Netflix
Performance Meetup

NETFLIX

Global Client Performance

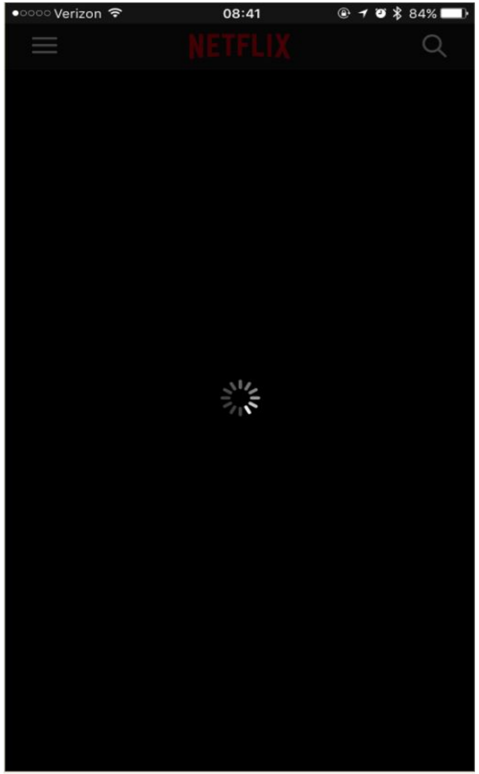
Fast Metrics

NETFLIX

3G in Kazakhstan

Making the Internet fast is slow.

- Global Internet:
  - faster (better networking)
  - slower (broader reach, congestion)
- Don't wait for it, measure it and deal
- Working app > Feature rich app



We need to know what the Internet looks like, without averages, seeing the full distribution.

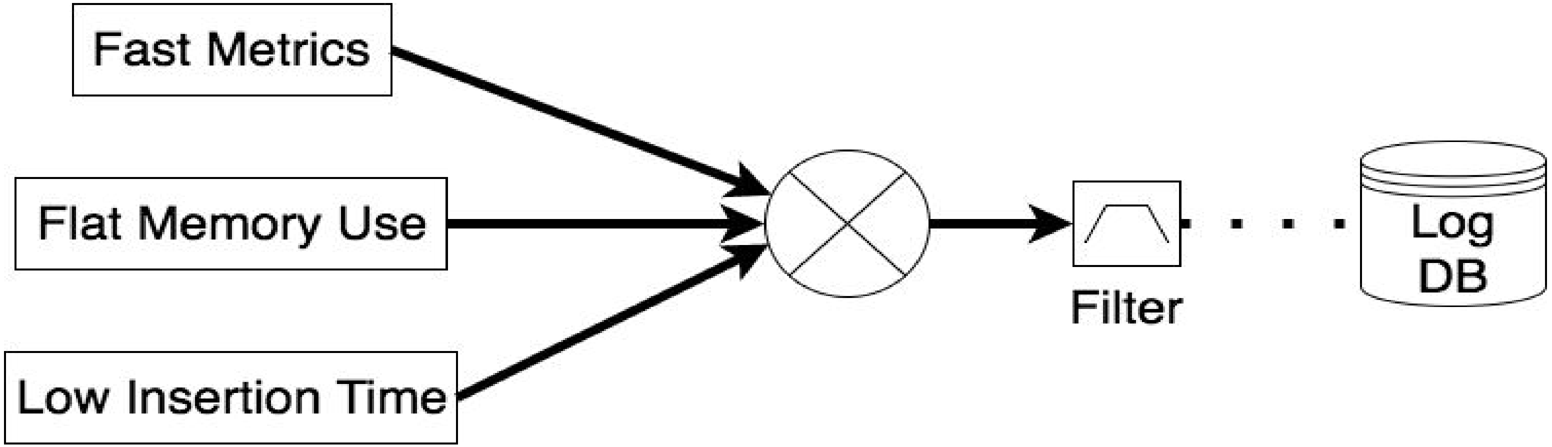
Logging Anti-Patterns

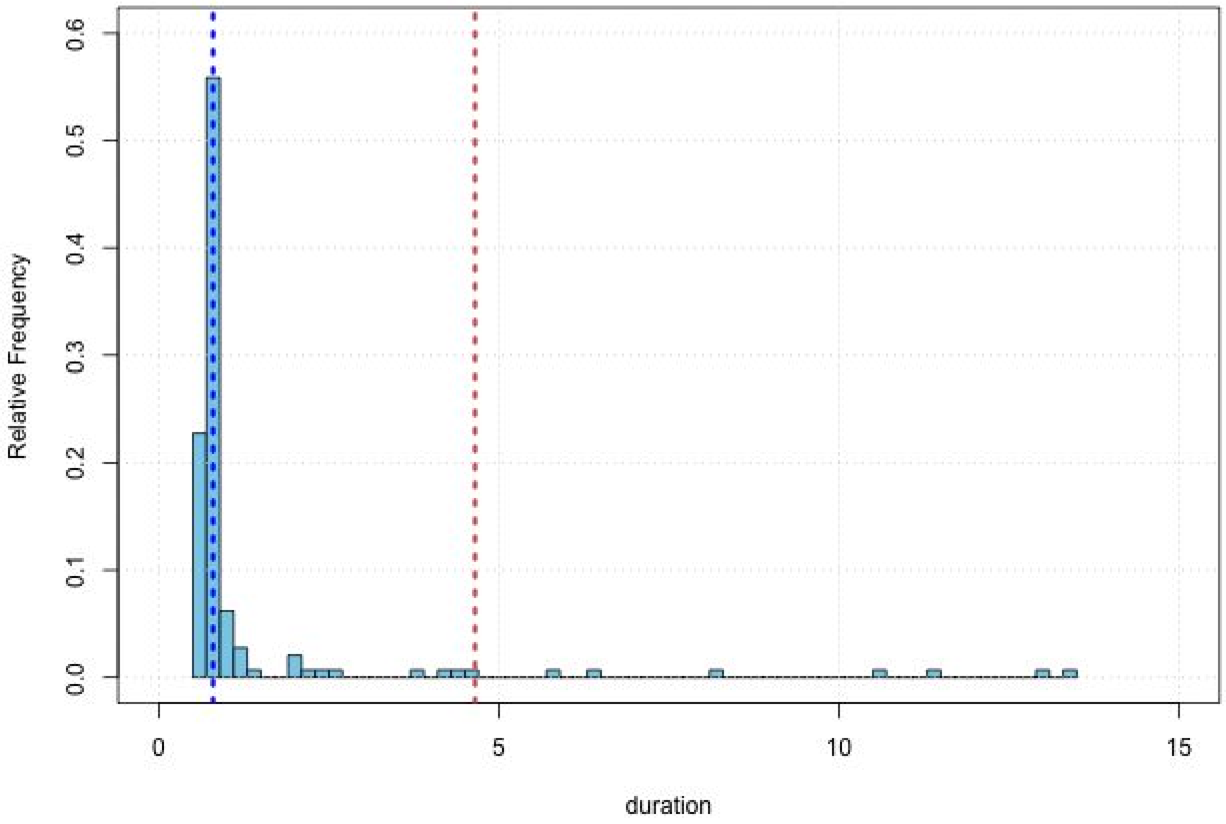
- Averages
  - Can't see the distribution
  - Outliers heavily distort
    - ∞, 0, negatives, errors
- Sampling
  - Missed data
  - Rare events
  - Problems aren’t equal in Population

Instead, use the client as a map-reducer and send up **aggregated** data, less often.

Sizing up the Internet.

Infinite (free) compute power!



