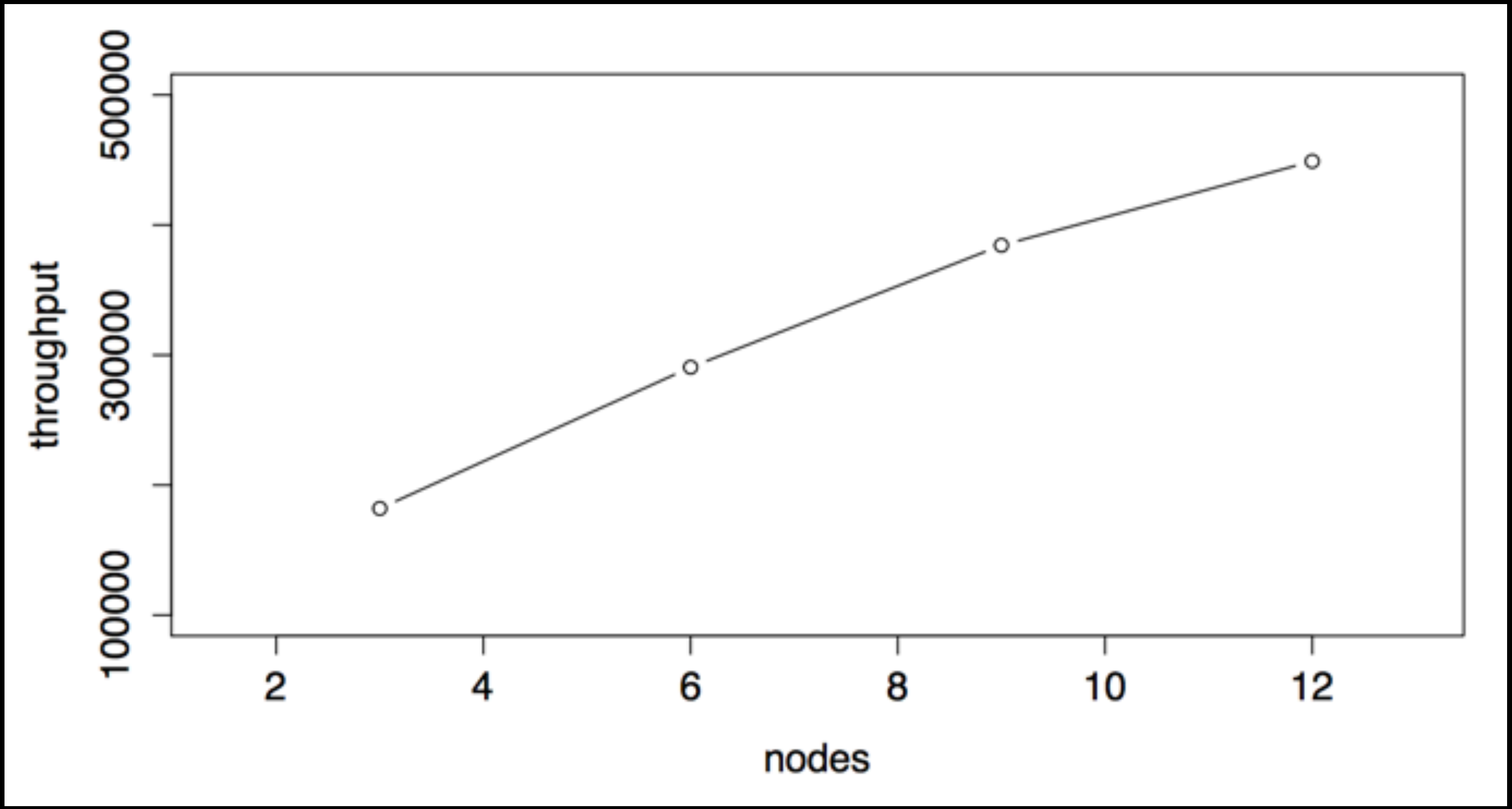
Forecast MySQL Scalability with USL

洪 斌

Linear Scalability?



