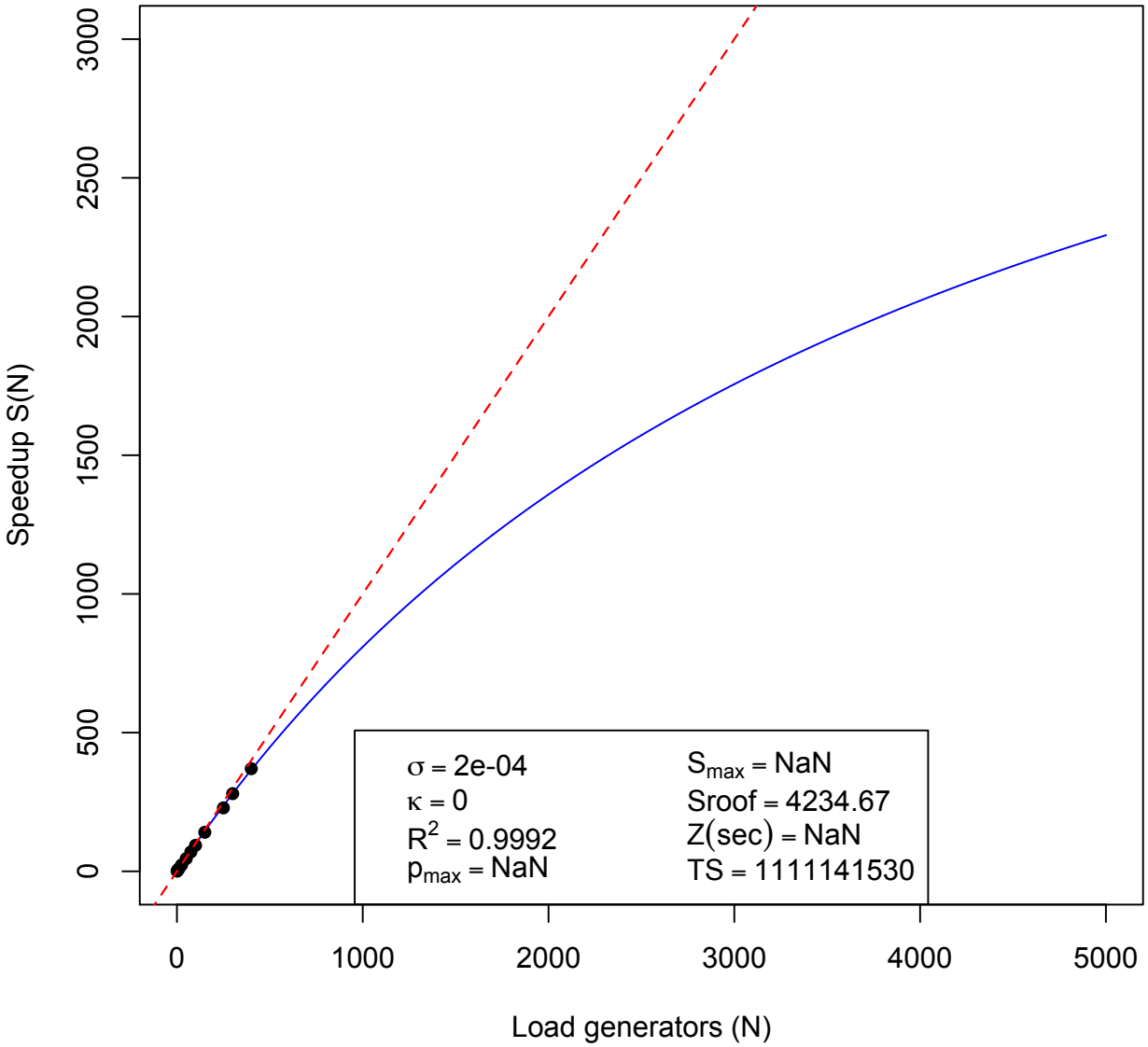
Varnish Comparison with Linear Scaling



Varnish Measurements Meet the USL Model



Varnish USL Scalability Projection



Memcached

Joint work with

S. Subramanyam *(Sun, USA)* and S. Parvu *(Sun, FI)*

Presented at **Velocity 2010**

Memcached Scalability

Scaleup



Scaleout



Memcached Scaleout Strategy

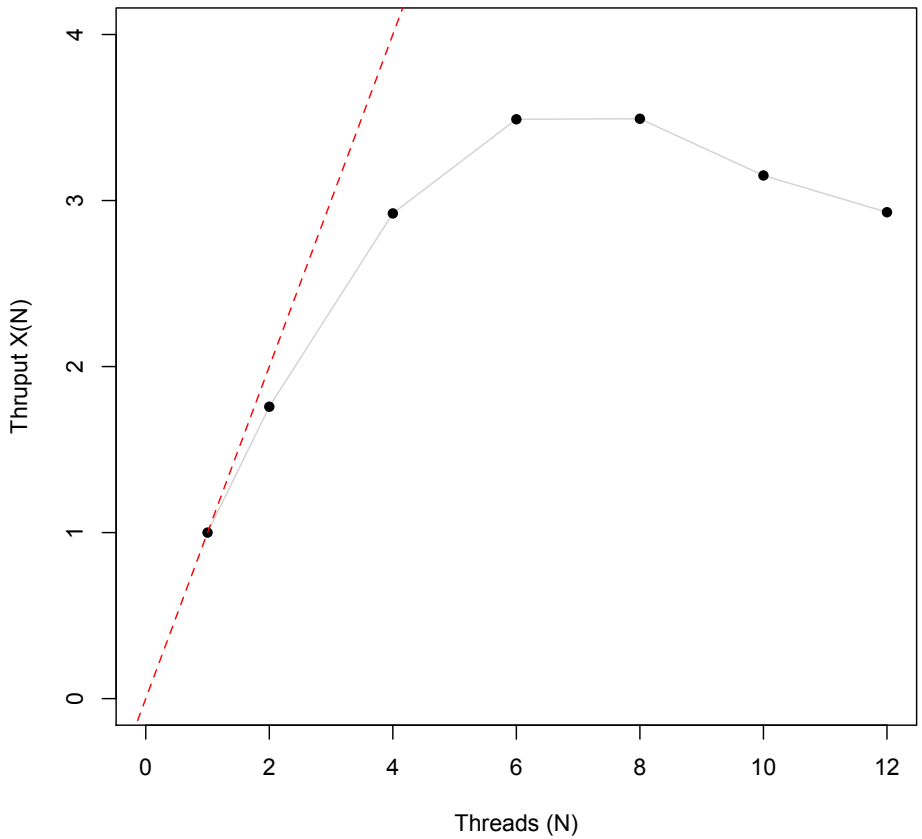


- Distributed cache of key-value pairs
- Pre-loaded from RDBMS
- Deploy mcd on tier of cheap, older CPUs (but not multicores)
- Single threaded mcd ok — until next hardware roll (i.e., multicores)

Memcached Measurements

Example (Read in raw data and plot it)

```
data <- read.table(fname,header=TRUE,sep="\t")
print(data)
plot(data$N,data$X_N,type="b")
```



Typing `data` into R console:

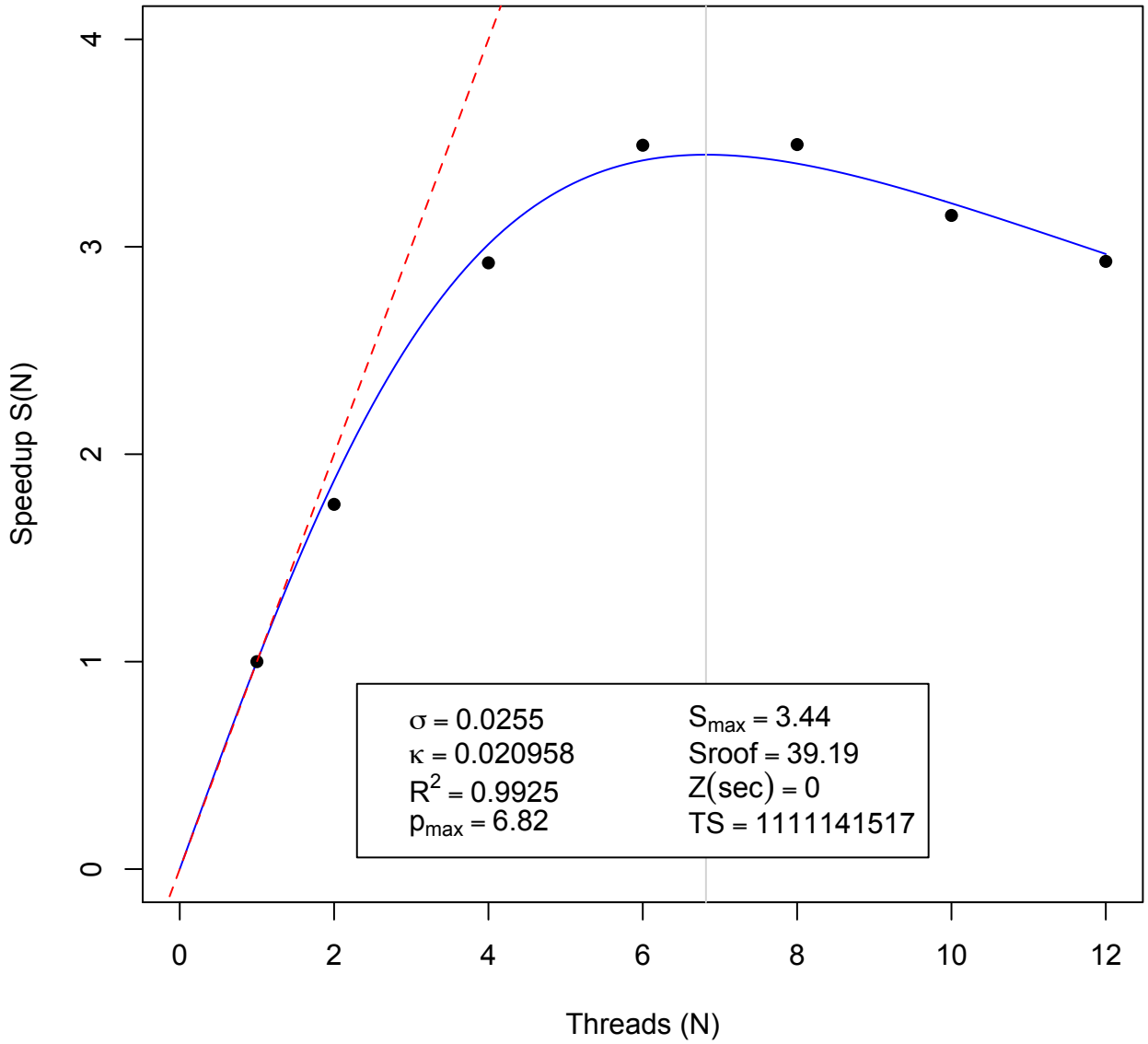
```
> data
   N X_N
1  1  89
2  2 160
3  4 272
4  8 333
5 10 352
6 12 339
7 14 315
```