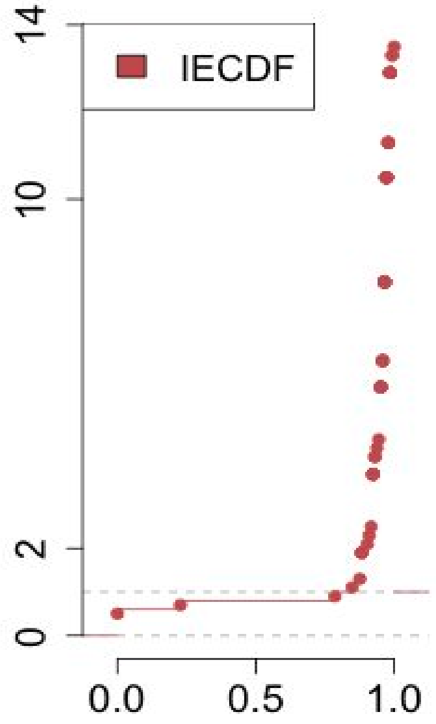

0.0
0.1
0.2
0.3
0.4
0.5
0.6
Relative Frequency
0
5
10
15
duration

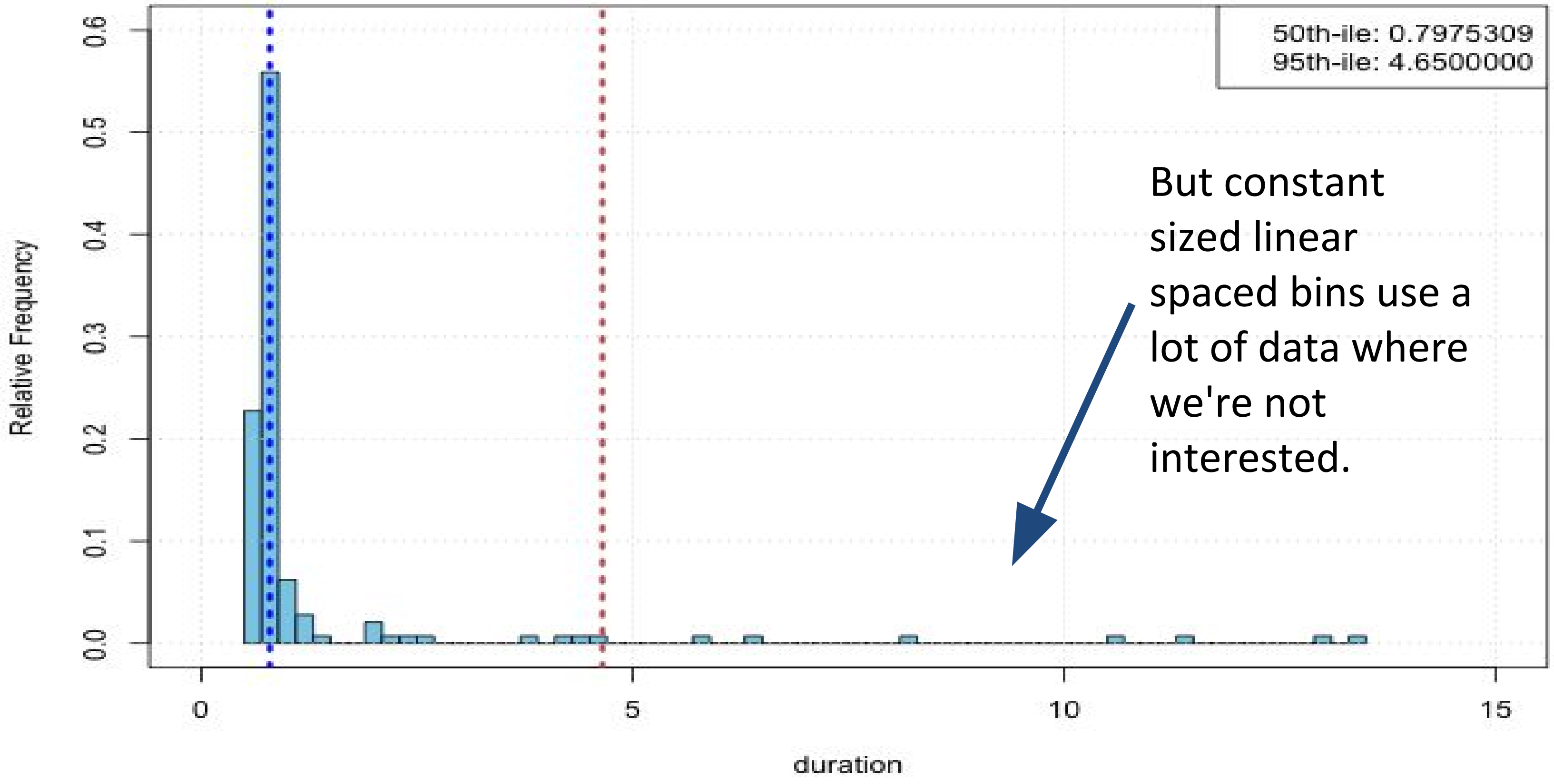
Get median, 95th, etc.

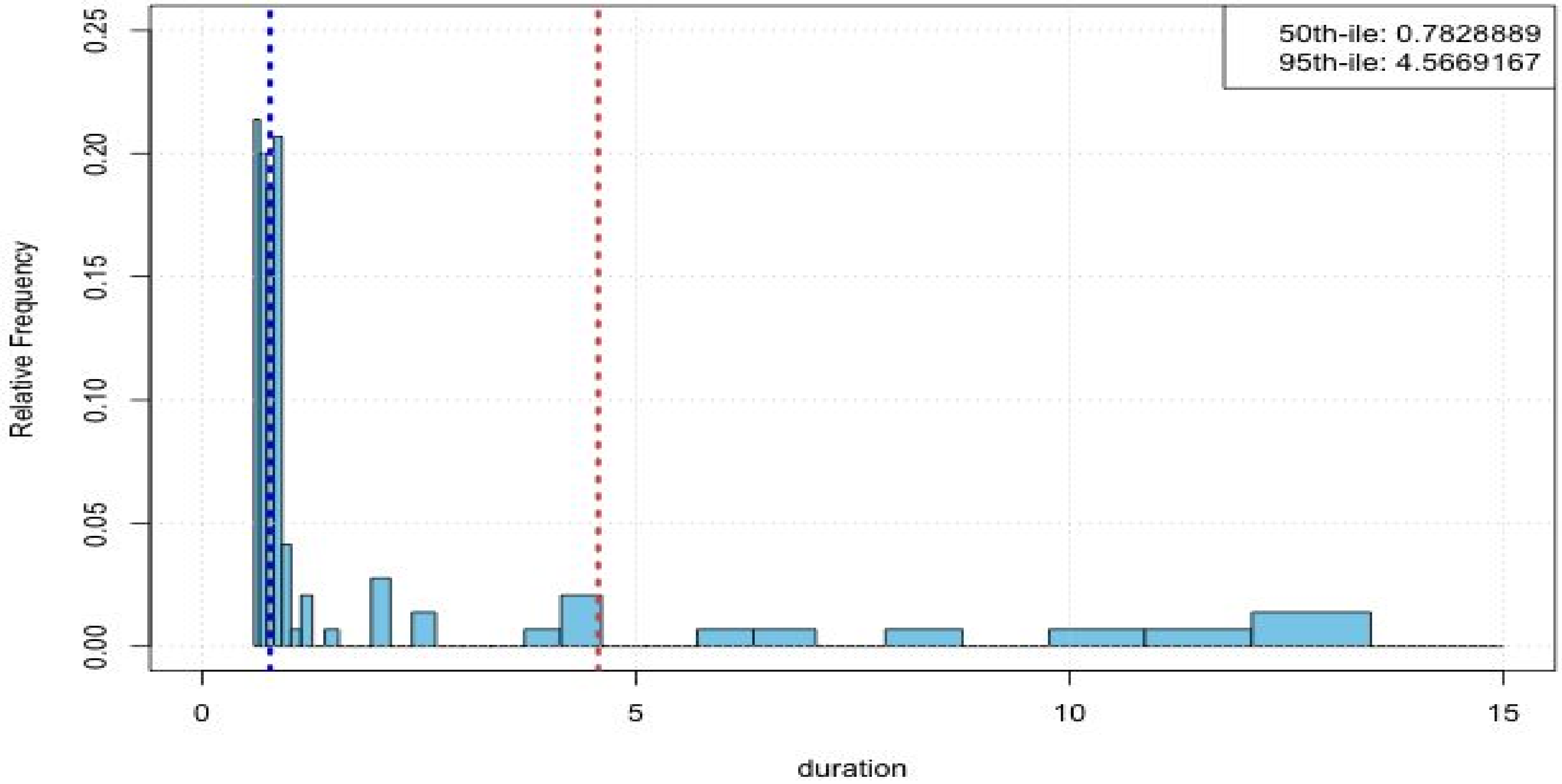
- Calculate the inverse empirical cumulative distribution function by math.
    - ...or just use R which is free and knows how to do it already

```
> library(HistogramTools)
> iecdf <- HistToEcdf(histogram,
              method='linear’, inverse=TRUE)

> iecdf(0.5)
[1] 0.7975309  # median
> iecdf(0.95)
[1] 4.65       # 95th percentile
```




50th-ile: 0.7975309
95th-ile: 4.6500000
But constant sized linear spaced bins use a lot of data where we're not interested.
Relative Frequency
0.0
0.1
0.2
0.3
0.4
0.5
0.6
duration
0
5
10
15


50th-ile: 0.7828889
95th-ile: 4.5669167
Relative Frequency
0.00
0.05
0.10
0.15
0.20
0.25
0
5
10
15
duration

Data > Opinions.

Better than debating opinions.

"We live in a 50ms world!"

"No one really minds the spinner."

"Why should we spend time on that instead of COOLFEATURE?"

"There's no way that the client makes that many requests."

Architecture is **hard**. Make it cheap to experiment where your users really are.

We built Daedalus

Fast

Slow

4	5	6	7	8	9	10
1.51%	1.72%	1.58%	1.77%	1.76%	1.75%	1.83%
3.21%	2.89%	2.80%	3.09%	3.18%	2.99%	2.89%
4.58%	4.68%	4.62%	4.46%	4.42%	4.50%	4.36%
8.98%	9.46%	8.44%	8.81%	8.76%	8.93%	8.77%
9.70%	9.20%	9.65%	9.23%	9.15%	9.00%	9.13%
11.90%	11.70%	11.83%	11.93%	11.85%	11.72%	11.96%
12.53%	12.64%	12.56%	11.94%	12.40%	12.46%	12.22%
10.03%	9.78%	9.54%	9.66%	10.24%	10.24%	9.80%
6.57%	6.67%	6.61%	6.50%	6.75%	6.87%	6.85%
4.40%	4.54%	4.71%	4.69%	4.49%	4.65%	4.68%
3.43%	3.37%	3.37%	3.66%	3.68%	3.59%	3.63%
2.67%	2.75%	2.83%	2.72%	2.65%	2.76%	2.79%
2.16%	2.14%	2.29%	2.43%	2.17%	2.11%	1.90%
2.04%	2.14%	2.29%	2.02%	1.97%	1.94%	2.10%
2.10%	2.34%	2.32%	2.29%	2.10%	2.22%	2.12%
1.94%	1.92%	1.99%	1.89%	1.95%	1.81%	1.82%
1.26%	1.30%	1.36%	1.33%	1.27%	1.36%	1.33%
1.34%	1.23%	1.36%	1.35%	1.17%	1.36%	1.26%
1.28%	1.20%	1.34%	1.40%	1.19%	1.30%	1.35%
1.36%	1.50%	1.23%	1.58%	1.37%	1.40%	1.46%
1.37%	1.44%	1.65%	1.48%	1.51%	1.33%	1.41%
0.71%	0.76%	0.76%	0.79%	0.79%	0.81%	0.92%
0.27%	0.20%	0.30%	0.30%	0.42%	0.33%	0.41%
0.48%	0.45%	0.47%	0.47%	0.49%	0.41%	0.41%
0.59%	0.44%	0.48%	0.48%	0.48%	0.46%	0.62%
0.56%	0.73%	0.57%	0.48%	0.63%	0.70%	0.75%
0.69%	0.46%	0.62%	0.64%	0.62%	0.56%	0.73%
0.35%	0.37%	0.32%	0.29%	0.32%	0.37%	0.35%
0.28%	0.28%	0.19%	0.49%	0.33%	0.31%	0.31%
0.20%	0.21%	0.15%	0.21%	0.31%	0.30%	0.25%
0.45%	0.50%	0.57%	0.57%	0.48%	0.43%	0.54%
0.23%	0.23%	0.20%	0.24%	0.21%	0.20%	0.29%
0.16%	0.14%	0.22%	0.31%	0.19%	0.15%	0.22%
0.09%	0.11%	0.10%	0.11%	0.19%	0.16%	0.09%
0.04%	0.07%	0.11%	0.06%	0.12%	0.14%	0.07%
0.33%	0.30%	0.36%	0.19%	0.21%	0.24%	0.23%
0.13%	0.11%	0.10%	0.09%	0.10%	0.08%	0.11%
0.09%	0.04%	0.11%	0.04%	0.04%	0.07%	0.04%

