

# Software Performance Analytics: Past, Present and Future

Kingsum Chow  
Alibaba Infrastructure Services

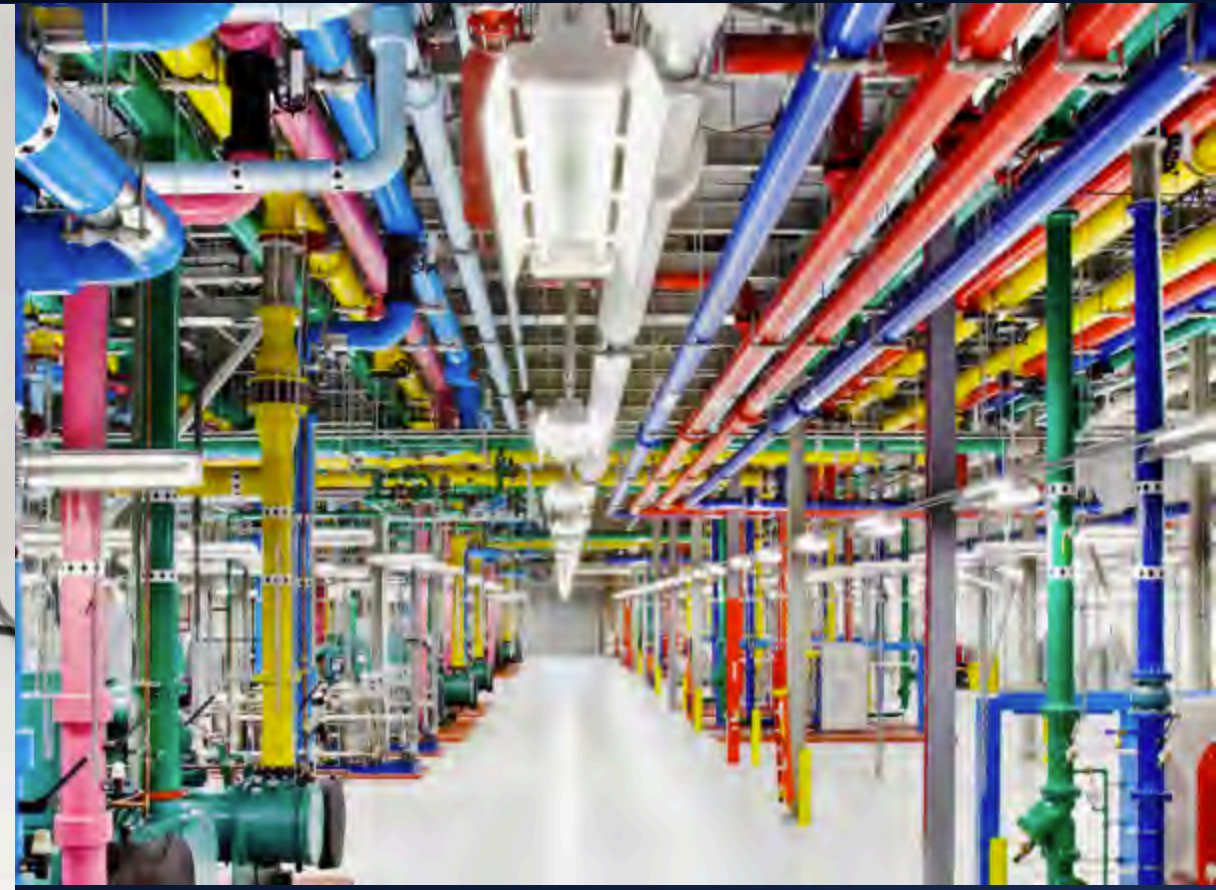
Keynote presented at Qcon Beijing on 2017.04.18

# Great programs, but...

- Software doesn't scale
- Hardware is too slow
- Tuning software doesn't work
- Tuning software in the data center is difficult



# Performance Scaling Personal Computer → Data Center



# Bridging Software, Hardware and Analytics

- Software Performance Scaling: Amdahl's Law, Gustafson's Law
- CPI model
- Tuning
- Data Center Optimizations