Memcached Regression Analysis

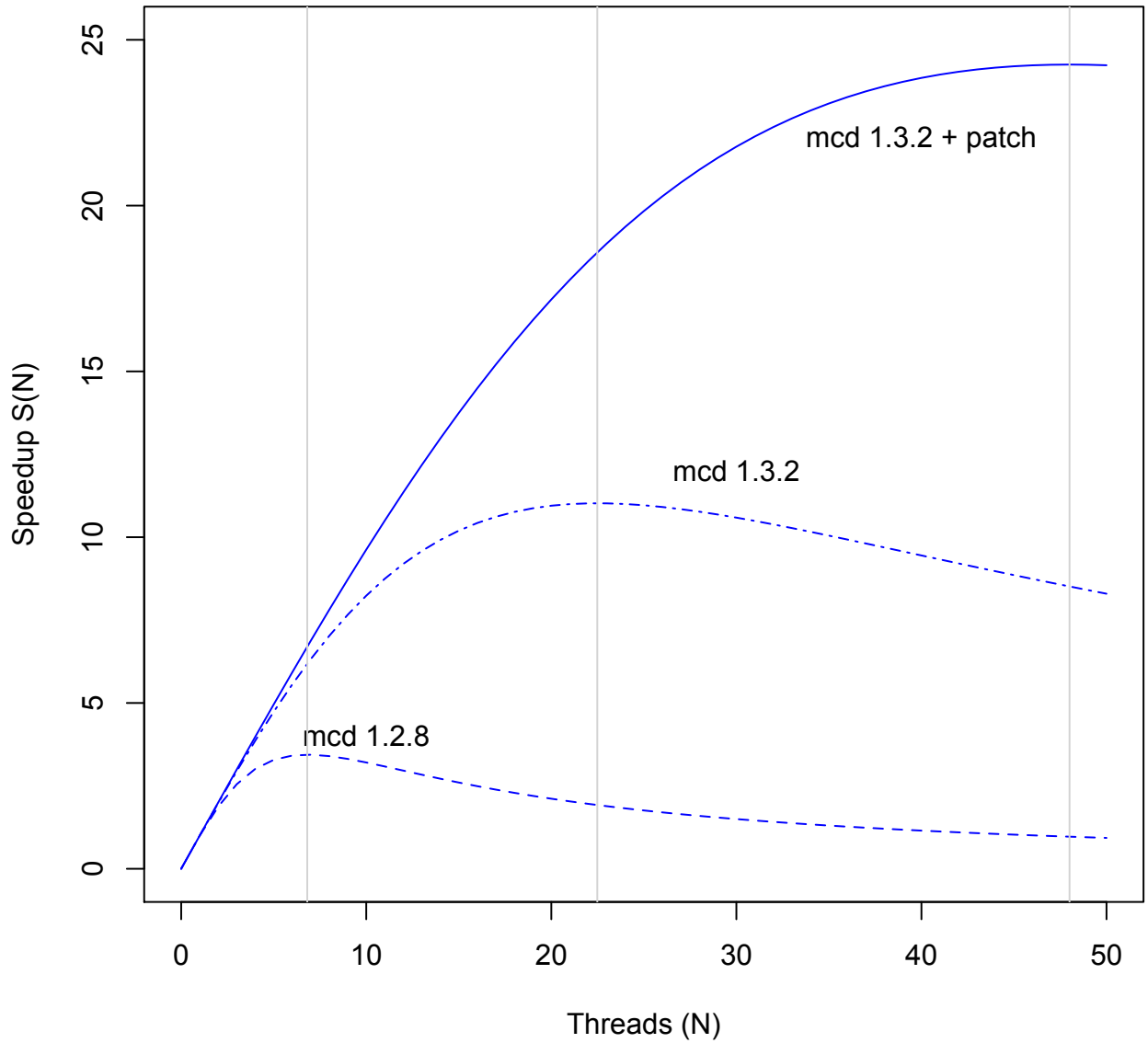
Example (Normalize, check efficiencies, USL fit)

```
> data
   p X_p    Speed     Effcy
1  1  89 1.000000 1.0000000
2  2 160 1.797753 0.8988764
3  4 272 3.056180 0.7640449
4  8 333 3.741573 0.4676966
5 10 352 3.955056 0.3955056
6 12 339 3.808989 0.3174157
7 14 315 3.539326 0.2528090

> summary(usl)

Formula: Speed ~ p/(1 + sigma * (p - 1) + kappa * p * (p - 1))

Parameters:
      Estimate Std. Error t value Pr(>|t|)
sigma 0.025517   0.014830   1.721    0.146
kappa 0.020958   0.001746  12.003 7.08e-05 ***
---
Signif. codes:  0 ?***? 0.001 ?**? 0.01 ?*? 0.05 ?.? 0.1 ? ? 1

Residual standard error: 0.08918 on 5 degrees of freedom

Algorithm "port", convergence message: relative convergence (4)
```