DNS Time

US

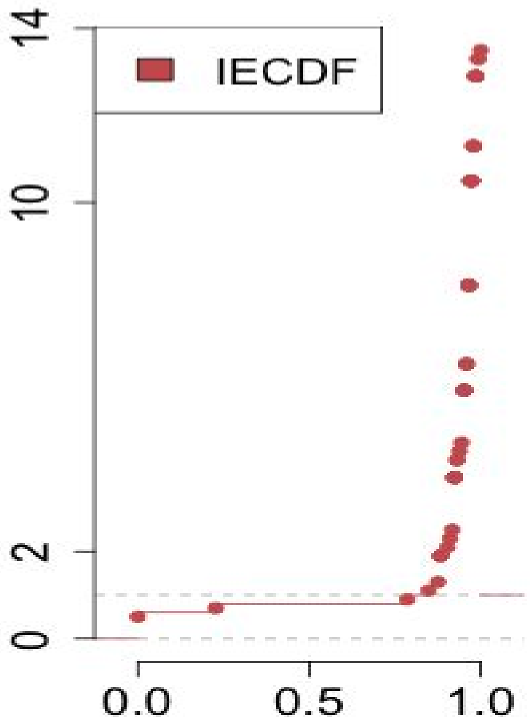
Elsewhere

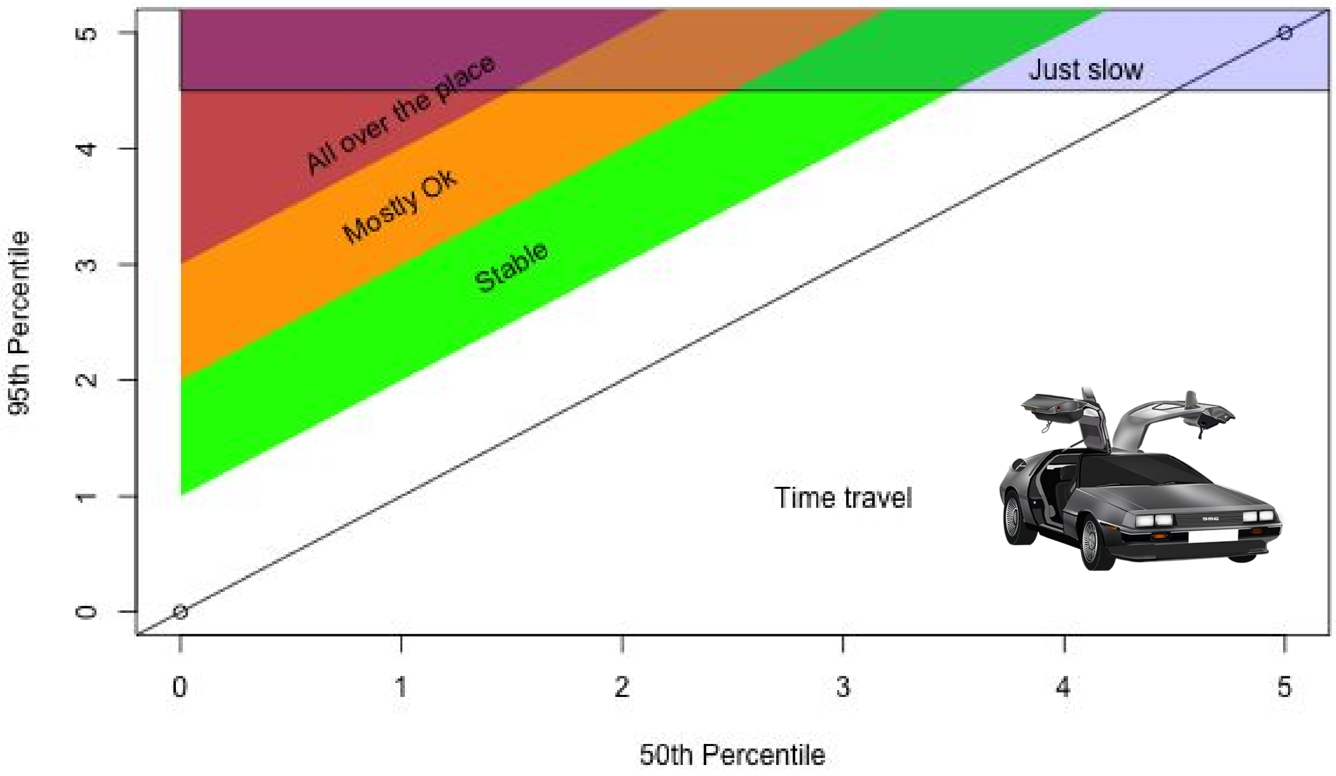
4	5	6	7	8	9	10
6.90%	1.35%	2.74%	3.61%	2.64%	5.07%	4.74%
3.45%	4.05%	1.83%	2.65%	3.36%	2.69%	2.77%
3.45%	5.41%	1.37%	3.13%	2.88%	3.62%	2.15%
0.00%	4.05%	2.74%	4.82%	3.84%	3.10%	4.47%
3.45%	2.70%	1.37%	3.86%	3.24%	2.48%	3.58%
0.00%	5.41%	10.96%	4.34%	4.80%	5.17%	5.19%
3.45%	1.35%	5.02%	4.82%	3.12%	3.72%	4.20%
10.34%	2.70%	7.31%	5.06%	5.04%	4.55%	6.26%
10.34%	14.86%	9.13%	6.51%	3.96%	3.62%	5.46%
10.34%	9.46%	12.79%	7.95%	5.64%	4.55%	5.72%
6.90%	6.76%	8.22%	8.67%	5.52%	6.41%	7.69%
0.00%	0.00%	8.68%	9.16%	7.31%	5.48%	6.71%
10.34%	5.41%	10.05%	8.43%	7.55%	8.38%	6.17%
0.00%	10.81%	5.94%	7.71%	6.71%	11.58%	6.26%
6.90%	0.00%	3.65%	2.65%	3.12%	5.58%	4.47%
0.00%	5.41%	2.74%	2.17%	3.96%	2.79%	2.95%
0.00%	10.81%	1.37%	0.24%	3.24%	1.55%	0.98%
3.45%	2.70%	0.00%	0.96%	4.08%	1.14%	0.36%
3.45%	1.35%	0.46%	0.96%	3.36%	2.59%	1.88%
0.00%	0.00%	0.00%	0.24%	3.24%	1.96%	2.68%
6.90%	0.00%	0.91%	0.48%	1.20%	3.41%	0.63%
0.00%	1.35%	0.91%	0.24%	0.48%	1.14%	1.43%
0.00%	0.00%	0.00%	0.48%	0.24%	1.34%	0.63%
6.90%	0.00%	0.91%	0.48%	1.32%	0.52%	0.72%
0.00%	1.35%	0.46%	0.24%	0.36%	0.62%	2.42%
0.00%	0.00%	0.46%	0.72%	1.80%	0.93%	2.50%
0.00%	1.35%	0.00%	2.65%	2.28%	1.24%	2.95%
0.00%	0.00%	0.00%	0.48%	0.72%	0.52%	0.54%
0.00%	0.00%	0.00%	0.24%	2.28%	0.62%	0.36%
0.00%	0.00%	0.00%	0.48%	1.08%	0.21%	0.27%
0.00%	0.00%	0.00%	4.82%	0.00%	1.96%	0.81%
0.00%	0.00%	0.00%	0.00%	1.08%	1.03%	1.52%
0.00%	0.00%	0.00%	0.24%	0.00%	0.10%	0.18%
0.00%	0.00%	0.00%	0.00%	0.24%	0.10%	0.09%
0.00%	0.00%	0.00%	0.24%	0.00%	0.10%	0.18%
3.45%	0.00%	0.00%	0.24%	0.12%	0.00%	0.00%
0.00%	0.00%	0.00%	0.00%	0.00%	0.10%	0.09%
0.00%	1.35%	0.00%	0.00%	0.24%	0.00%	0.00%