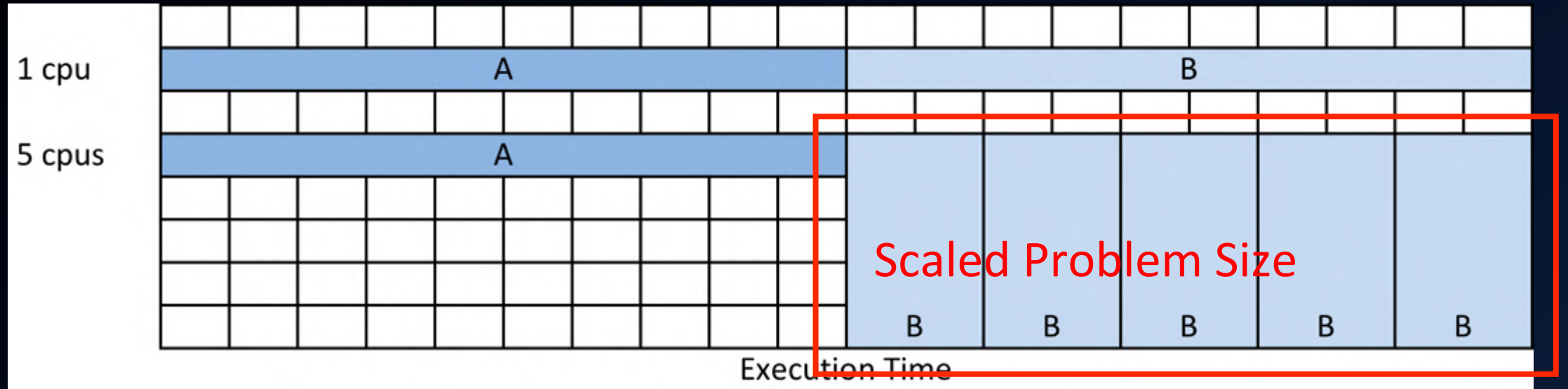


# Amdahl's law (1967)



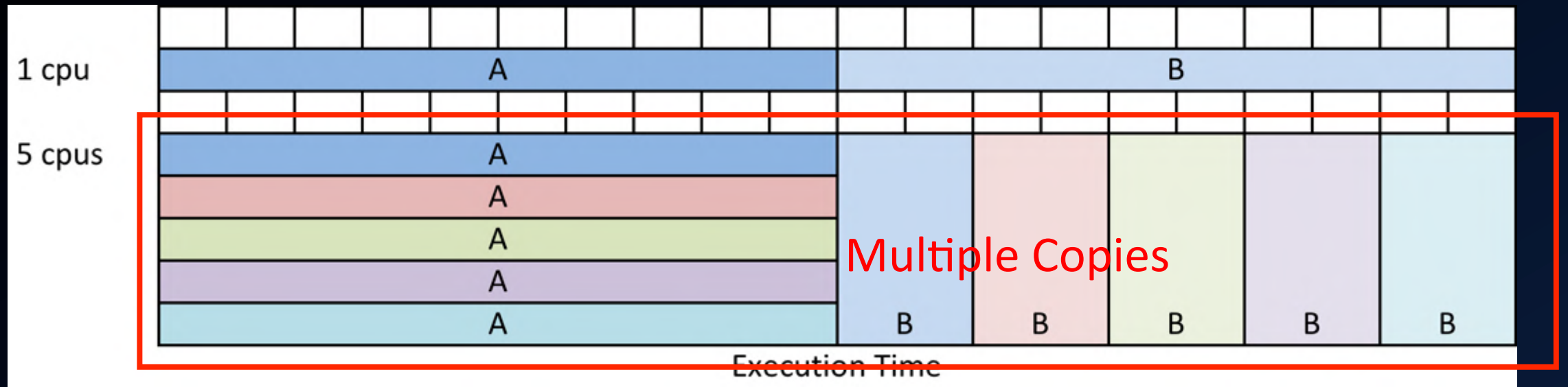
Speed up from 1 cpu to 5 cpu's  
= 1 cpu execution time / 5 cpu's execution time  
= 20 / 12 = 1.67

# Gustafson's Law (1988)



Speed up from 1 cpu to 5 cpu's @ 20 time units (perfect B scaling)  
= work units @ 5 cpu's / work units @ 1 cpu  
= 5 / 1 = 5

## Even if B doesn't scale...



Speed up from 1 cpu to 5 cpu's @ 20 time units (B doesn't scale)