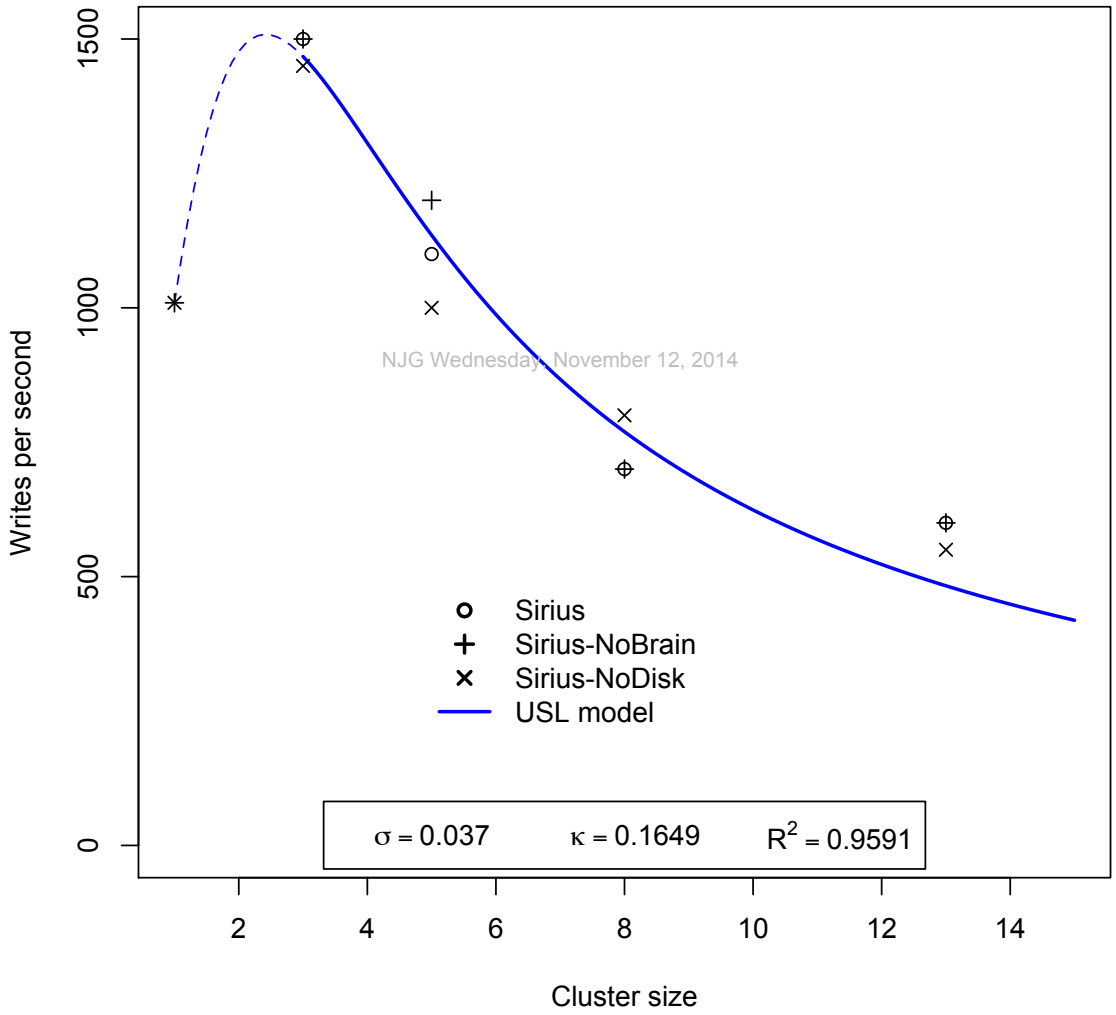
Memcached USL Scalability Fit



MCD Scalability Improvements (Sun SPARC patch)



Sirius: Distributed coordination by voting



See Figure 4 in USENIX ;login: October 2014, p. 23 (cf. Apache **Zookeeper**)

Outline

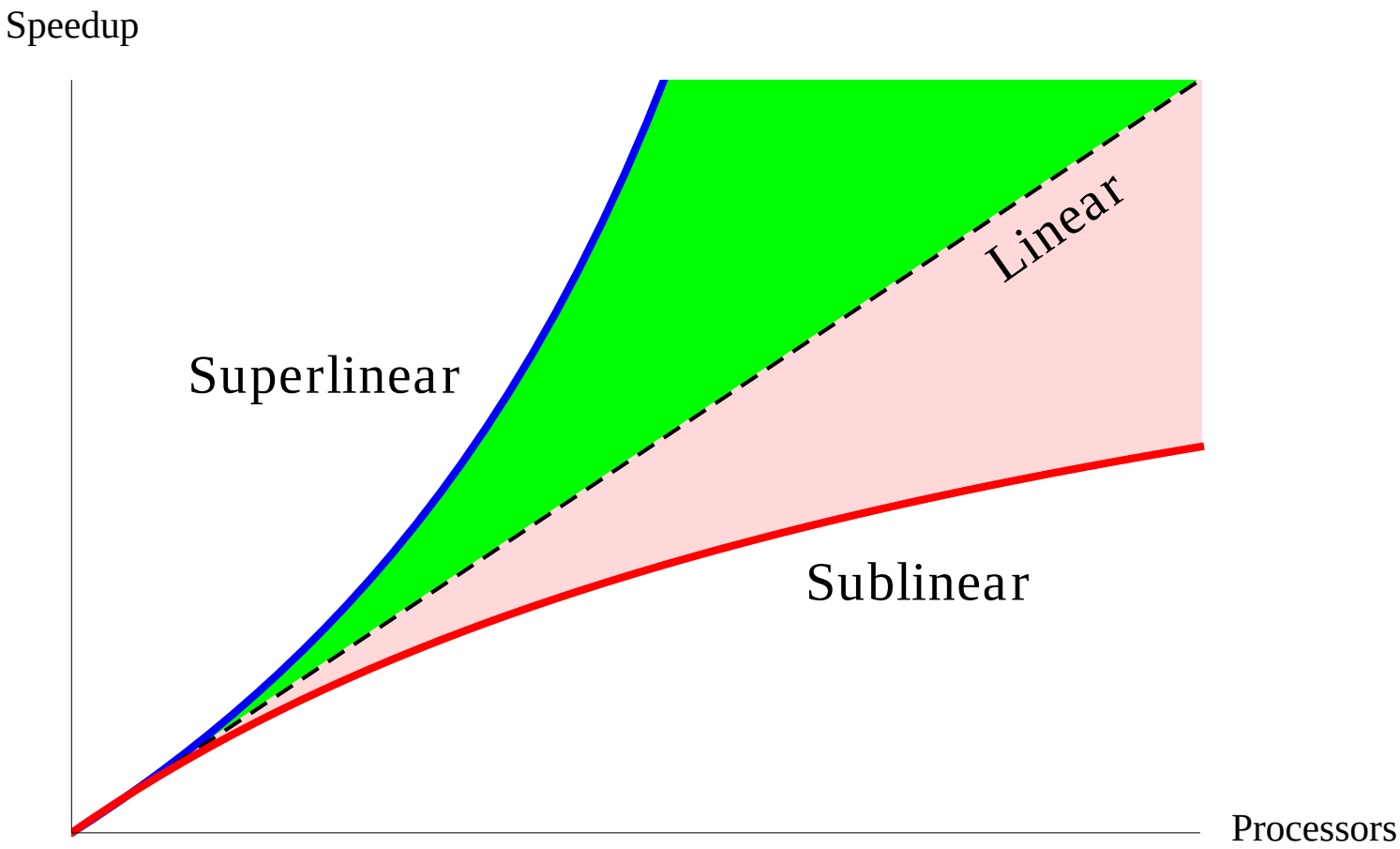
1. How to Quantify Scalability
   - Components of Scalability
   - Universal Scalability Law
2. Application of USL
   - Varnish
   - Memcached
   - Sirius
3. Superlinear Sizing
   - What it looks like
   - Perpetual motion
   - Hunting the Superlinear Snark
4. Summary

Superlinear Sizing

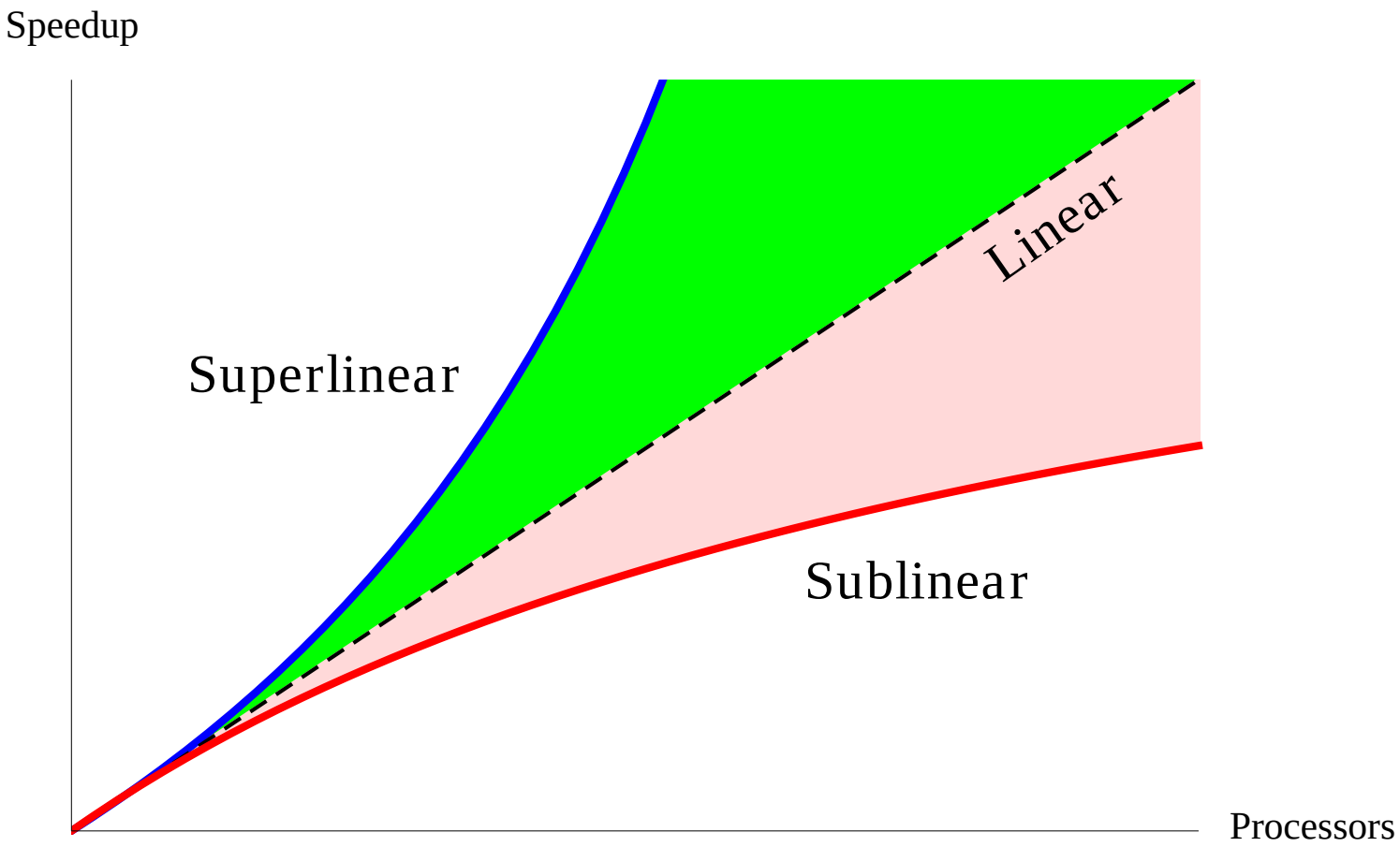
Joint work with

P. Puglia and K. Tomasette

Remember this?