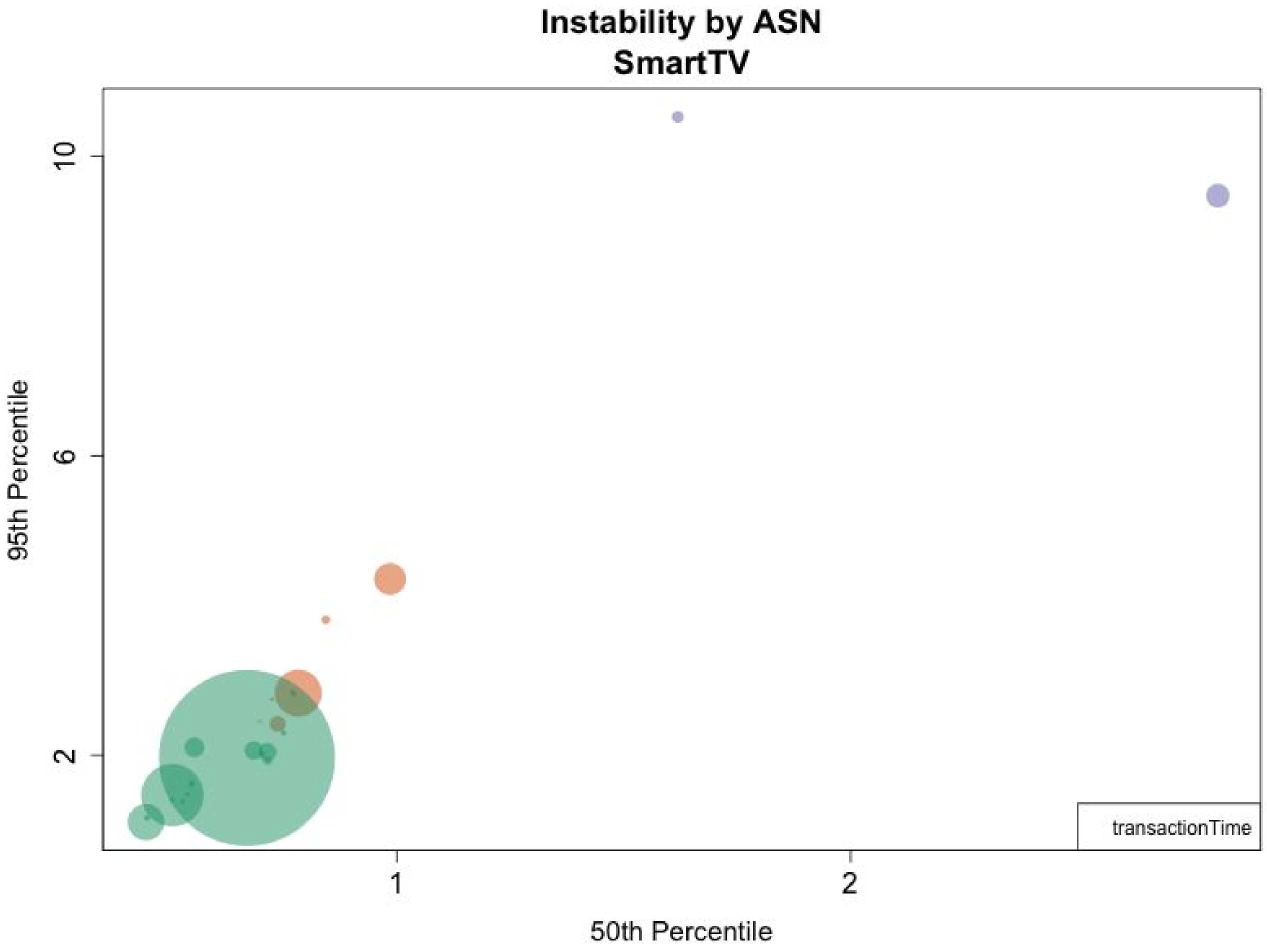
Interpret the data

- Visual → Numerical, need the IECDF for Percentiles
  - $f(0.50) = 50^{th}$ (median)
  - $f(0.95) = 95^{th}$

- Cluster to get ~~pretty colors~~ similar experiences.
  (k-means, hierarchical, etc.)



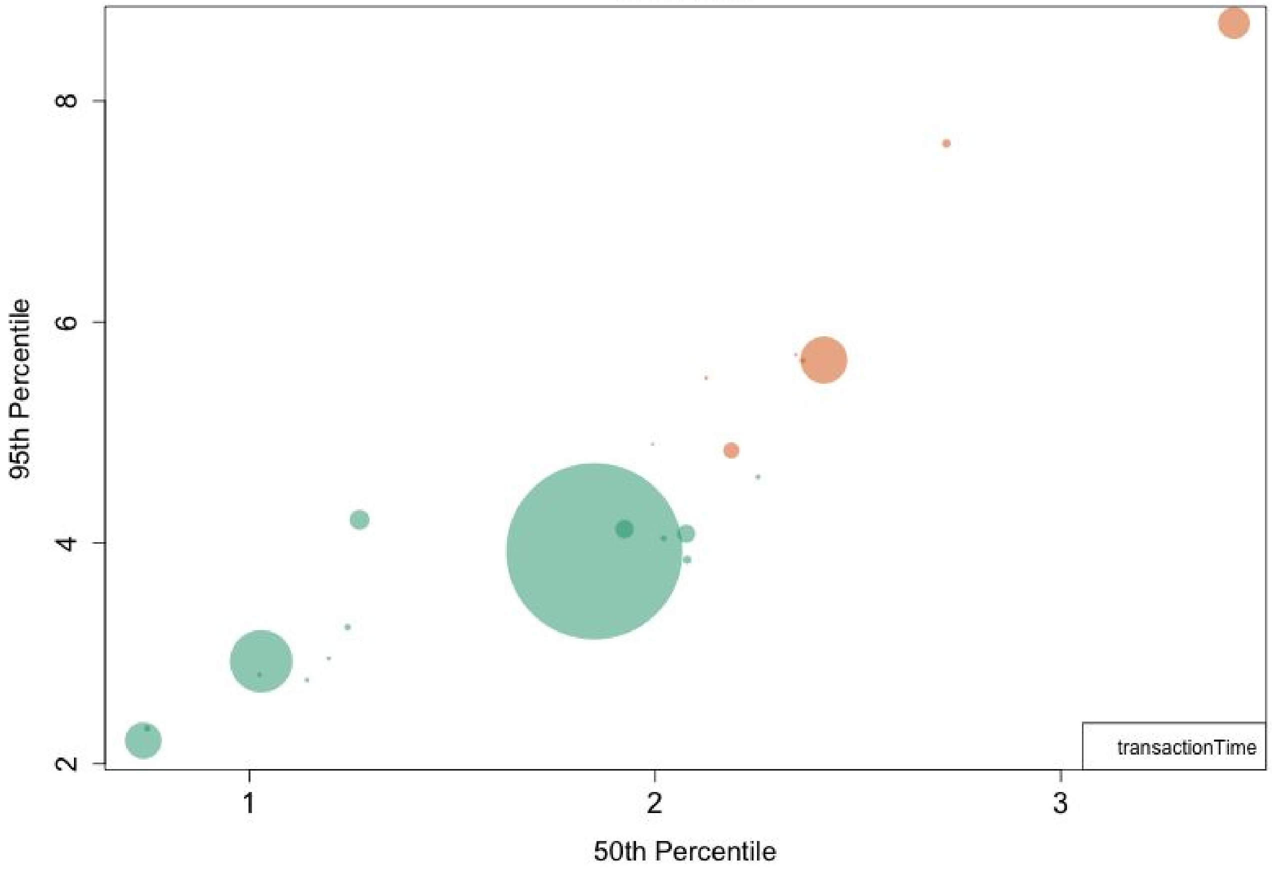

All over the place
Mostly Ok
Stable
Just slow
Time travel
95th Percentile
50th Percentile
0
1
2
3
4
5

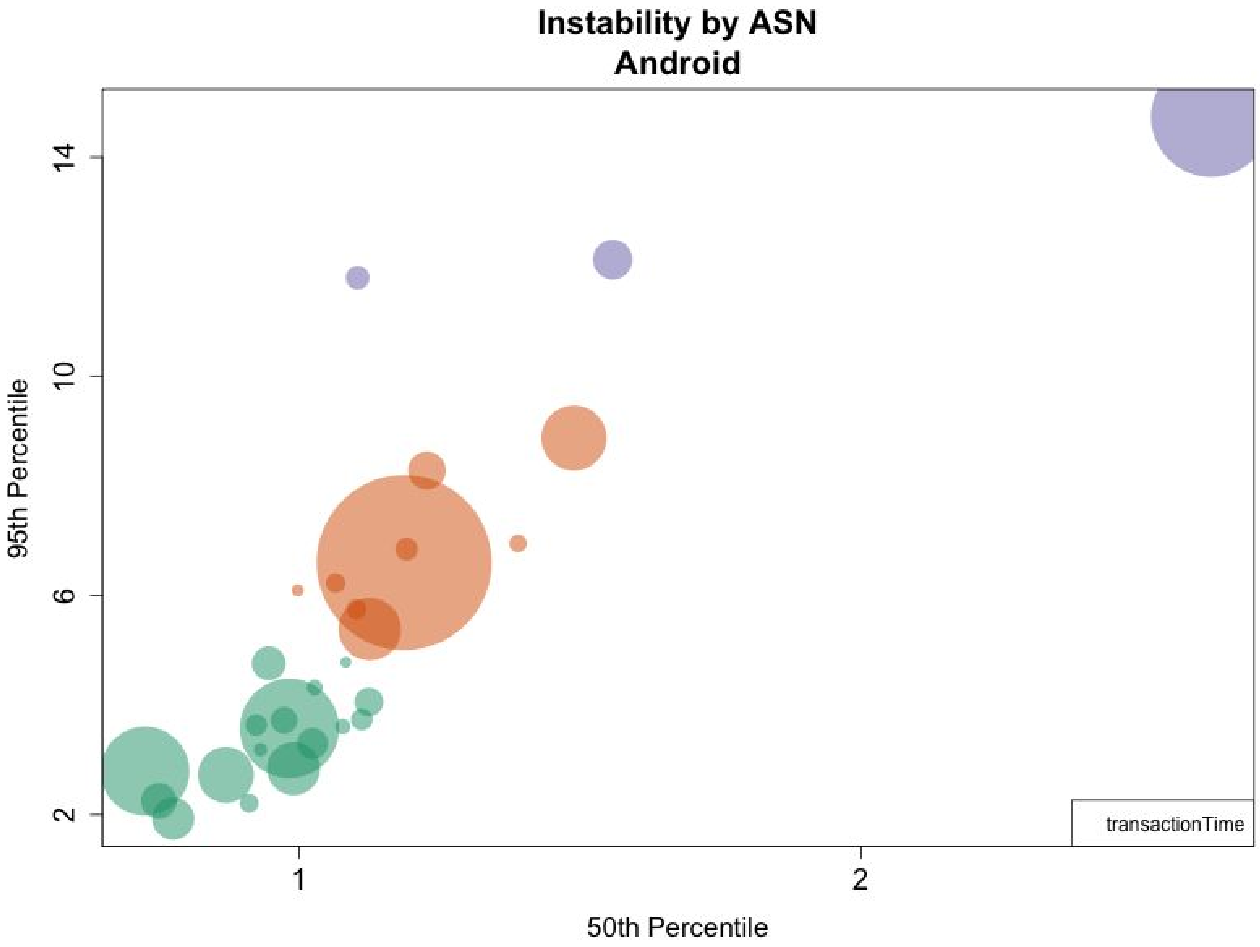


Instability by ASN
SmartTV
95th Percentile
10
6
2
1
2
50th Percentile
transactionTime

NETFLIX

Instability by ASN
SmartTV




Instability by ASN
Android
14
10
6
2
95th Percentile
1
2
50th Percentile
transactionTime

NETFLIX

Practical Teleportation.