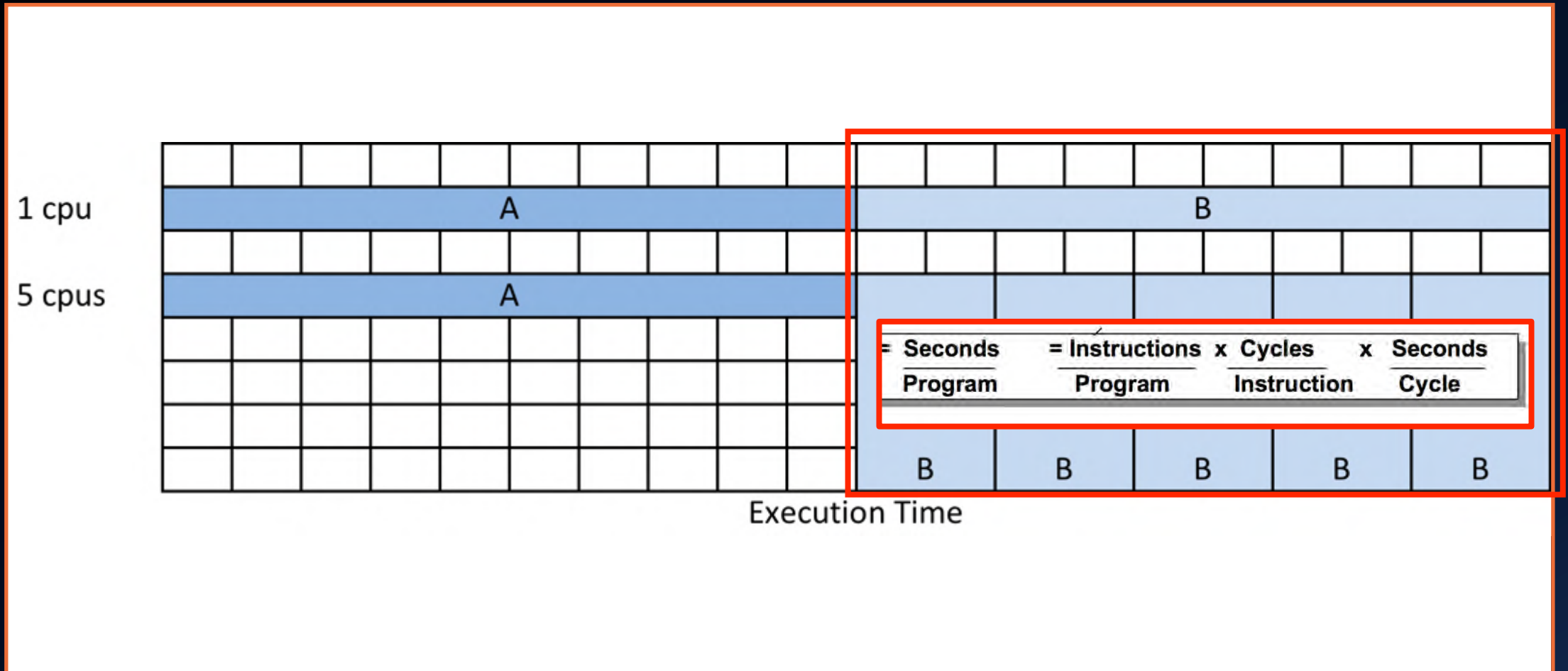
= work units @ 5 cpu's / work units @ 1 cpu

= 5 / 1 = 5

Average Latency = 16 time units @ 5 cpu's

= 20 time units @ 1 cpu

# What if the software performance doesn't scale?





3. How does the length of instructions increase with the problem size

1. Inverse of CPU Clock Frequency in Hz

$$= \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

2. How efficient are the instructions executed on the hardware

# 1. Increase CLK (and cost)

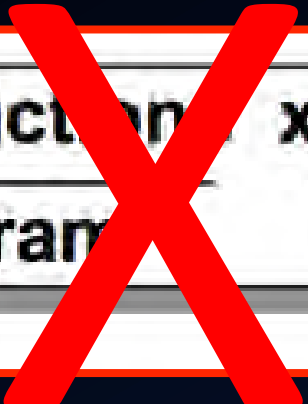
- Hardware
  - Increase CPU frequency (as power consumption may increase super-linearly, new CPU's may or may not run at higher frequencies)
  - Increase cores (new CPU's probably come with more cores)
  - Increase sockets (expensive)
  - Hyper-Threading (creates the illusion of more clocks, may increase performance, note counting clocks is no longer straight-forward, essentially free)
  - ...

## 2. Reduce CPI

- Hardware + Software
  - Advanced CPU designs (better cache placements, bigger caches, better prefetches, more efficient ITLB's and DTLB's, advanced branch prediction engines, faster memory accesses, **new CPUs probably lower CPI, but note frequencies in the previous page**)
  - Better inlining (new compilers and just in time compilation in runtime)
  - Efficient Placement of objects in cache lines
  - ...