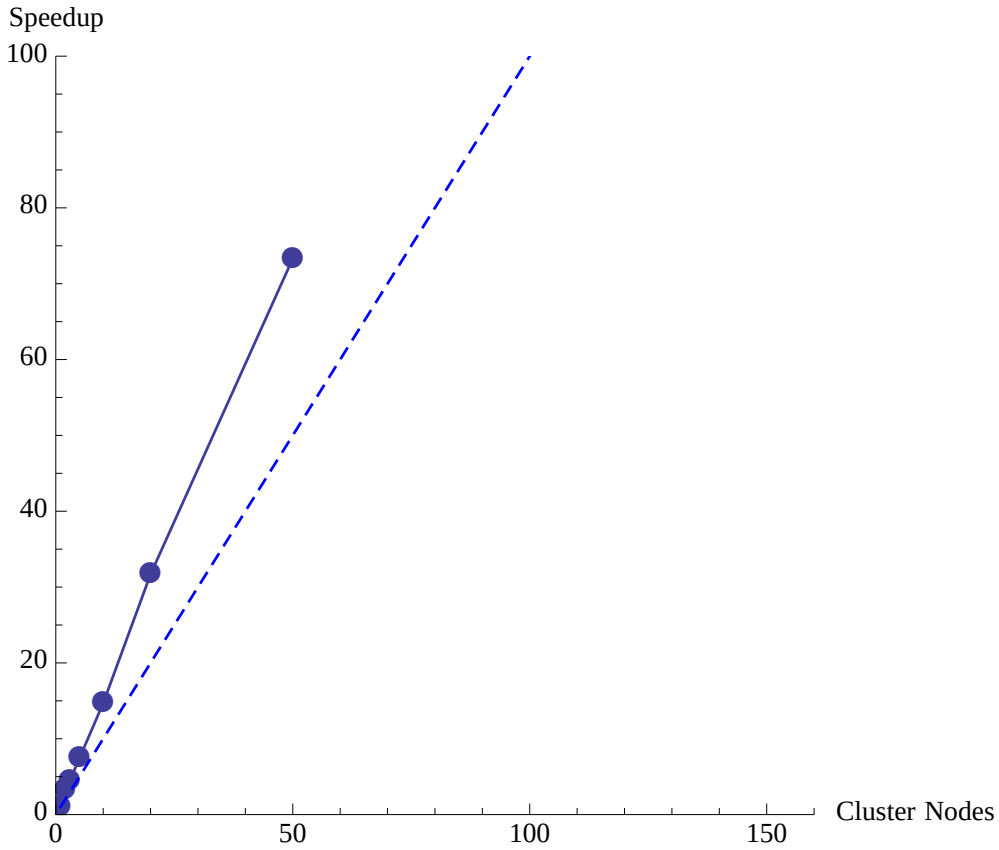
Remember this?



Blue curve is more than 100% efficient (WTW!??)

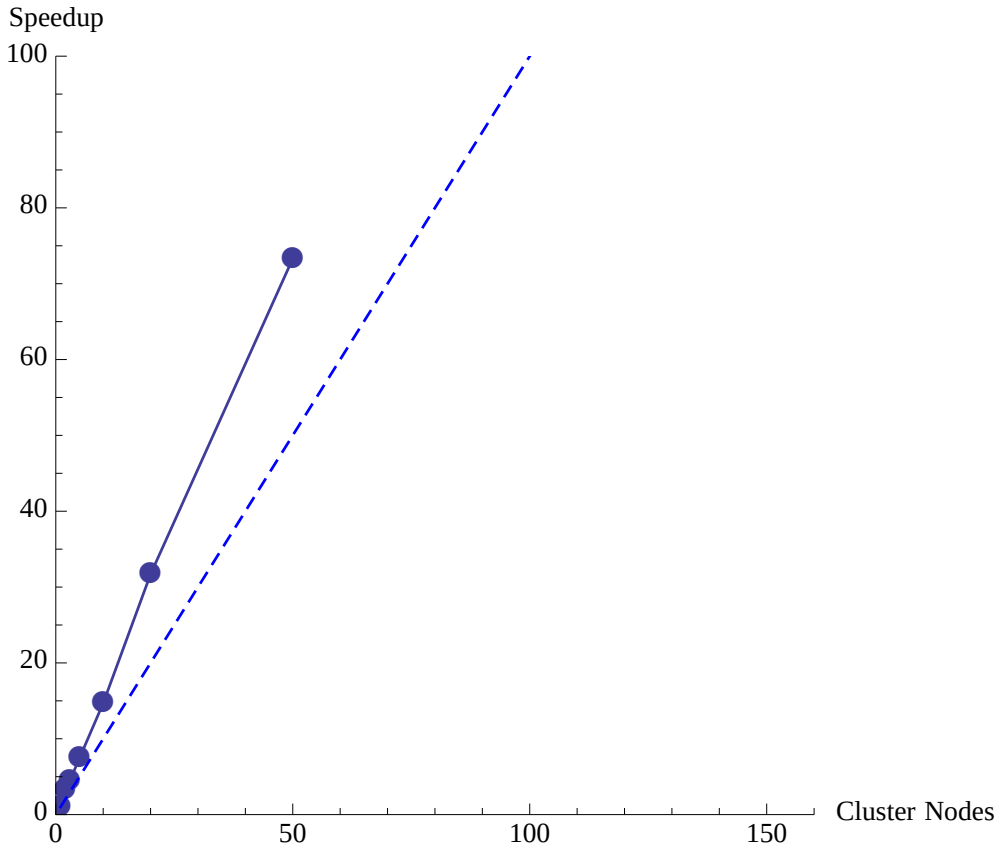
- "Speedup in Parallel Contexts."
  http://en.wikipedia.org/wiki/Speedup#Speedup_in_Parallel_Contexts
- "Where does super-linear speedup come from?"
  http://stackoverflow.com/questions/4332967/where-does-super-linear-speedup-come-from
- "Sun Fire X2270 M2 Super-Linear Scaling of Hadoop TeraSort and CloudBurst Benchmarks."
  https://blogs.oracle.com/BestPerf/entry/20090920_x2270m2_hadoop
- Haas, R. "Scalability, in Graphical Form, Analyzed."
  http://rhaas.blogspot.com/2011/09/scalability-in-graphical-form-analyzed.html
- Sutter, H. 2008. "Going Superlinear."
  Dr. Dobb's Journal 33(3), March.
  http://www.drdobbs.com/cpp/going-superlinear/206100542
- Sutter, H. 2008. "Super Linearity and the Bigger Machine."
  Dr. Dobb's Journal 33(4), April.
  http://www.drdobbs.com/parallel/super-linearity-and-the-bigger-machine/206903306
- "SDN analytics and control using sFlow standard — Superlinear."
  http://blog.sflow.com/2010/09/superlinear.html
- Eijkhout, V. 2014. *Introduction to High Performance Scientific Computing*. Lulu.com