- Go there!
- Abstract analysis - hard
- Feeling reality is much simpler than looking at graphs. Build!

Make a Reality Lab.


ISP: Ora[redacted]nge
CN: France

Don't guess.

Developing a model based on production data, without missing the distribution of samples (network, render, responsiveness) will lead to **better software.**

Global reach doesn't need to be scary.

@gcirino42 http://blogofsomeguy.com

Icarus

Martin Spier
@spiermar

Performance Engineering @ Netflix

Problem & Motivation

- Real-user performance monitoring solution
- More insight into the App performance (as perceived by real users)
- Too many variables to trust synthetic tests and labs
- Prioritize work around App performance
- Track App improvement progress over time
- Detect issues, internal and external

Device Diversity

- Netflix runs on all sorts of devices
- Smart TVs, Gaming Consoles, Mobile Phones, Cable TV boxes, ...
- Consistently evaluate performance


#netflixeverywhere

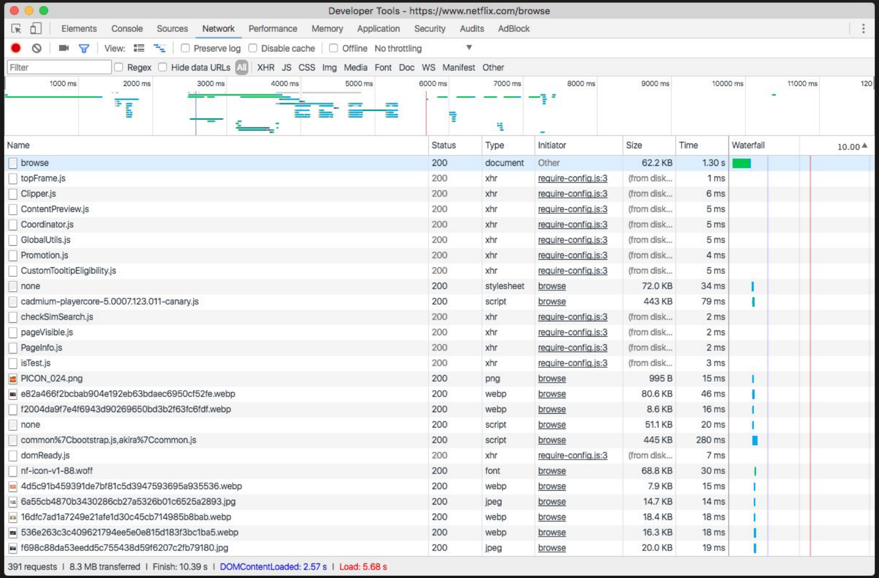
What are we monitoring?

- User Actions

  (or things users do in the App)
- App Startup
- User Navigation
- Playing a Title
- *Internal App metrics*

What are we measuring?

- When does the timer start and stop?
- Time-to-Interactive (TTI)
    - Interactive, even if some items were not fully loaded and rendered
- Time-to-Render (TTR)
    - Everything above the fold (visible without scrolling) is rendered
- Play Delay
- Meaningful for what we are monitoring



High-dimensional Data

- Complex device categorization
- Geo regions, subregions, countries
- Highly granular network classifications
- High volume of A/B tests
- Different facets of the same user action
  - Cold, suspended and backgrounded App startups
  - Target view/page on App startup

kafka

HIVE

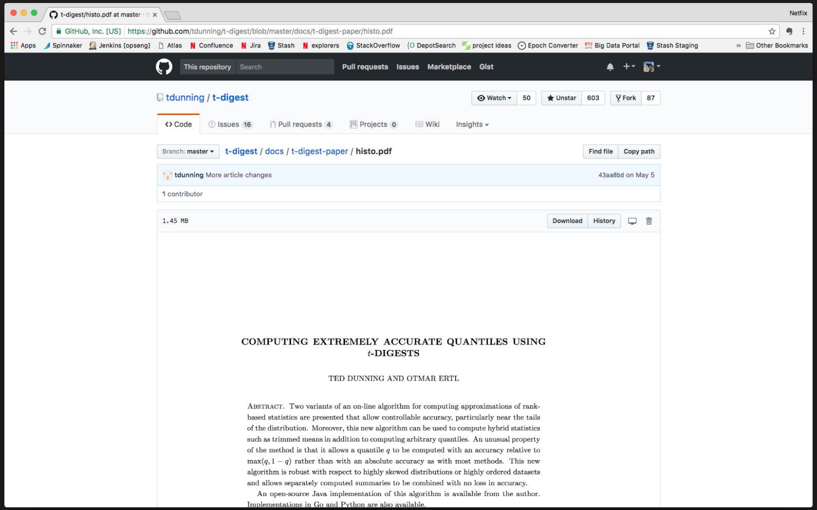
druid

Data Sketches

- Data structures that *approximately* resemble a much larger data set
- Preserve essential features!
- Significantly smaller!
- Faster to operate on!

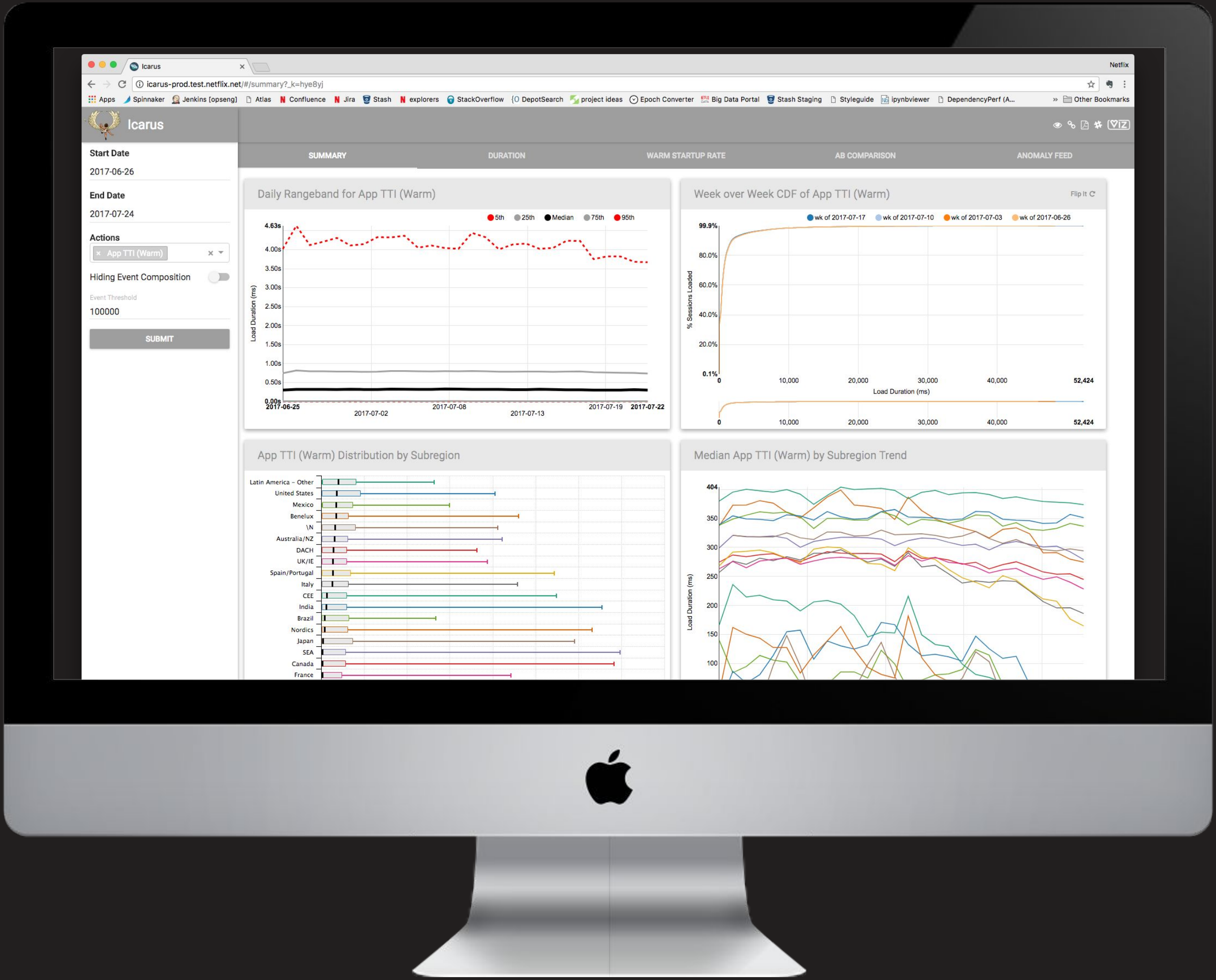
t-Digest

- t-Digest data structure
- Rank-based statistics (such as quantiles)
- Parallel friendly (can be merged!)
- Very fast!
- Really accurate!