### 3. Reduce path length

- SW (compiler, runtime config, programming)
  - Better compilers
  - Profile guided optimizations
  - Runtime optimizations (recompilation and inlining of hot methods)
  - Reduce garbage collections
  - Rewrite source code (hopefully more efficient)
  - ...


$$= \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instruction}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

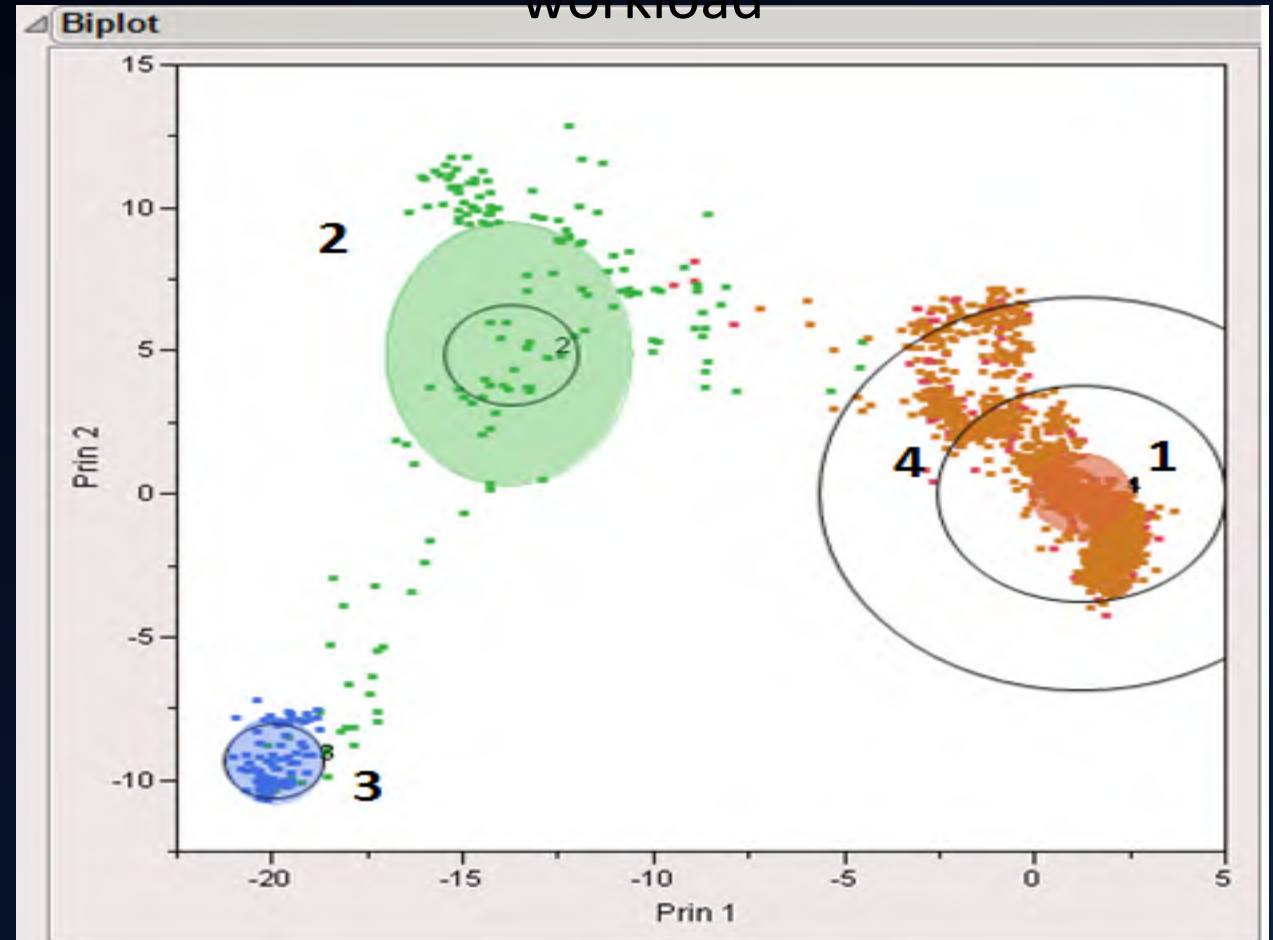
$$\text{Execution Time} = \sum_j ( I_j \times CPI_j ) \times \text{cycle\_time}$$