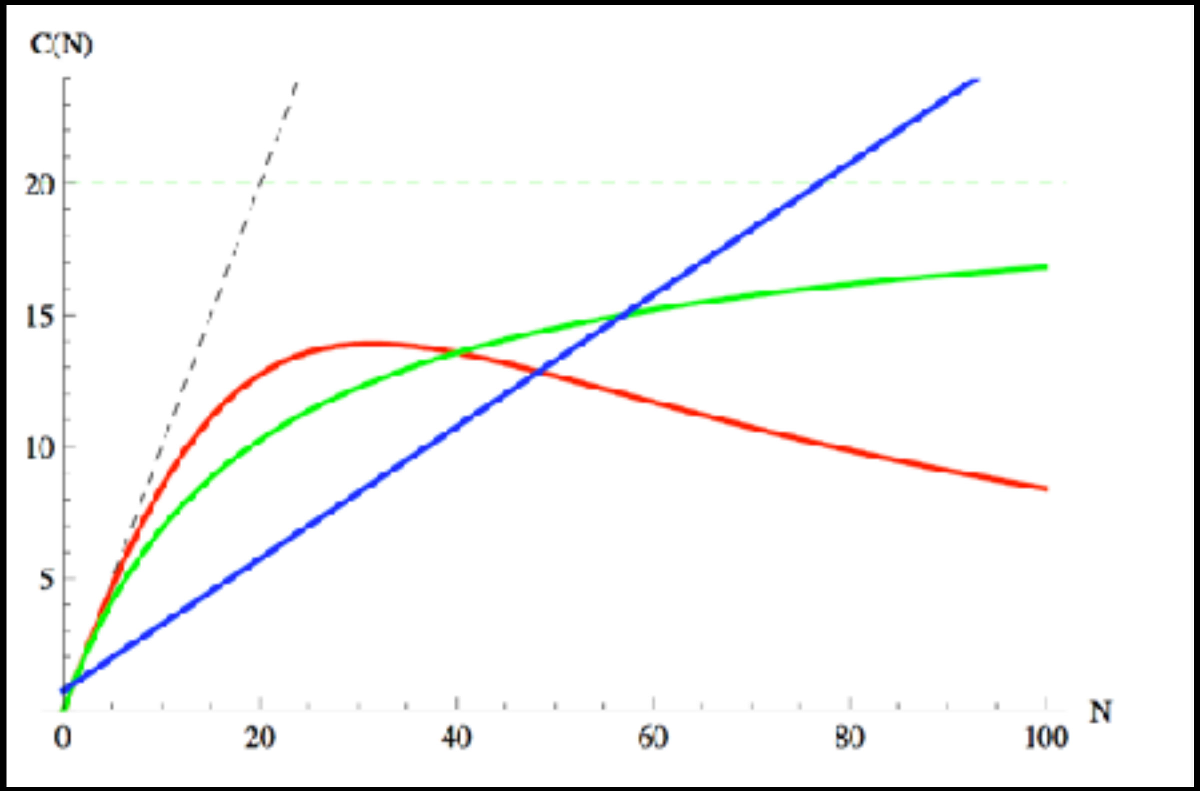
What is Scalability?

the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth

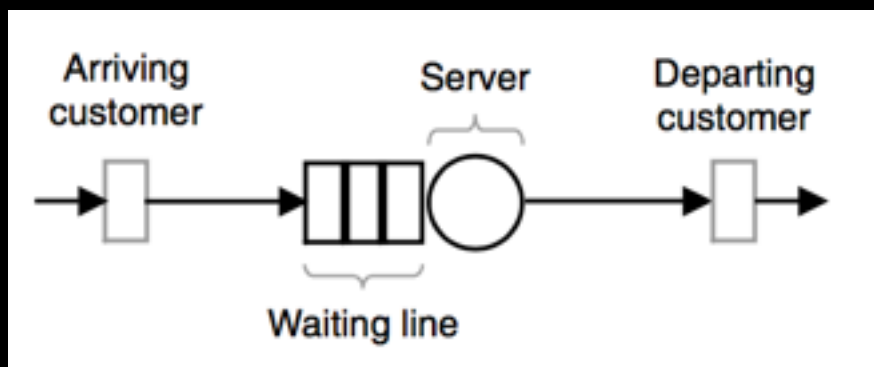
Scalability is function.