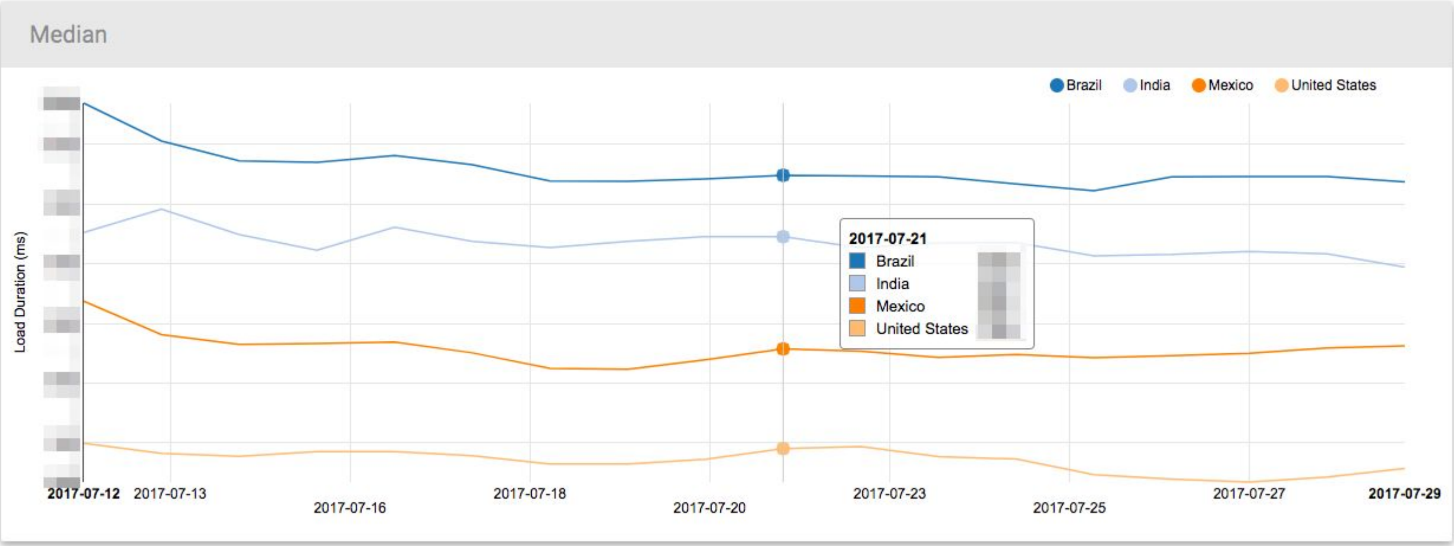
https://github.com/tdunning/t-digest

+ t-Digest sketches

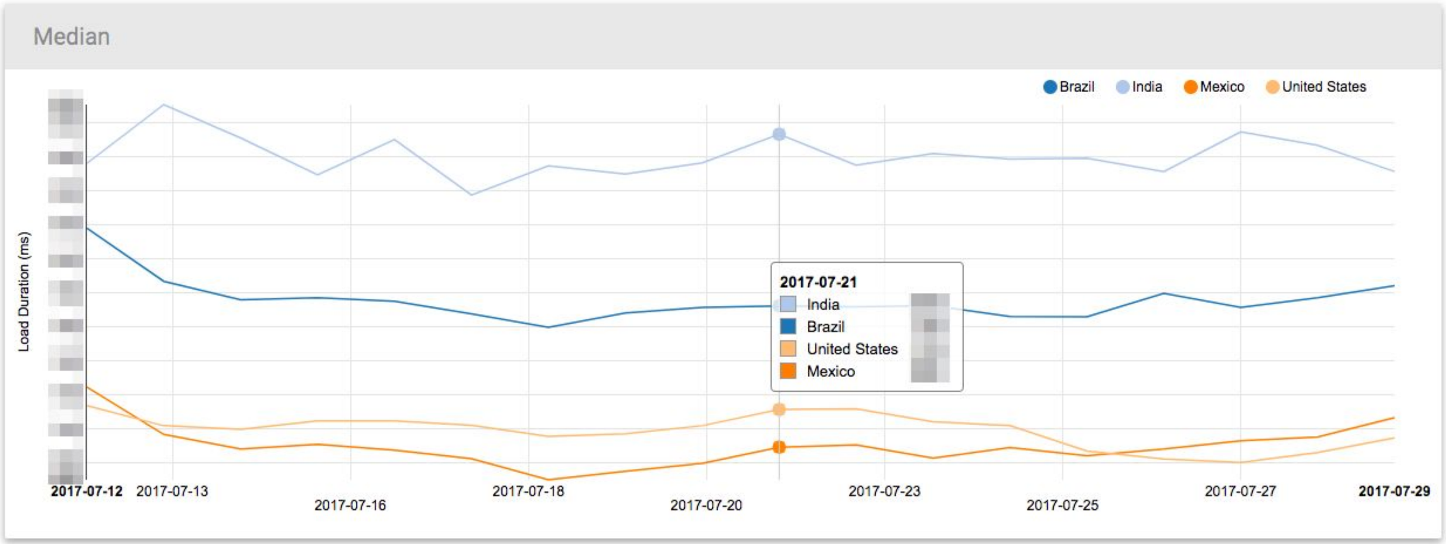


Icarus
icarus-prod.test.netflix.net/#/summary?_k=hye8yj
Netflix
Apps
Spinnaker
Jenkins [opseng]
Atlas
Confluence
Jira
Stash
explorers
StackOverflow
DepotSearch
project ideas
Epoch Converter
Big Data Portal
Stash Staging
Styleguide
ipynbviewer
Other Bookmarks
Icarus
Start Date
2017-06-26
End Date
2017-07-24
Actions
App TTI (Warm)
Hiding Event Composition
Event Threshold
100000
SUBMIT
SUMMARY
DURATION
WARM STARTUP RATE
AB COMPARISON
ANOMALY FEED
Daily Rangeband for App TTI (Warm)
5th
25th
Median
75th
95th
Load Duration (ms)
2017-06-25
2017-07-02
2017-07-08
2017-07-13
2017-07-19
2017-07-22
Week over Week CDF of App TTI (Warm)
Flip It
wk of 2017-07-17
wk of 2017-07-10
wk of 2017-07-03
wk of 2017-06-26
% Sessions Loaded
Load Duration (ms)
App TTI (Warm) Distribution by Subregion
Latin America - Other
United States
Mexico
Benelux
\N
Australia/NZ
DACH
UK/IE
Spain/Portugal
Italy
CEE
India
Brazil
Nordics
Japan
SEA
Canada
France
Median App TTI (Warm) by Subregion Trend
Load Duration (ms)