Superlinear Hadoop Application

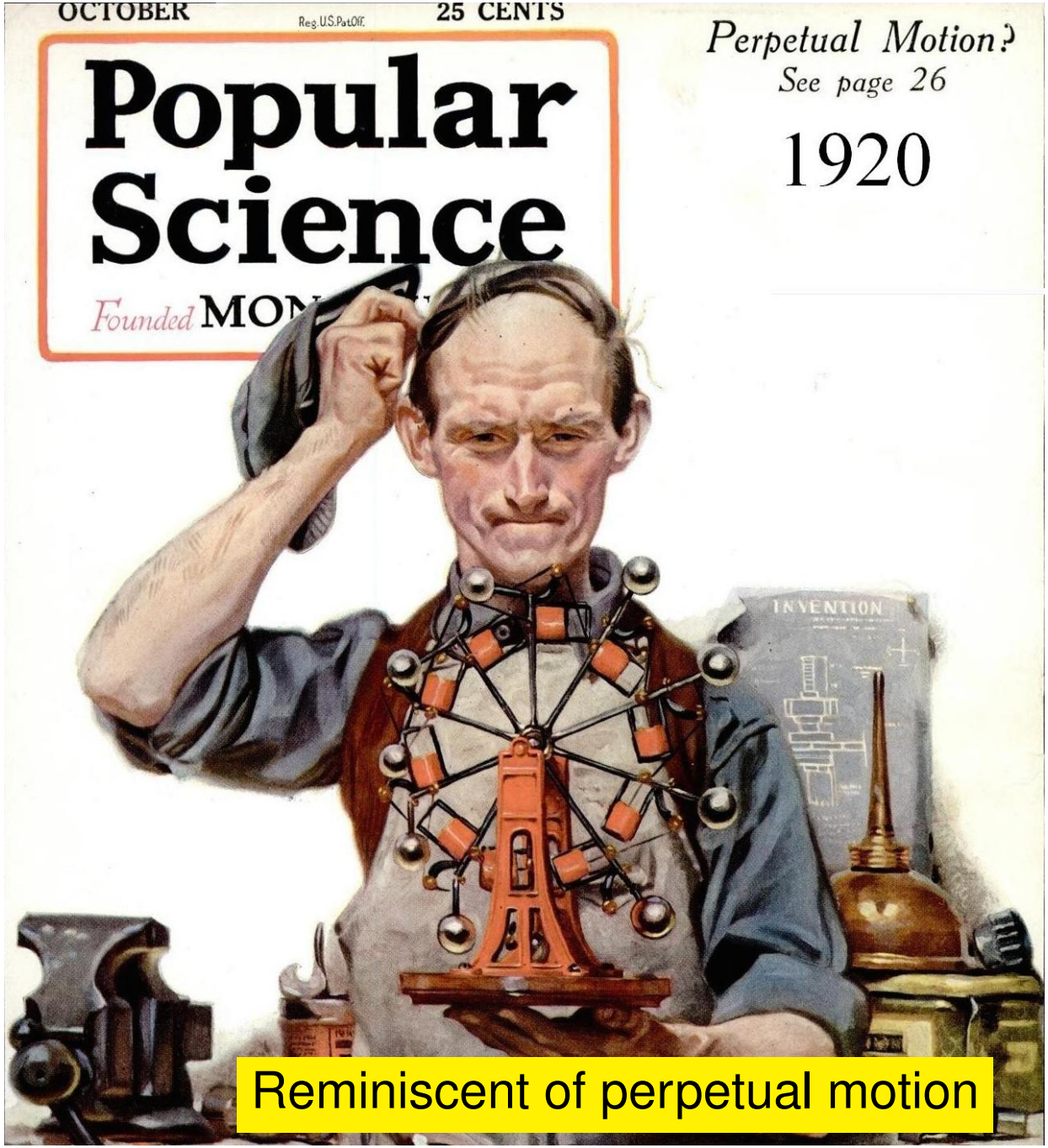


Faster than linear data!

Superlinear Hadoop Application



Faster than linear data! ← Ship it !!!



Reminiscent of perpetual motion

Perpetual motion

Perpetual motion machines:

- Perpetual motion contraptions violate *conservation of energy* law.
- Super efficiency is tantamount to getting more than 100% output.
- Even if you believe it's wrong, **proving** it is the hard part.

Superlinear scalability:

- Superlinear scaling exhibits measured speedup performance that **exceeds** 100% of available capacity.
- Violates the Universal Scalability Law (USL).
- Again, **proving** that it's wrong is the hard part.
- Takes some serious system **debugging**.

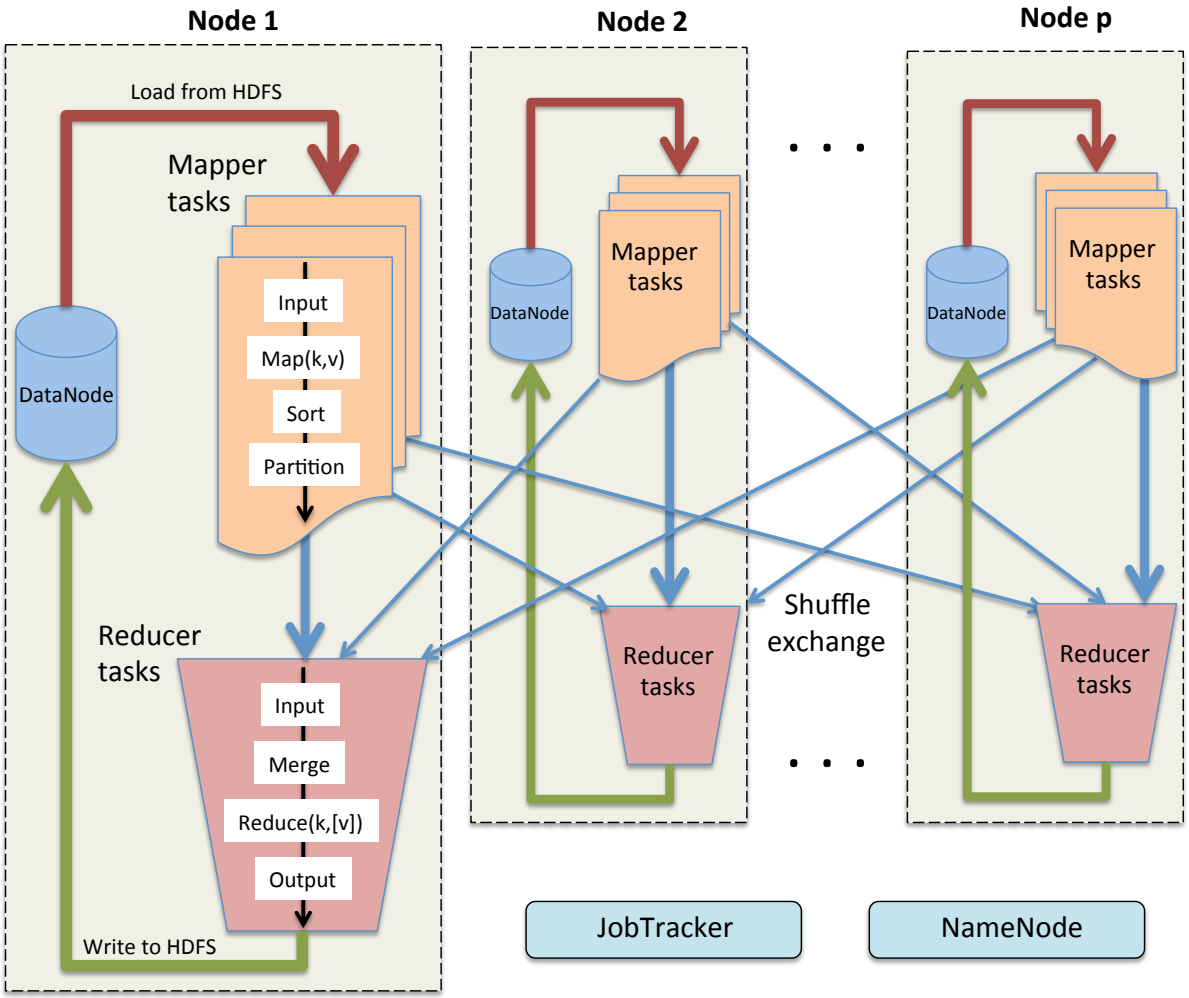
TeraSort Cluster Simulations

- Want a controlled environment to study superlinearity
- **TeraSort** workload sorts 1 TB data in parallel
- TeraSort has benchmarked Hadoop **MapReduce** performance
- We used just 100 GB data input (not benchmarking anything)
- Simulate in **AWS cloud** (more flexible and much cheaper)
- Many runs, some done in **parallel**

Table 1 : Amazon EC2 Configurations

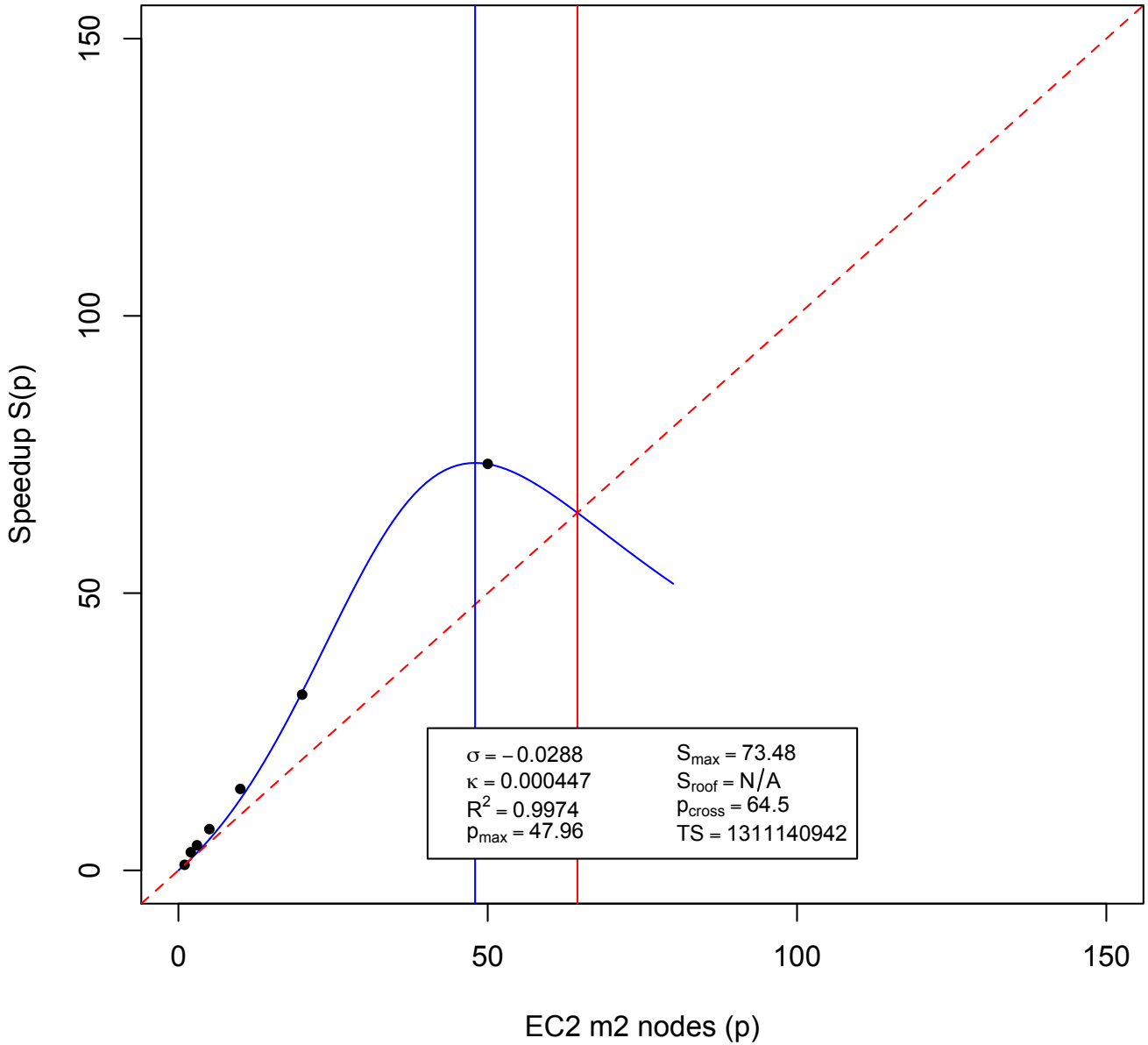
	Optimized for	Processor Arch	vCPU number	Memory (GiB)	Instance Storage (GB)	Network Performance
BigMem	Memory	64-bit	4	34.2	1 x 850	Moderate
BigDisk	Compute	64-bit	8	7	4 x 420	High