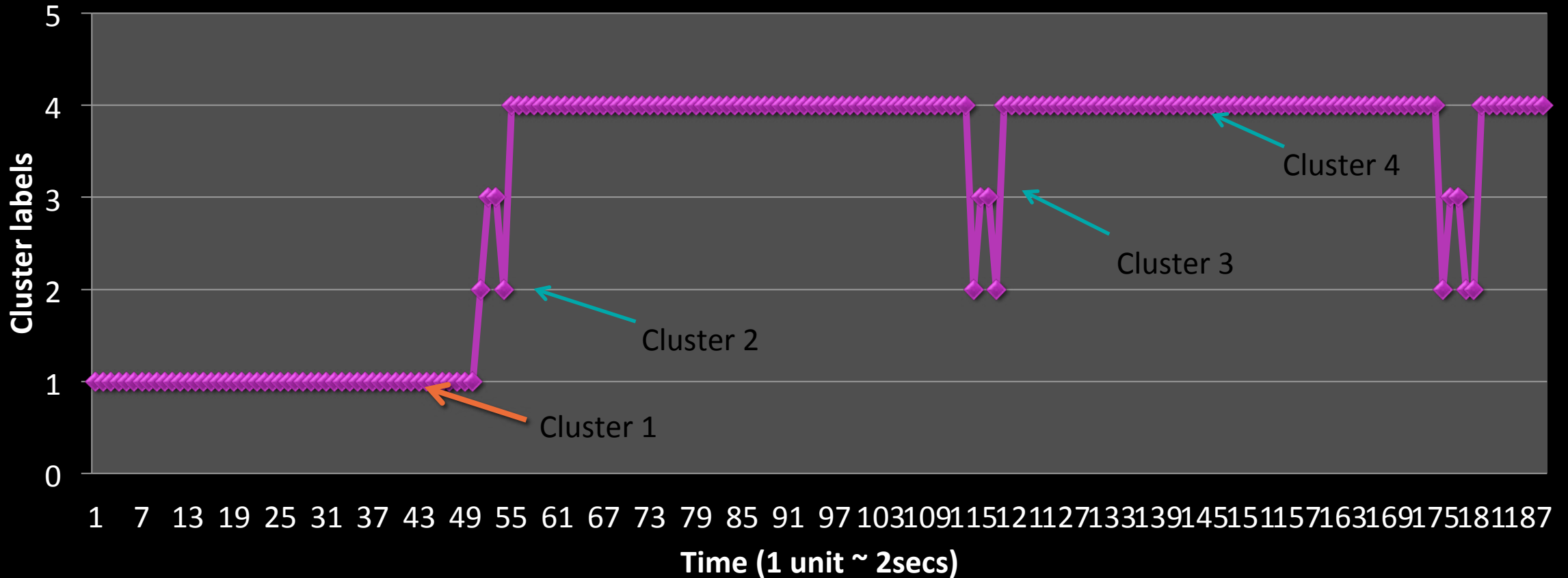
# Phase detection for server workload

Phases in performance data from server workload

- CPI models for workload phases
- Clustering analysis applied
- Clusters 2 & 3 mapped to Full Garbage Collection (GC) phase.
- Clusters 1 & 4 mapped to application phases.
- Displayed on the dimensions of the first two principal components



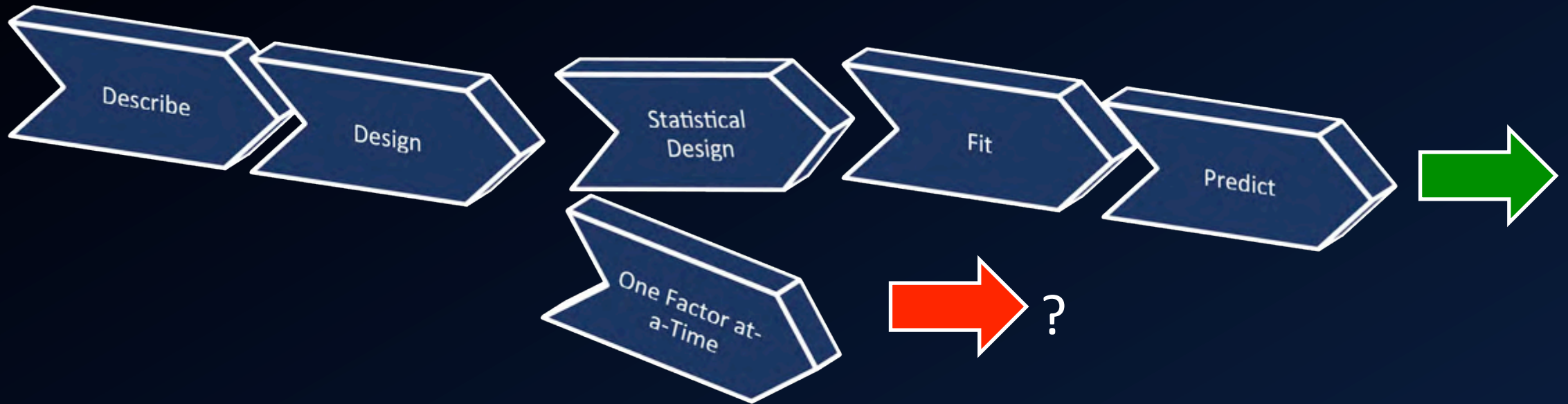
## Phases mapped to timestamp in server workload