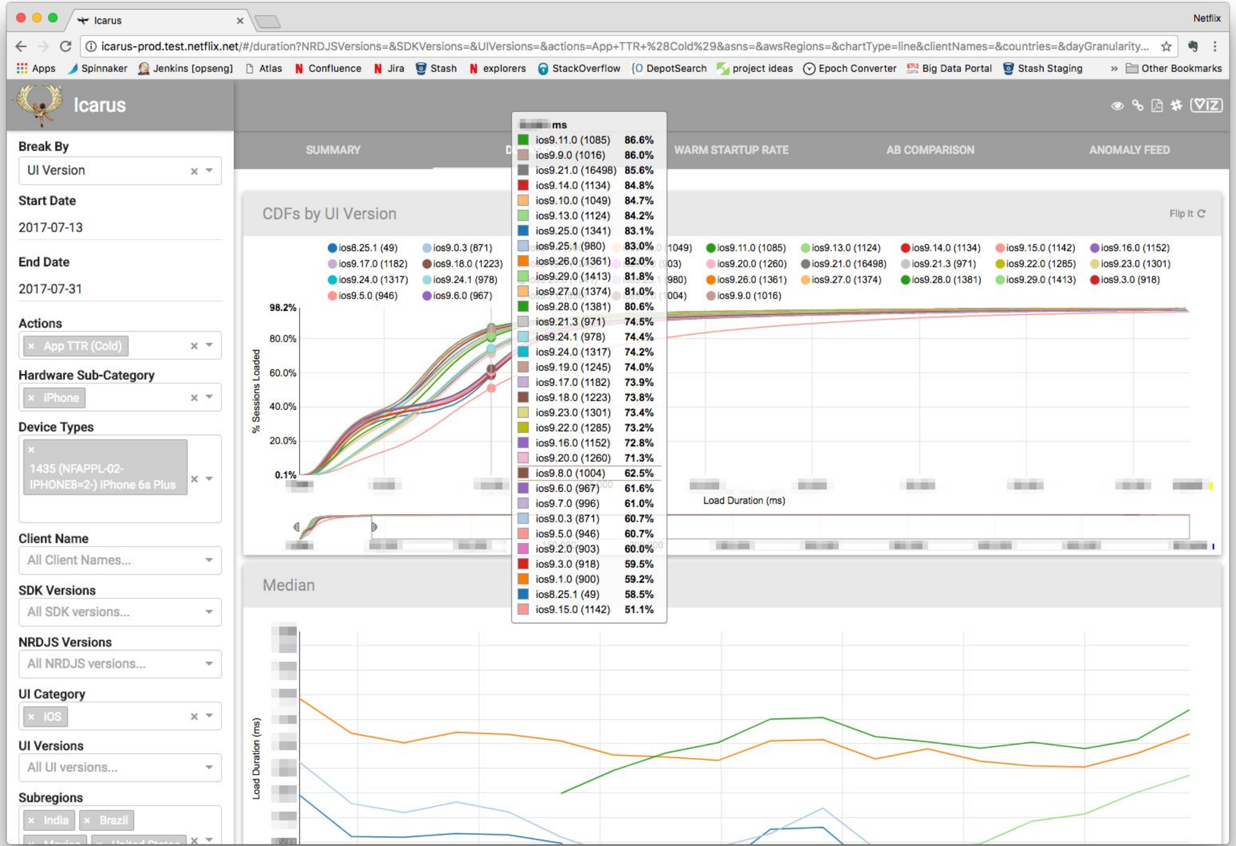
iOS Median Comparison, Break by Country



iOS Median Comparison, Break by Country + iPhone 6S Plus

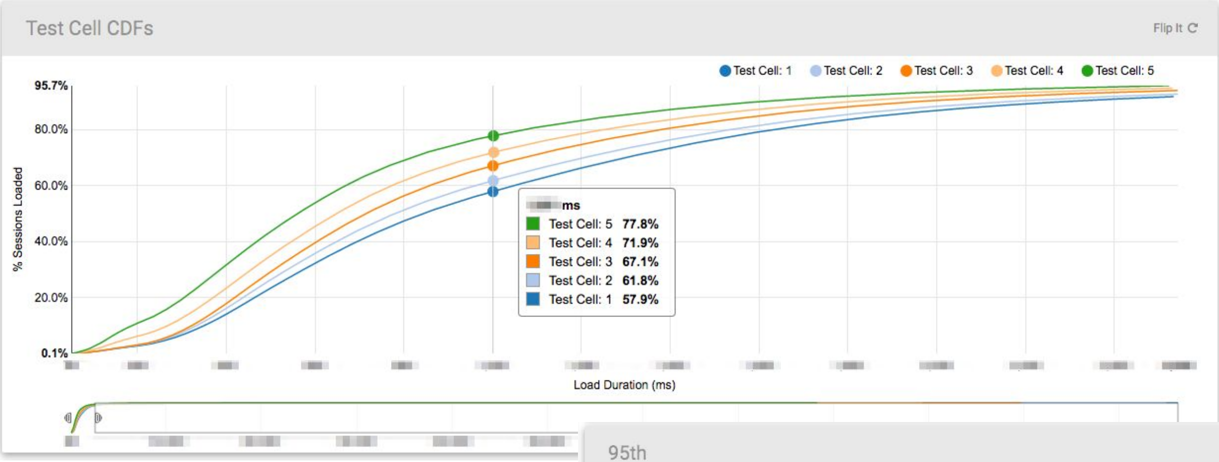


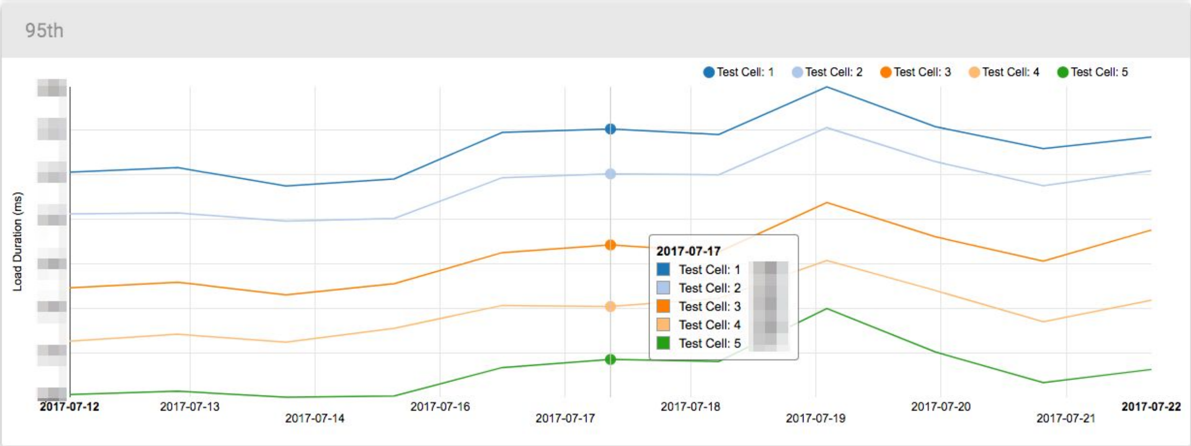
CDFs by UI Version



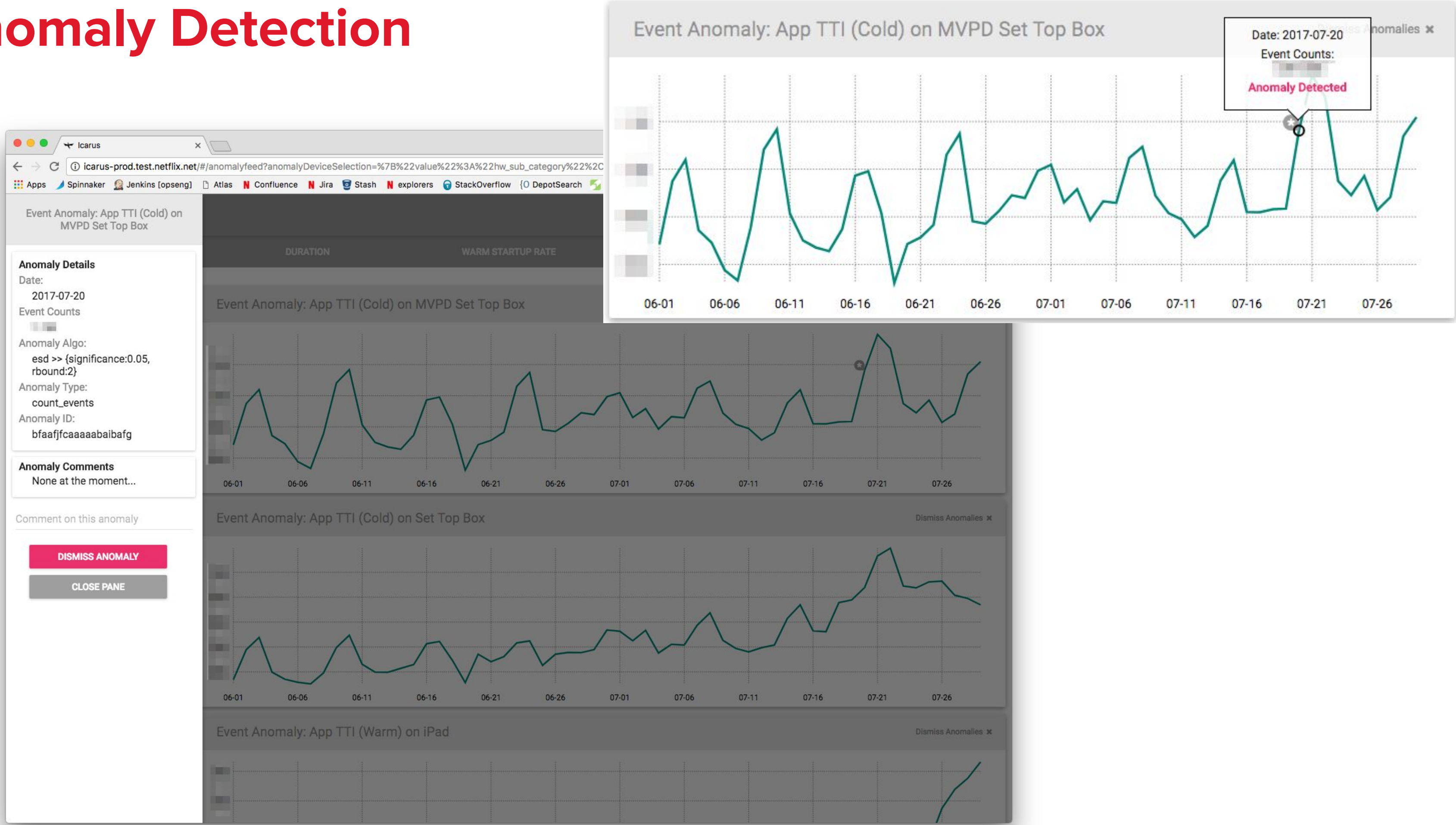
Warm Startup Rate

A/B Cell Comparison





Anomaly Detection



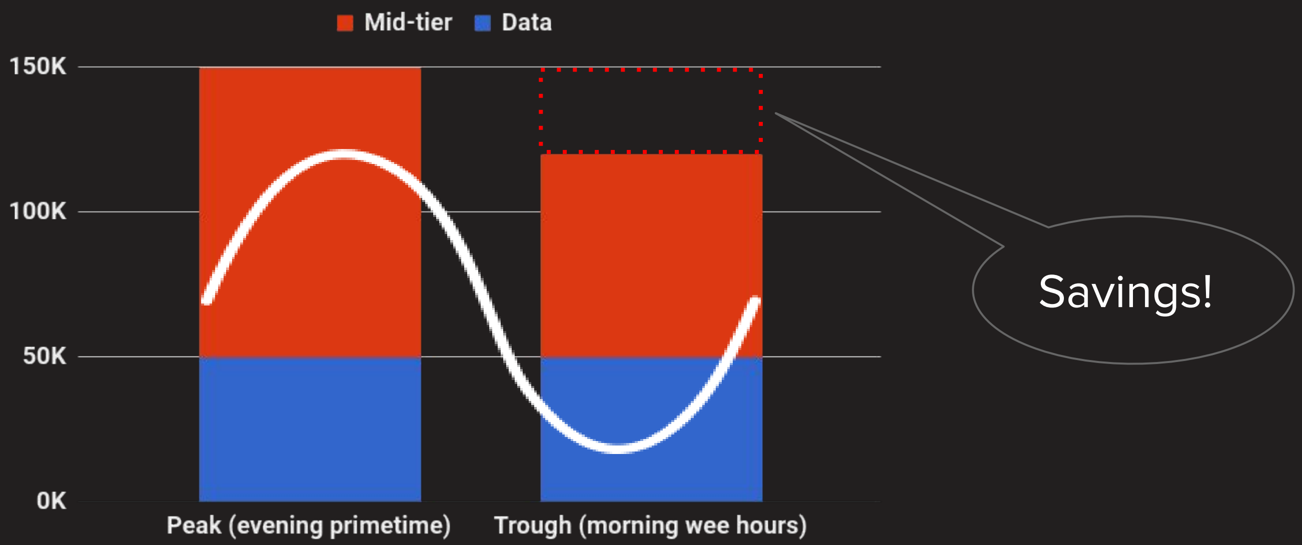
Going Forward

@spiermar

- Resource utilization metrics
- Device profiling
  - Instrumenting client code
- Explore other visualizations
  - Frequency heat maps
- Connection between perceived performance, acquisition and retention

Netflix
Autoscaling for experts

Vadim



- Mid-tier stateless services are ~2/3rd of the total
- Savings - 30% of mid-tier footprint (roughly 30K instances)
  - Higher savings if we break it down by region
  - Even higher savings on services that scale well

Why we autoscale - philosophical reasons

Why we autoscale - pragmatic reasons

- Encoding
- Precompute
- Failover
- Red/black pushes
- Curing cancer**
- And more...

** Hack-day project

Should you autoscale?

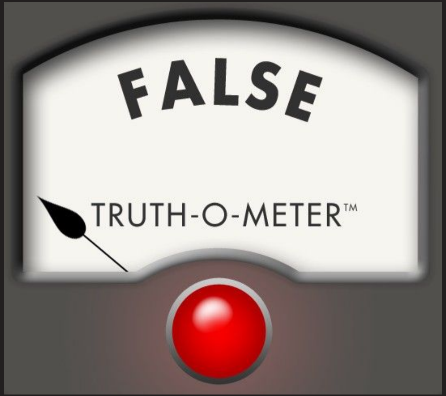
Benefits

- On-demand capacity: direct $$ savings
- RI capacity: re-purposing spare capacity

However, for each server group, beware of

- Uneven distribution of traffic
- Sticky traffic
- Bursty traffic
- Small ASG sizes (<10)

Autoscaling impacts availability - true or false?