Hadoop MapReduce Architecture

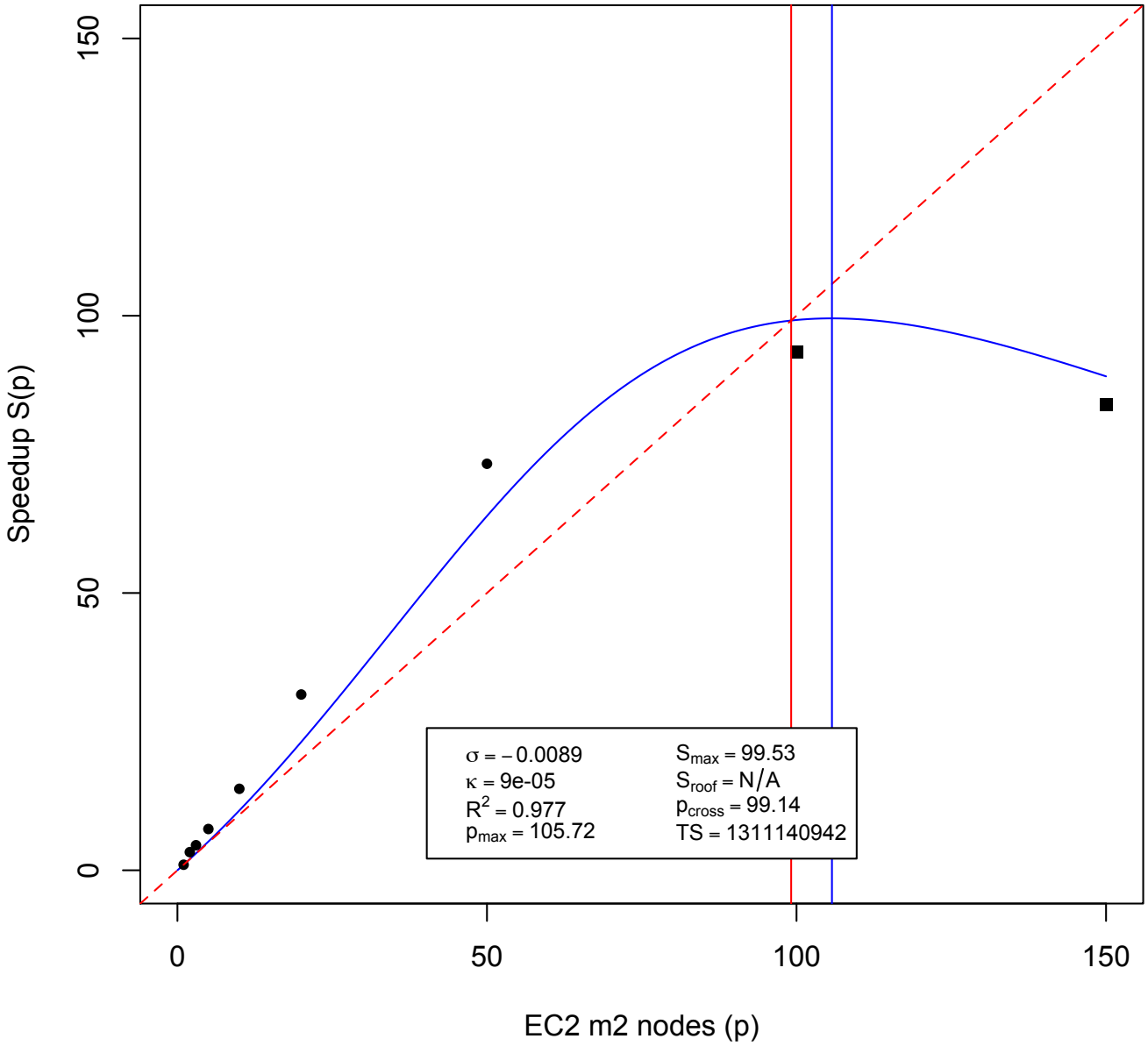


- 100 GB input data
- 840 Mappers
- 3 Reducers/node
- EC2 nodes: $p = 1 \ldots 200$
- $p = 1$ runtimes $\sim$ 5 hrs
- Cloudera 4.7.0 dsn
- Apache Whirr
- Linux perf tools

USL Model of BigMem $p = 50$ Speedup



USL Model of BigMem $p = 150$ Speedup



Signs of Superlinearity

- **Negative** USL contention coefficient:
  - $\sigma = -0.0288$
  - $\sigma = -0.0089$
- The sign that suplinear scaling is really there
- Positive σ means capacity **loss** due to overhead
- Negative σ means capacity **boost**
- But what could provide that boost?