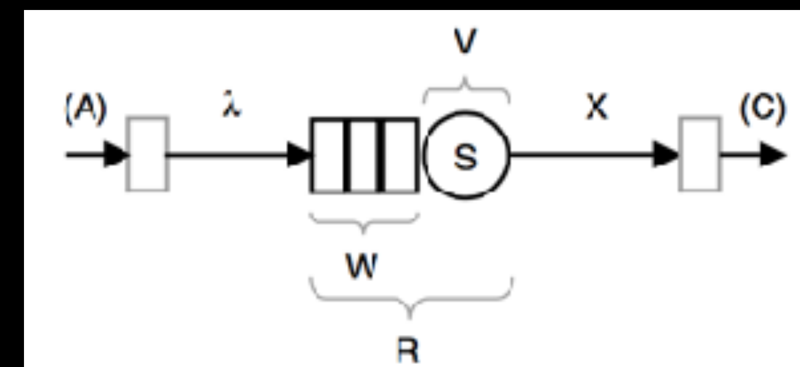
Scalability Law

- Little’s Law (1961)
- Amdahl’s Law (1967)
- Gustafson’s Law (1988)
- Universal Scalability Law (1993)

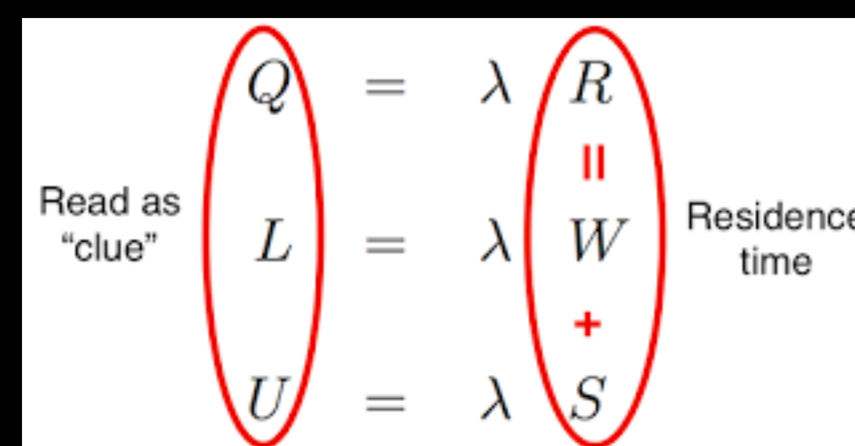


Queueing theory





- 服务请求量=到达率 * 驻留时间(响应时间)
- 队列长度=到达率 * 等待时间
- 利用率=到达率 * 服务时间

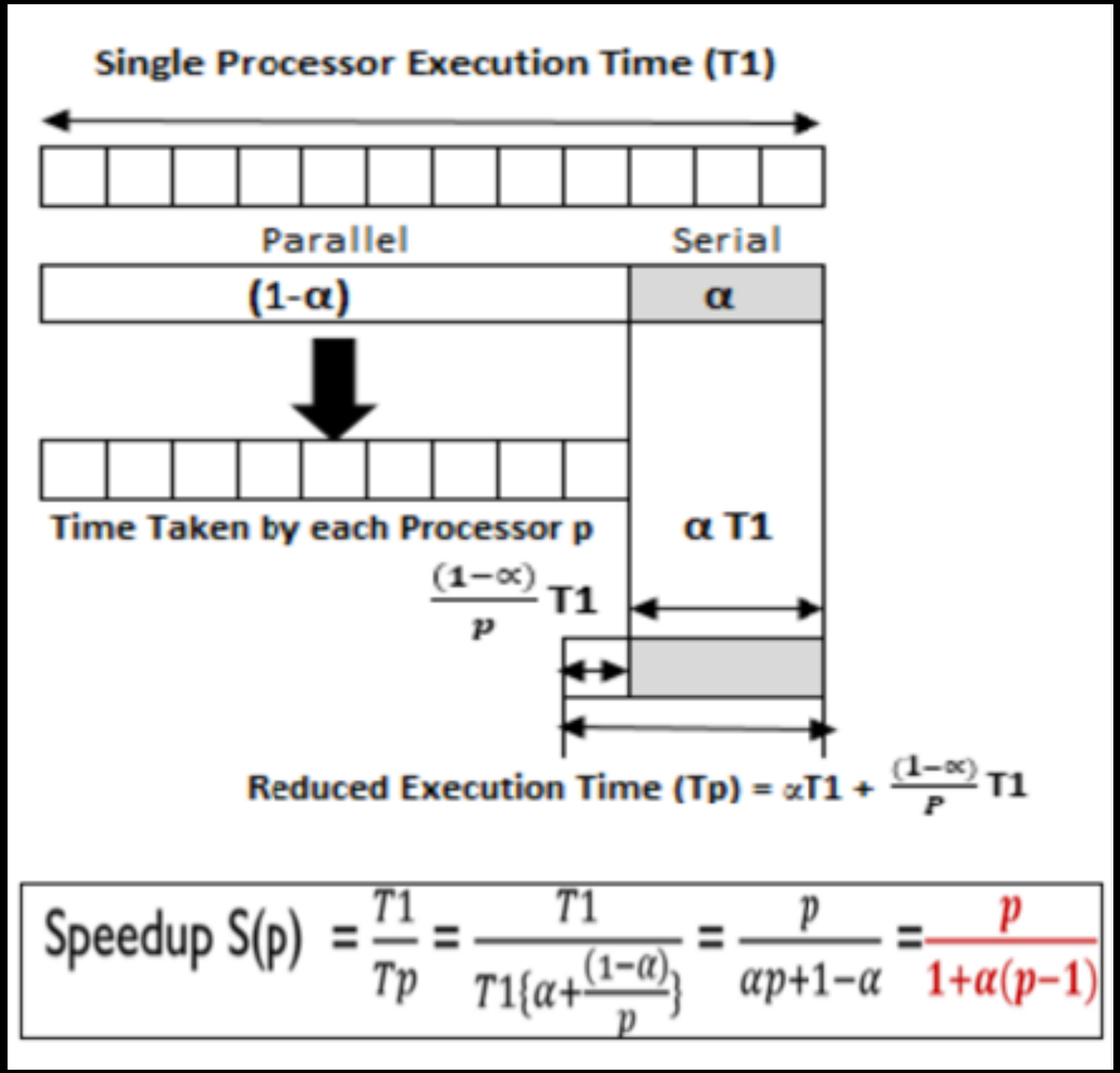


Amdahl’s Law

If an amount of work N is completed in time T 1 on a uniprocessor, the same amount of work can be completed in time T p < T 1 on a p-way multiprocessor. The speedup S p = T 1 /T p is one measure of scalability.

$$C_A(N,\alpha) = \frac{N}{1+\alpha(N-1)}$$

- N = Processor
- α = Contention (串行化比率)



Gustafson's Law

Amdahl's law assumes the size of the work is fixed. Gustafson's modification is based on the idea of scaling up the size of the work to match p.

$$S_n^{'} = \alpha + (1-\alpha)n$$

Amdahl's Law

$$S_n = \frac{W/1}{\frac{\alpha W}{1} + \frac{(1-\alpha)W}{n}} = \frac{n}{1+(n-1)\alpha}$$

负载扩展至n个节点

$$W' = \alpha W + (1-\alpha)nW$$

$$S_n^{'} = \frac{(\alpha W + (1-\alpha)nW)/1}{\frac{\alpha W}{1} + \frac{(1-\alpha)nW}{n}}$$

USL

The USL is equivalent to the synchronous queueing bound on throughput for a linear load-dependent machine repairman model of a multiprocessor.

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

- N = Concurrency (or Processor)
- α = Contention (waiting for shared resources)
- β = Coherency (waiting data synchronous)

A General Theory of Computational Scalability Based on Rational Functions

USL

$$
\begin{aligned}
C(N) &= \frac{N}{1+\sigma(N-1)+\kappa N(N-1)} \\
\frac{C(N)}{N} &= \frac{1}{1+\sigma(N-1)+\kappa N(N-1)} \\
\frac{N}{C(N)} &= 1+\sigma(N-1)+\kappa N(N-1) \\
\frac{N}{C(N)}-1 &= \sigma(N-1)+\kappa N(N-1)
\end{aligned}
$$