Phases 2 and 3 showed different characteristics of a Full GC phase

Phase 1 and 4 showed application behavior

# An alternative: Can we just try different configurations?



# Tuning one factor at a time



X1 = 70  
X2 = 100  
X3 = OFF  
X4 = ON  
X5 = OFF

X1 = 70  
X2 = 100  
X3 = OFF  
X4 = ON  
X5 = OFF

X1 = 70  
X2 = 100  
X3 = OFF  
X4 = ON  
X5 = ON

3.372 seconds

3.3813 seconds

X1 = 70  
X2 = 100  
X3 = OFF  
X4 = OFF  
X5 = OFF

2.83 seconds

X1 = 70  
X2 = 100  
X3 = OFF  
X4 = ON  
X5 = OFF

3.372 seconds

# Tuning one factor at a time

Speed Up  
= 3.38 / 2.69  
= 1.26x





# Design of Experiments

Tunable Parameter	Type	Function	Level
X1	Continuous	Lock	30/70/120
X2	Continuous	Optimization	50/100/200
X3	2-Level Categorical	Optimization	ON/OFF
X4	2-Level Categorical	Optimization	ON/OFF
X5	2-Level Categorical	Lock	ON/OFF



One Factor  
at-at-time



DOE SW opt



After HW-SW opt



Further opt

Speedup 5.9x  
→ 6.4x → 8x

# An alternative: Response Surface Method for 2 numeric parameters

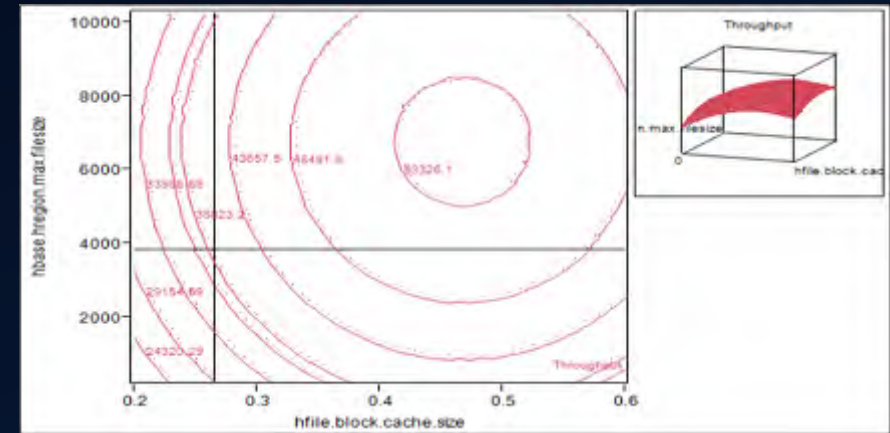
- Throughput measured with varying **hbase.hregion.max.filesize(in GB)** and **hfile.block.cache.size (as a percentage)**

hfile.block.cache.size (as a percentage)	hbase.hregion.max.filesize(in GB)		
	Throughput in ops/sec	.25	3.5
.2	19486	38115	21550
.4	44796	36096	58160
.6	31887	51324	44382