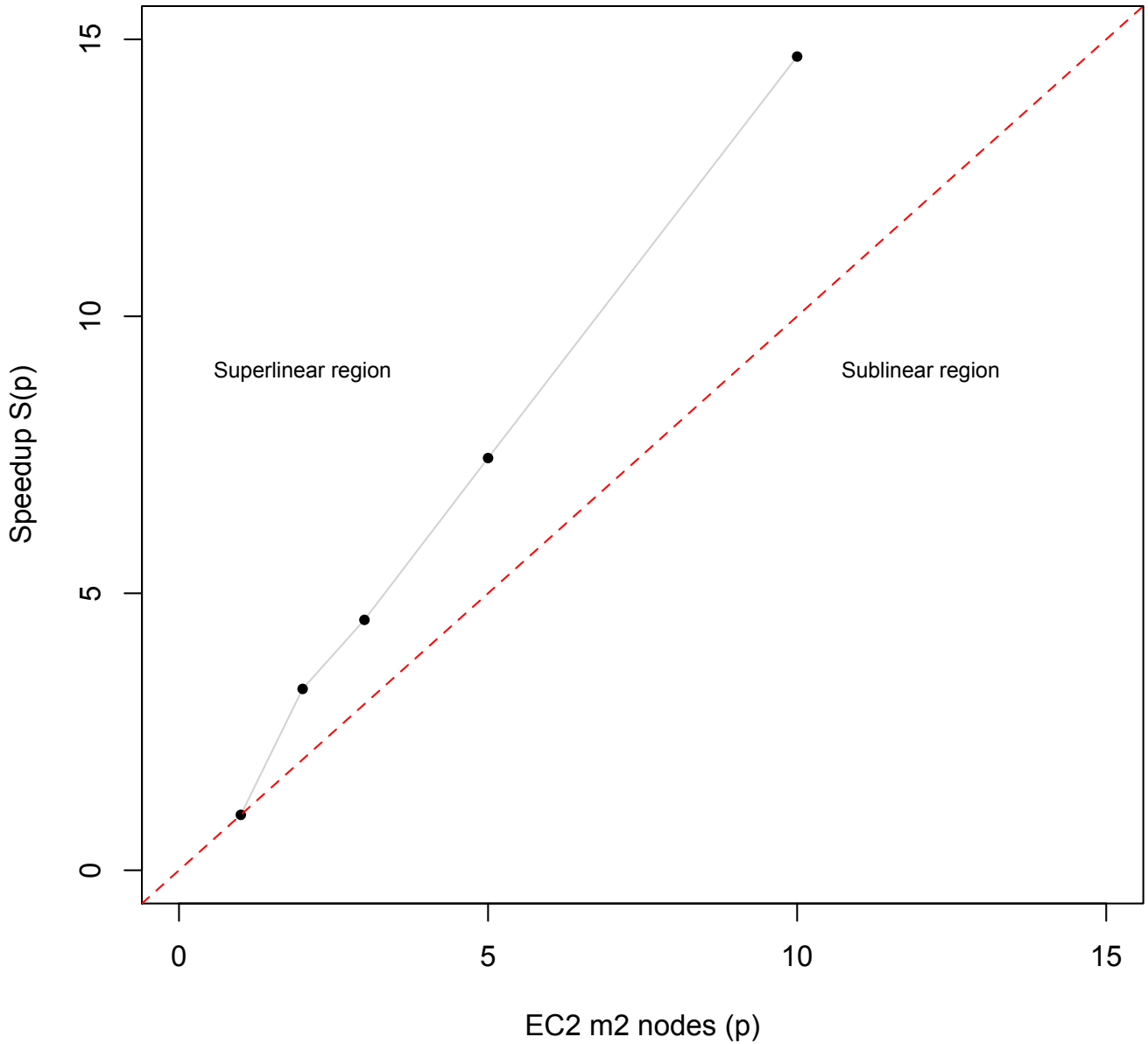
Recall from AMZ EC2 Configs

	Optimized for	Processor Arch	vCPU number	Memory (GiB)	Instance Storage (GB)	Network Performance
BigMem	Memory	64-bit	4	34.2	1 x 850	Moderate
BigDisk	Compute	64-bit	8	7	4 x 420	High

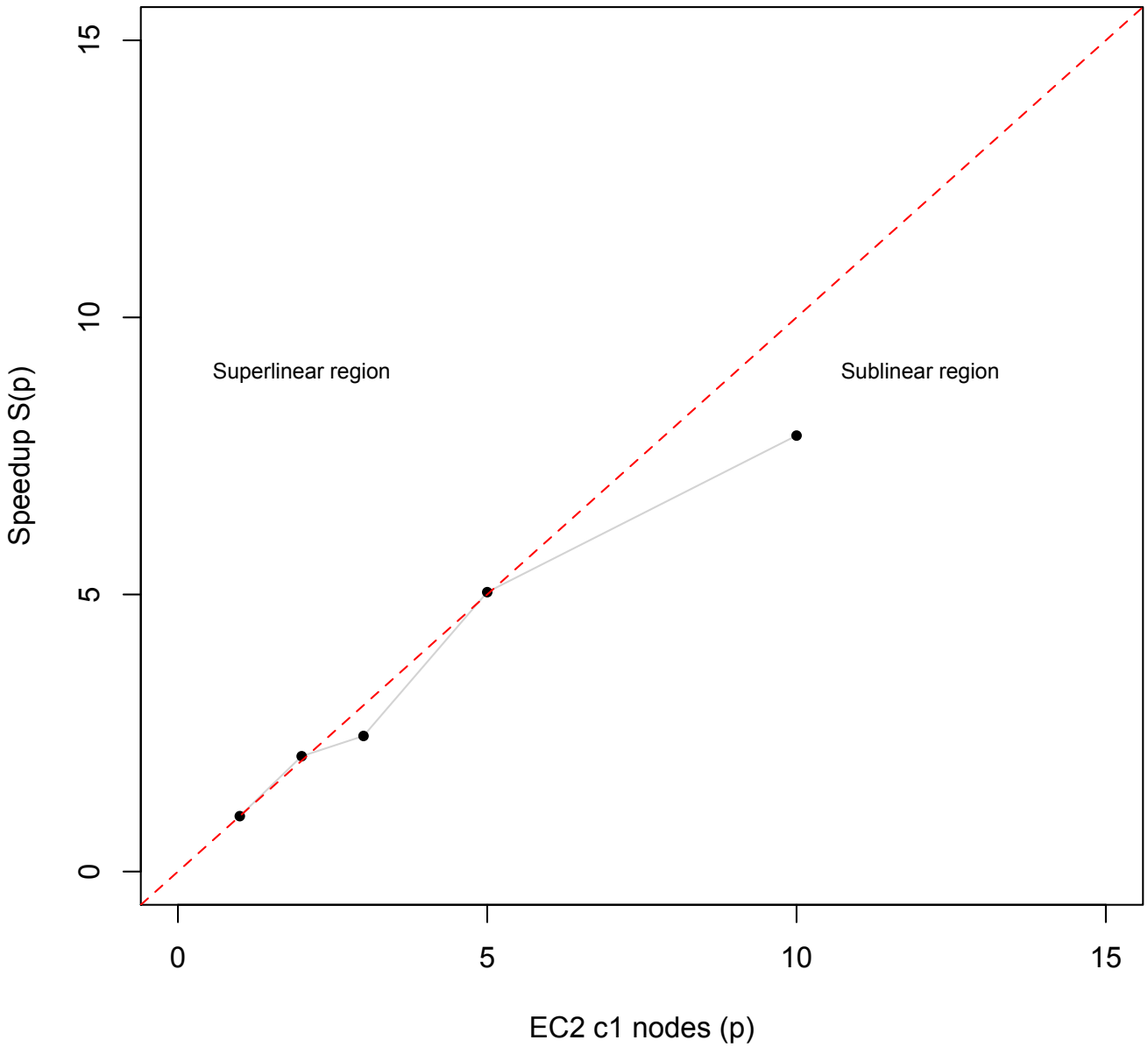
From Table 1:

1. *BigMem* has a 1 disk per EC2 node type
2. *BigDisk* has 4 disks per EC2 node type

Speedup Data on $p = 10$ BigMem Nodes (1 disk)



Speedup Data on $p = 10$ BigDisk Nodes (4 disks)



Brief Explanation

1. Apparent capacity boost produced by IO bottleneck:
   - Boost $\equiv$ Gradual reduction in IO constraint
   - Relaxation of the latent IO bandwidth constraint.
   - Constraint decreases with cluster size $p = 1, 2, 3, \ldots$
2. IO bottleneck induces random Reducer retries:
   - Up to 10% variation in runtimes
   - Stretches measured runtimes
   - Distorts normalization of the speedup data

Brief Explanation

1. Apparent capacity boost produced by IO bottleneck:
   - Boost $\equiv$ Gradual reduction in IO constraint
   - Relaxation of the latent IO bandwidth constraint.
   - Constraint decreases with cluster size $p = 1, 2, 3, \ldots$
2. IO bottleneck induces random Reducer retries:
   - Up to 10% variation in runtimes
   - Stretches measured runtimes
   - Distorts normalization of the speedup data

Details to appear in forthcoming ACM Queue article

Brief Explanation

1. Apparent capacity boost produced by IO bottleneck:
   - Boost $\equiv$ Gradual reduction in IO constraint
   - Relaxation of the latent IO bandwidth constraint.
   - Constraint decreases with cluster size $p = 1, 2, 3, \ldots$
2. IO bottleneck induces random Reducer retries:
   - Up to 10% variation in runtimes
   - Stretches measured runtimes
   - Distorts normalization of the speedup data

Details to appear in forthcoming ACM Queue article

[Hadoop experts plz tweet me @DrQz (we have some questions)]

Outline

1. How to Quantify Scalability
   - Components of Scalability
   - Universal Scalability Law
2. Application of USL
   - Varnish
   - Memcached
   - Sirius
3. Superlinear Sizing
   - What it looks like
   - Perpetual motion
   - Hunting the Superlinear Snark
4. Summary

Summary

- USL requires $\boxed{\sigma, \kappa > 0}$ for S_p to be concave function
- Convex efficiencies $S_p/p > 100\%$ do (appear to) exist
- Data $\rightarrow$ $\boxed{\sigma < 0}$ in USL model $\equiv$ **superlinear detector**
- Super efficiency is not free
  - Like perpetual motion, it's an **illusion**
  - You will pay the piper eventually
  - Debugging latent capacity boost can be very tricky

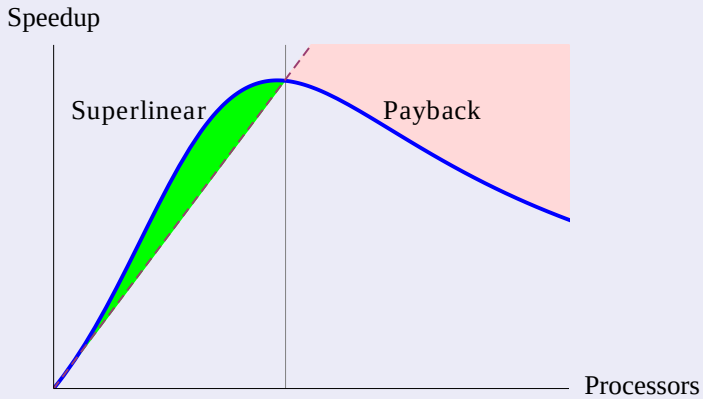
Summary

- USL requires $\sigma, \kappa > 0$ for S_p to be concave function
- Convex efficiencies $S_p/p > 100\%$ do (appear to) exist
- Data $\rightarrow$ $\sigma < 0$ in USL model $\equiv$ **superlinear detector**
- Super efficiency is not free
  - Like perpetual motion, it's an **illusion**
  - You will pay the piper eventually
  - Debugging latent capacity boost can be very tricky