(1)

$$
\begin{aligned}
x &= N-1 \\
y &= \frac{N}{C(N)}-1
\end{aligned}
$$

(2)

$$
\begin{aligned}
y &= \sigma(N-1)+\kappa N(N-1) \\
&= \kappa N(N-1)+\sigma(N-1) \\
&= \kappa(N-1+1)(N-1)+\sigma(N-1) \\
&= \kappa(N-1)(N-1+1)+\sigma(N-1) \\
&= \kappa x(x+1)+\sigma x \\
&= \kappa x^2+\kappa x+\sigma x \\
&= \kappa x^2+(\kappa+\sigma)x
\end{aligned}
$$

(3)

$$
\begin{aligned}
a &= \kappa \\
b &= \sigma+\kappa
\end{aligned}
$$

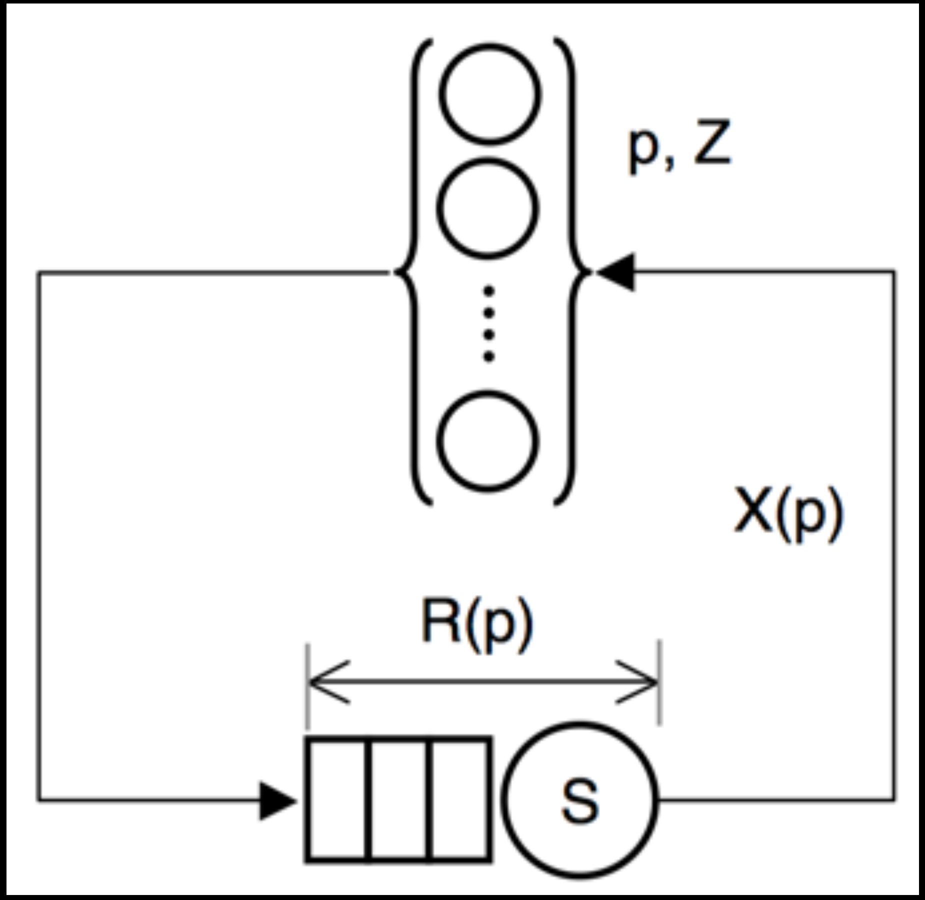
(4)

$$y = ax^2 + bx + 0$$

(5)