Maximum throughput = 58160 ops/sec

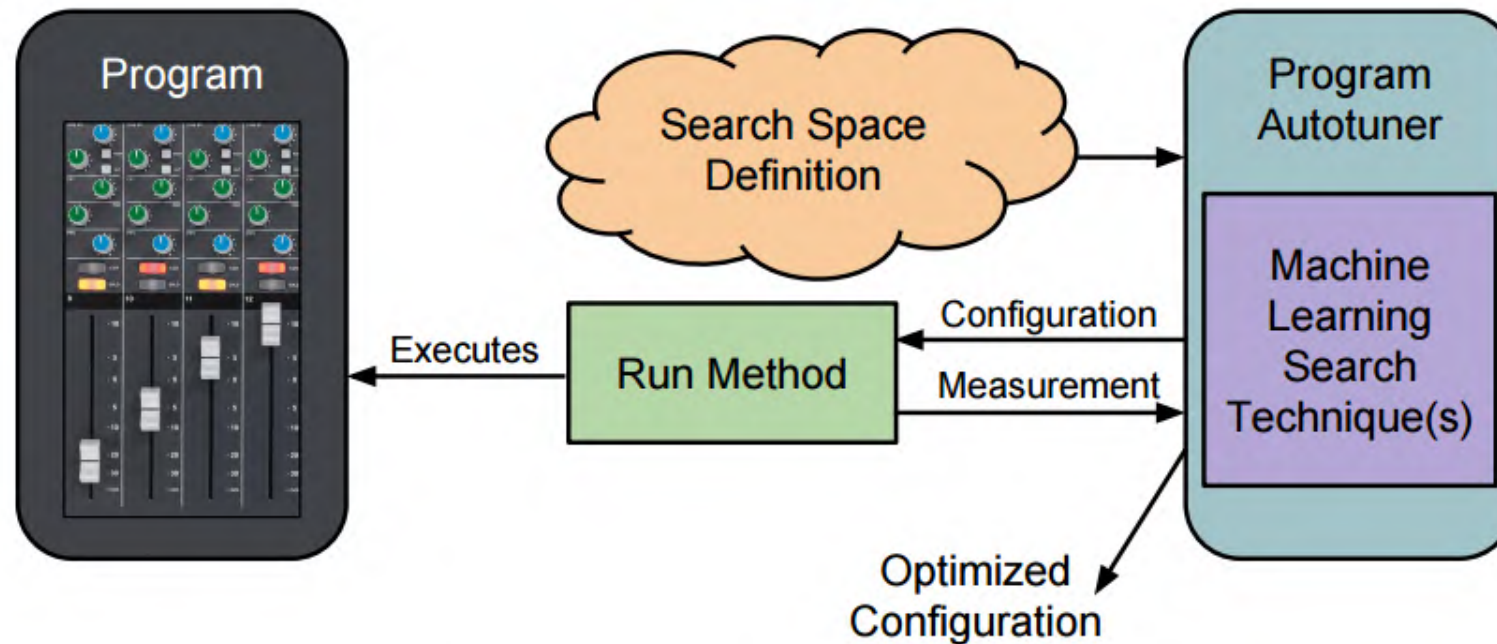
# Results from a YCSB run based on optimal configuration derived from RSM

- Configuration parameters
  - `hfile.block.cache.size` in percentage = 0.4678844
  - `hbase.hregion.max.filesize` in MB = 6.511 GB
- Measured throughput = 74662 ops/sec
- **Additional 28%** performance boost as compared to the best throughput from the screening experiments