Theorem (Gunther 2012)

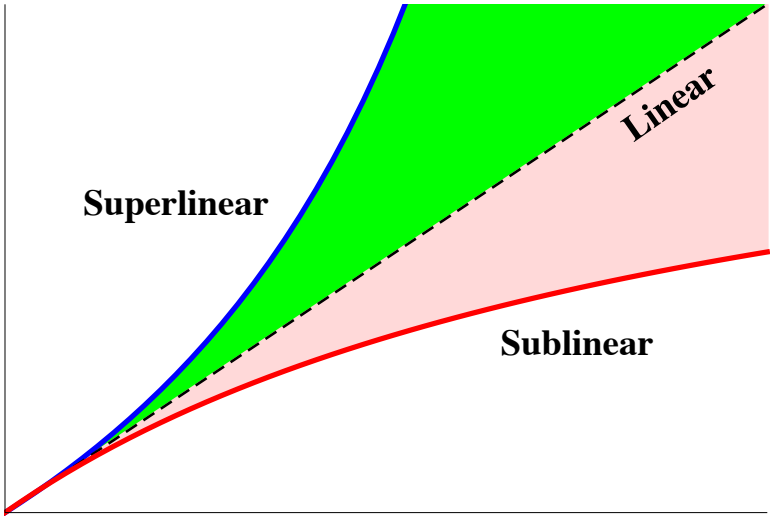
USL Payback Trap:
Superlinearity is always followed by severe speedup degradation in the payback region

Validated on AWS EC2 Hadoop-Terasort by Kris Tomasette (2014)

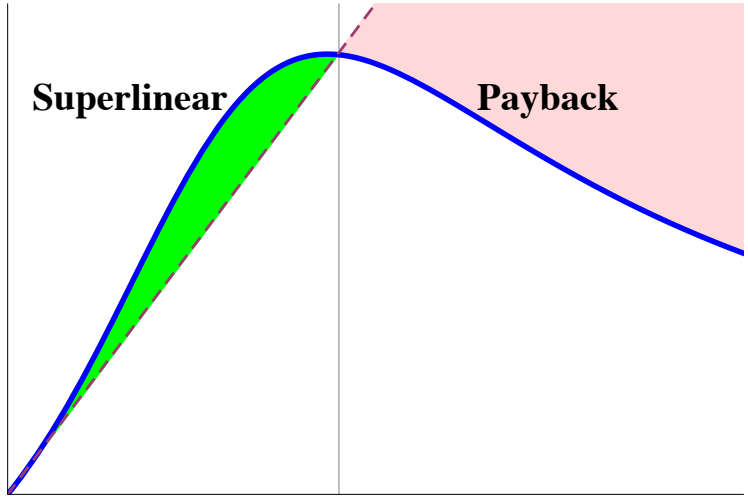


The Takeaway Visual

Not even this ...

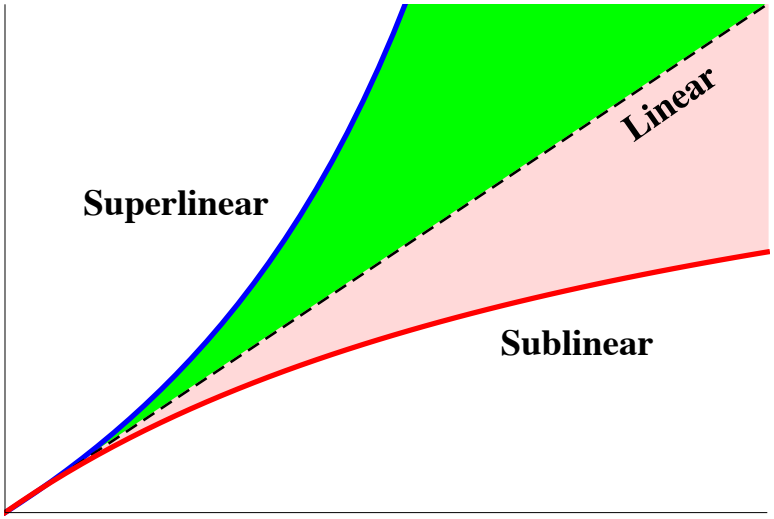


Its really this...

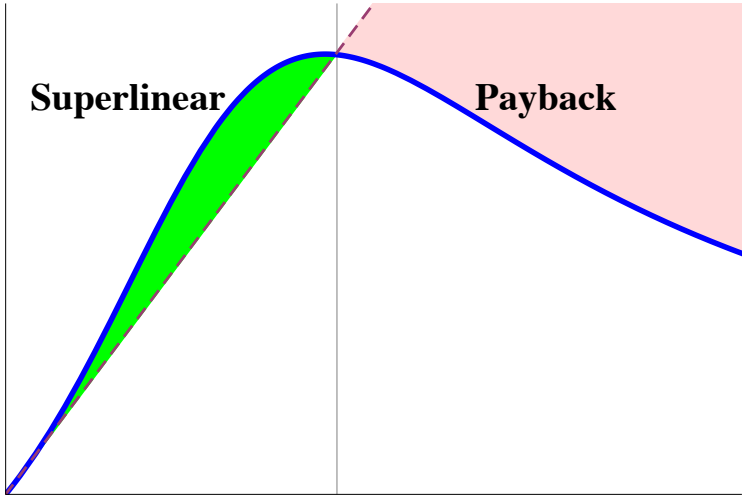


The Takeaway Visual

Not even this ...

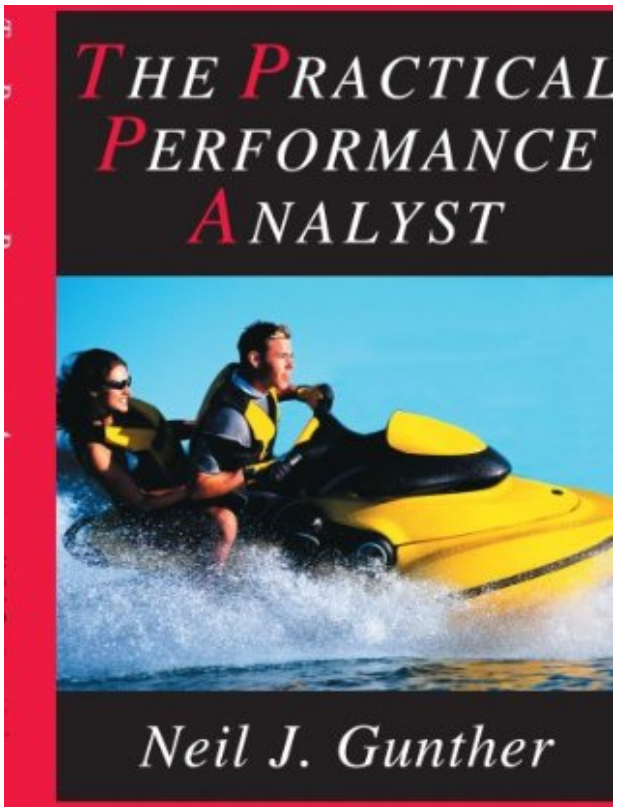


Its really this...

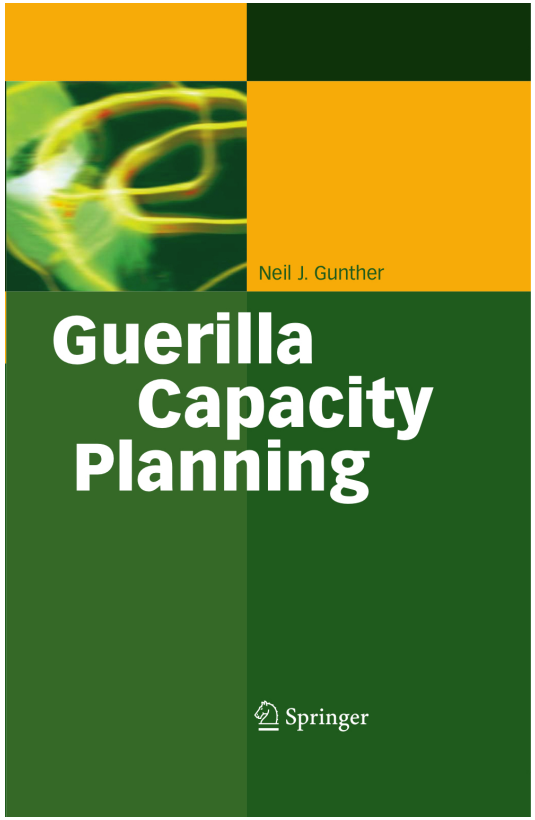


You have been warned ☺

More Details About USL Modeling



Chapters 6 and 14



Chapters 4–6

Also check out:

- USL web page
- Guerrilla classes



Thank you!

Castro Valley, California
www.perfdynamics.com
perfdynamics.blogspot.com
Twitter/DrQz
Facebook
njgunther@perfdynamics.com
+1-510-537-5758