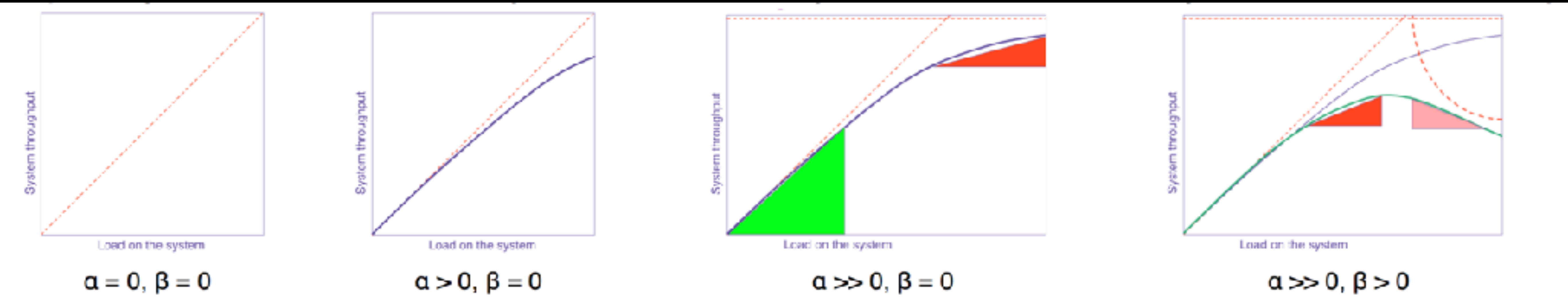
Standard MRM

在有限的p个机器的生产线，每工作Z段时间就有机器故障，需要花费S段时间修复，如果多个机器故障按FIFO顺序修复。



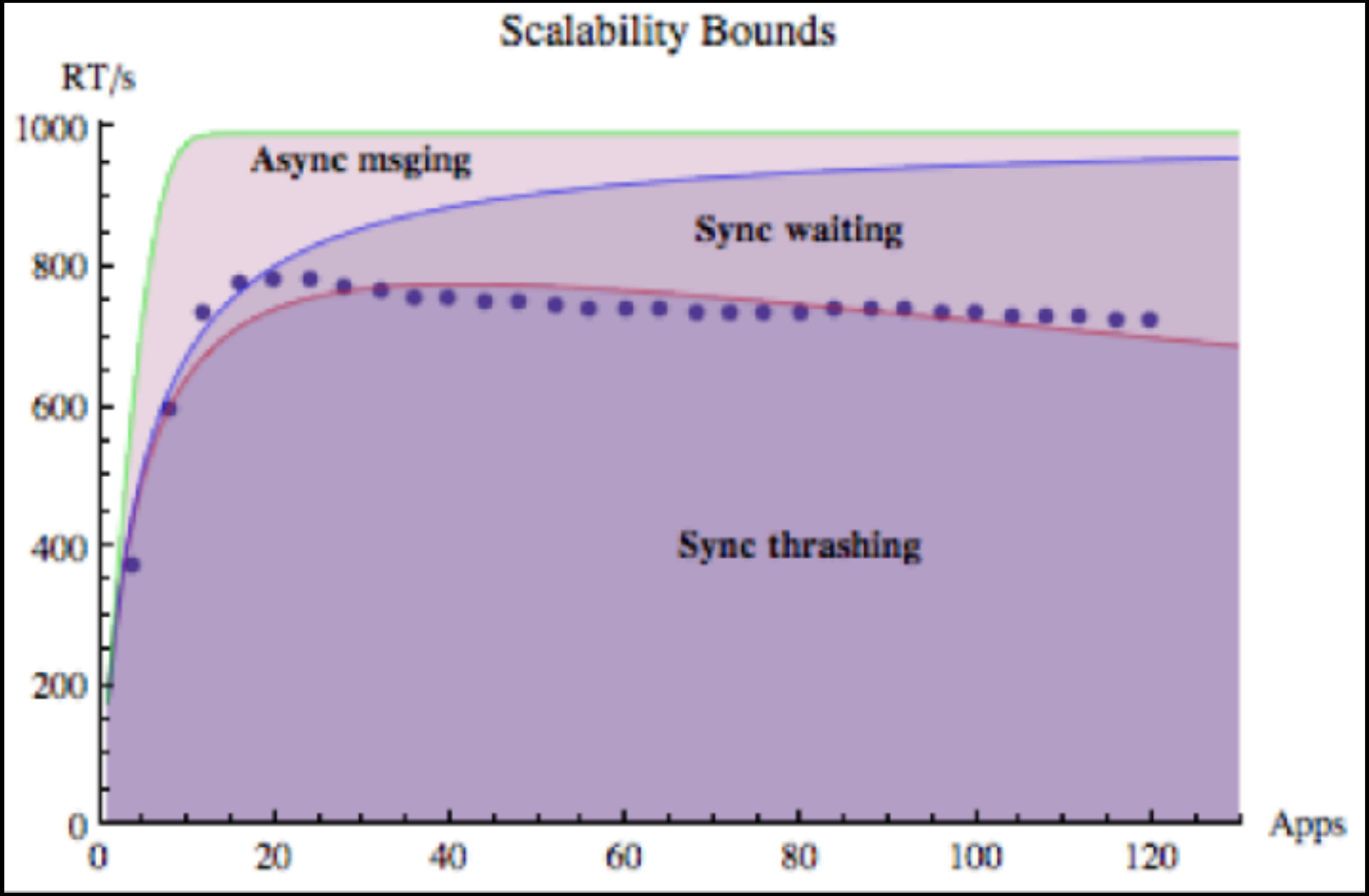
Metric	Repairman	Multiprocessor	Time share
p	machines	processors	users
Z	up time	execution period	think time
S	service time	transmission time	CPU time
R(p)	residence time	interconnect latency	run-queue time
X(p)	failure rate	bandwidth	throughput