# Auto-Tuning can be used for many parameters



# Automated JVM Tuning with Bayesian Optimization

## TUNING AT THE JVM LAYER

- Hotspot JVM has hundreds of tunable knobs:

```
$ java -XX:+PrintFlagsFinal -version | grep "="
uintx AdaptiveSizePolicyWeight           = 10           {product}
uintx AdaptiveSizeThroughPutPolicy       = 0            {product}
uintx AdaptiveTimeWeight                  = 25           {product}
bool AdjustConcurrency                     = false        {product}
bool AggressiveOpts                       = false        {product}
intx AliasLevel                           = 3            {C2 product}
bool AlignVector                          = false        {C2 product}
...

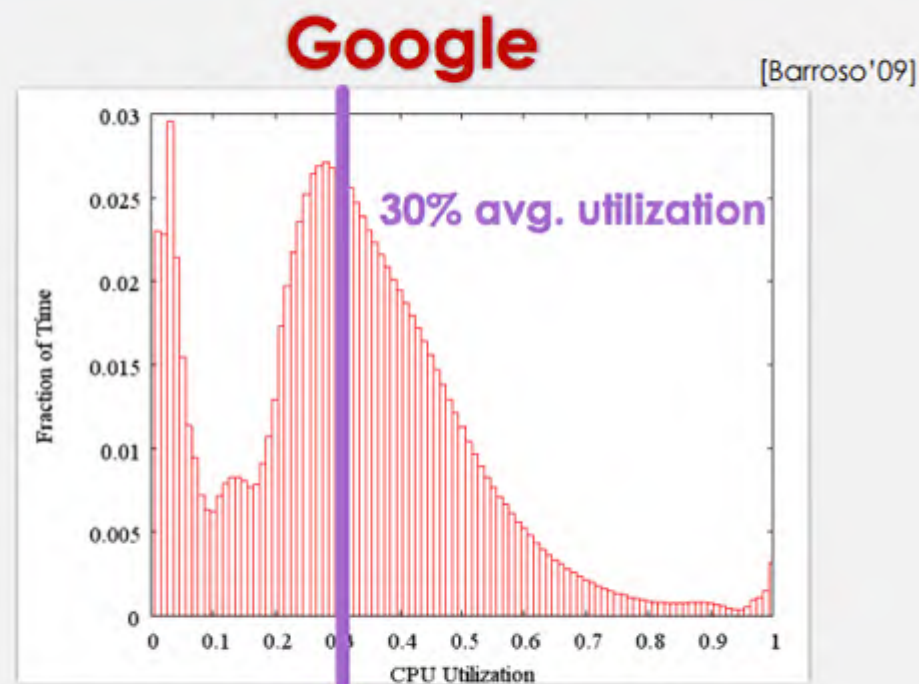
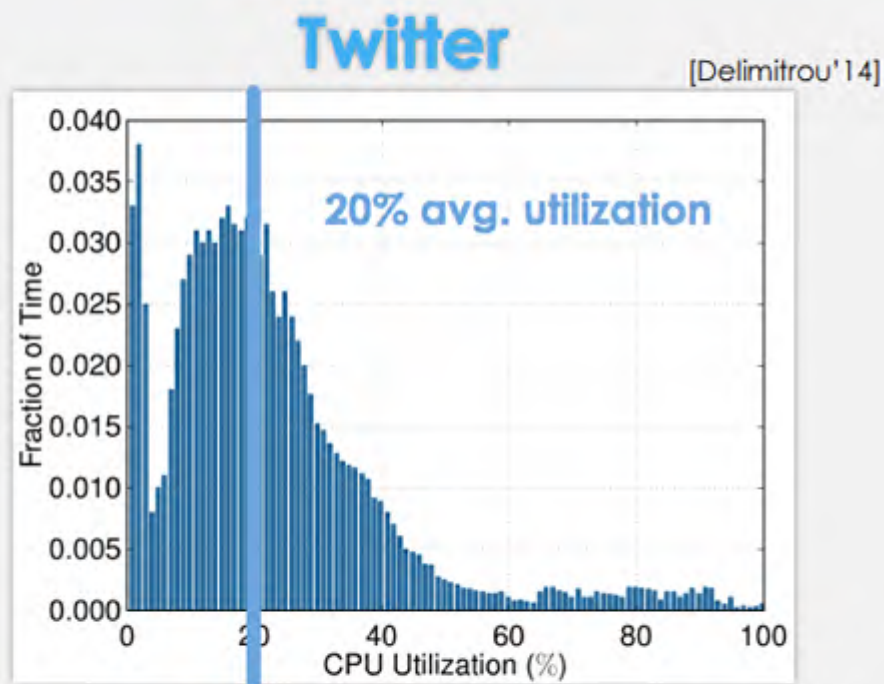
$ java -XX:+PrintFlagsFinal -version | grep "=" | wc -l
757
```

A large variety of parameters:

- performance-sensitivity
- hardware-dependency
- mutual (in)dependency



# But the datacenters are poorly utilized!



- Low utilization in large-scale clouds, even with automated management systems

# Heracles in a nutshell

- Oversubscription by co-locating a **Best Effort (BE)** job with a **Latency Critical (LC)** job, such as OLDI workloads
- Uses hardware and software knobs to mitigate interference
- Uses latency information from the application to adjust knobs
- Runs locally on each node, complements cluster manager

# Recommendations

- Evaluate performance scaling with problem size
- Apply CPI performance model for simple workloads
- Optimize with Design of Experiments, if number of experiments is small
- Auto tune with tools if there are a lot of parameters to tune and the response attributes can be readily obtained
- Increase data center utilizations while meeting service level agreements

# Past Me

Ugh, so much work to do... I'll do it later.



# Present Me

Why didn't Past Me already do this? Oh well, I'm sure Future Me will get it done.



# Future Me

Damn it! Why are you two such lazy bums? Stop leaving me all the work!



Knowledge is a treasure, but practice is the key to it. - Lao Tzu

FBGAGS.COM