Scalability Model



A: Ideal concurrency ($\sigma, \kappa = 0$)	B: Contention-limited ($\sigma > 0, \kappa = 0$)
Single-threaded tasks	Tasks requiring locking or sequencing
Parallel text search	Message-passing protocols
Read-only queries	Polling protocols (e.g., hypervisors)
C: Coherency-limited ($\sigma = 0, \kappa > 0$)	**D: Worst case** ($\sigma, \kappa > 0$)
SMP cache pinging	Tasks acting on shared-writable data
Incoherent application state between cluster nodes	Online reservation systems
	Updating database records

Scalability Zones



Contention & Coherency

	Contention (α)	Coherency (β)
含义	共享数据的争用	一致性的开销
举例	不同请求更新相同数据行	内存与磁盘间或不同CPU的缓存间的一致性
根源	无法并行的任务	进程间同步的开销
自变量	N-1: 假设需要处理N个进程,最坏场景下有N-1个进程在等待	N*(N-1): 假设需要处理N个进程, 每个进程间要与N-1个进程同步，即N*(N-1)

Predict

- Predict maximum scalability

$$N_{max} = \sqrt{(1 - \alpha) / \beta}$$

- Predict throughput Xmax at load Nmax

$$X_{max} = X(1) * C(N_{max})$$

DB Capacity Planning

- 基准测试估计容量（时间和成本）
- 没有完整数据库的负载组成信息
- 无法准确度量事务的执行时间

Step to Apply USL

1. 选择度量参数
   - Load: QPS/TPS
   - Concurrency: Thread_running(MySQL)
2. 搜集数据
   - mysqladmin -i1 ext |awk 'BEGIN{printf "%5s %5s\n", "conn", "tput" } /Threads_running/{run=$4} /Queries/{q=$4-qp;qp=$4;printf "%5d %5d\n", q, run}'
3. 整理数据
4. 拟合数据
5. 分析结果

Example

```
sample <- read.csv(“8003.tput",sep="")

usl <- nls(tput ~ conn/(1+sigma * (conn-1)+
conn*(conn-1)),sample,start=c(sigma=0.1,kappa=0.01))

sigma <- coef(usl)['sigma']
kappa <- coef(usl)['kappa']

u=function(x){y=x/(1+sigma * (x-1)+ kappa*x*(x -1))}

plot(u,0,max(benchmark$conn)*2,xlab=“Concurrency",col="green", ylab="Throughput",
lty=“dashed",add=TRUE)

points(benchmark$conn,benchmark$tput)
```

https://kevinbin.shinyapps.io/uslapp/

Conclusions

- Scalability 是可以被量化的
- 线性扩展意味着资源翻倍，负载也翻倍
- 资源垂直扩展不意味处理性能增加，关键是串行化比例。
- 即便极小Coherency也会使Scalability倒退
- 具备良好Scalability的系统应尽可能避免Contention和Coherency

Reference

- How to Quantify Scalability (Neil J. Gunther)
- Getting in the Zone for Successful Scalability
- USL for R package
- A Little Triplet
- Guerrilla Capacity Planning
- Analyzing Computer Systems Performance with Perl PDQ

“all models are wrong, but some are useful.”

–George E. P. Box