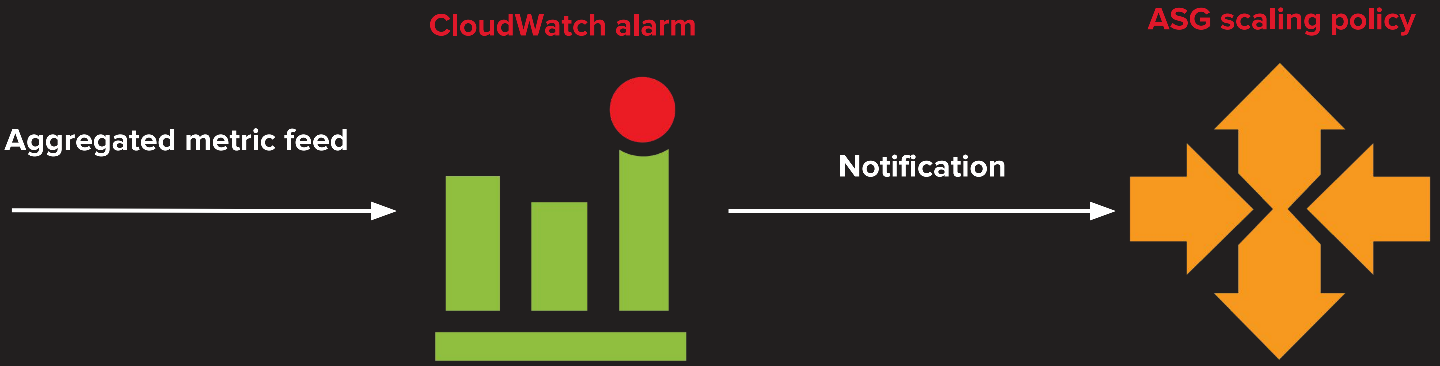


* If done correctly

Under-provisioning, however, can impact availability

- Autoscaling is not a problem
- The real problem is not knowing performance characteristics of the service

AWS autoscaling mechanics



Tunables

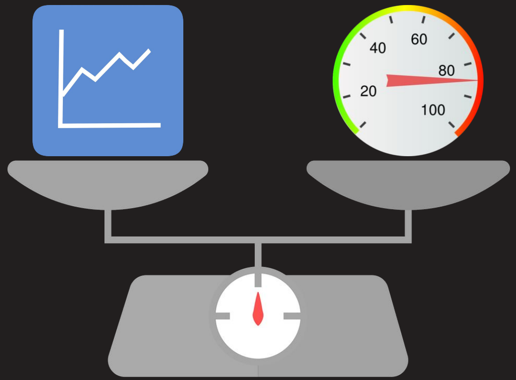
Metric

- Threshold
- # of eval periods

- Scaling amount
- Warmup time

What metric to scale on?

Throughput



Resource utilization

Throughput	Resource utilization
Pros	
• Tracks a direct measure of work • Linear scaling • Predictable	• Requires less adjustment over time
Cons	
• Thresholds tend to drift over time • Prone to changes in request mixture	• Less predictable • More oscillation / jitter

Autoscaling on multiple metrics

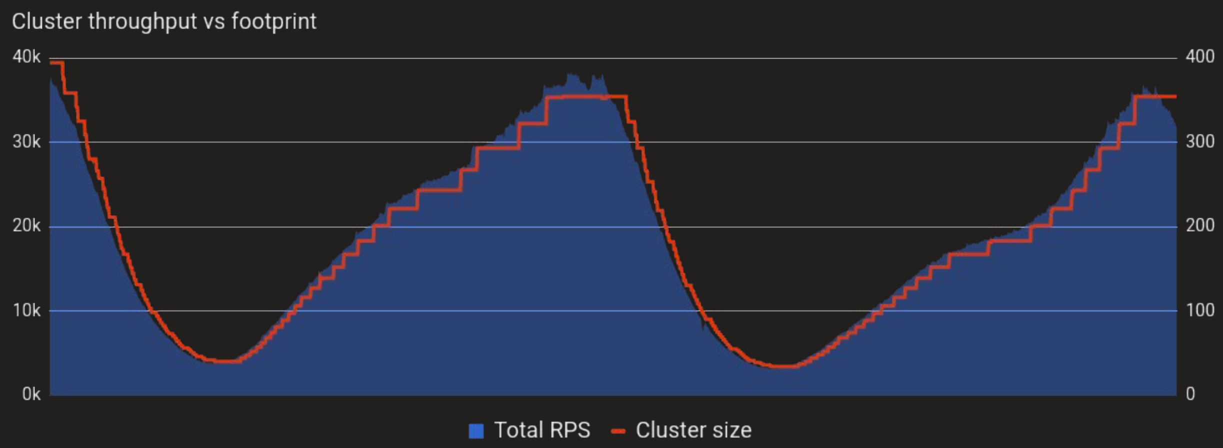
Proceed with caution

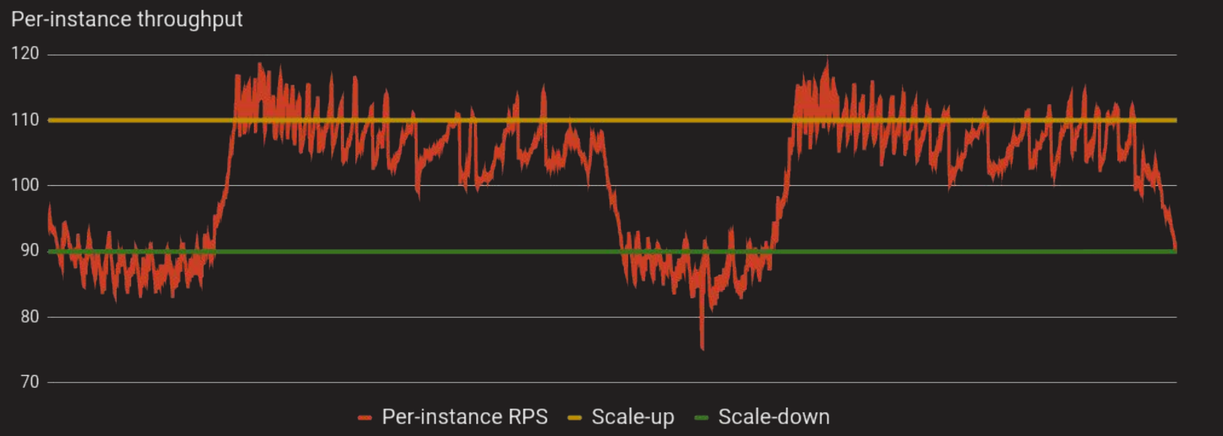
- Harder to reason about scaling behavior
- Different metrics might contradict each other, causing oscillation

Typical Netflix configuration:

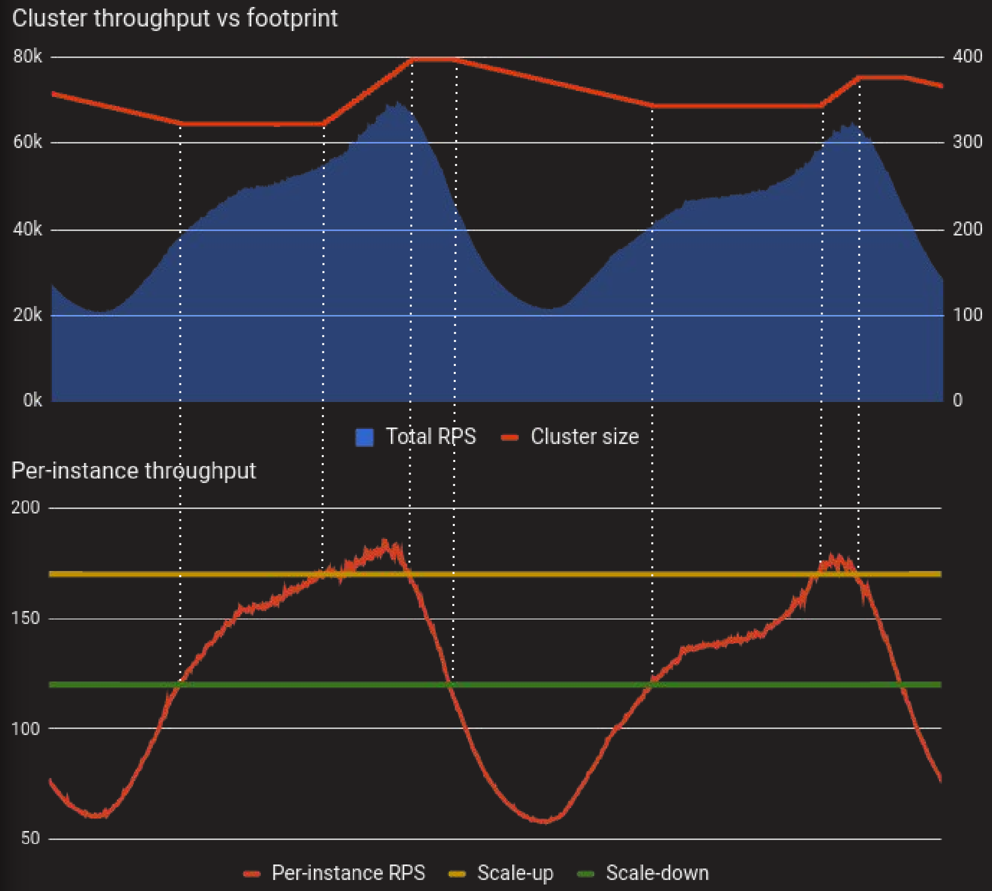
- Scale-up policy on throughput
- Scale-down policy on throughput
- Emergency scale-up policy on CPU, aka “the hammer rule”

Well-behaved autoscaling





Common mistakes - “no rush” scaling

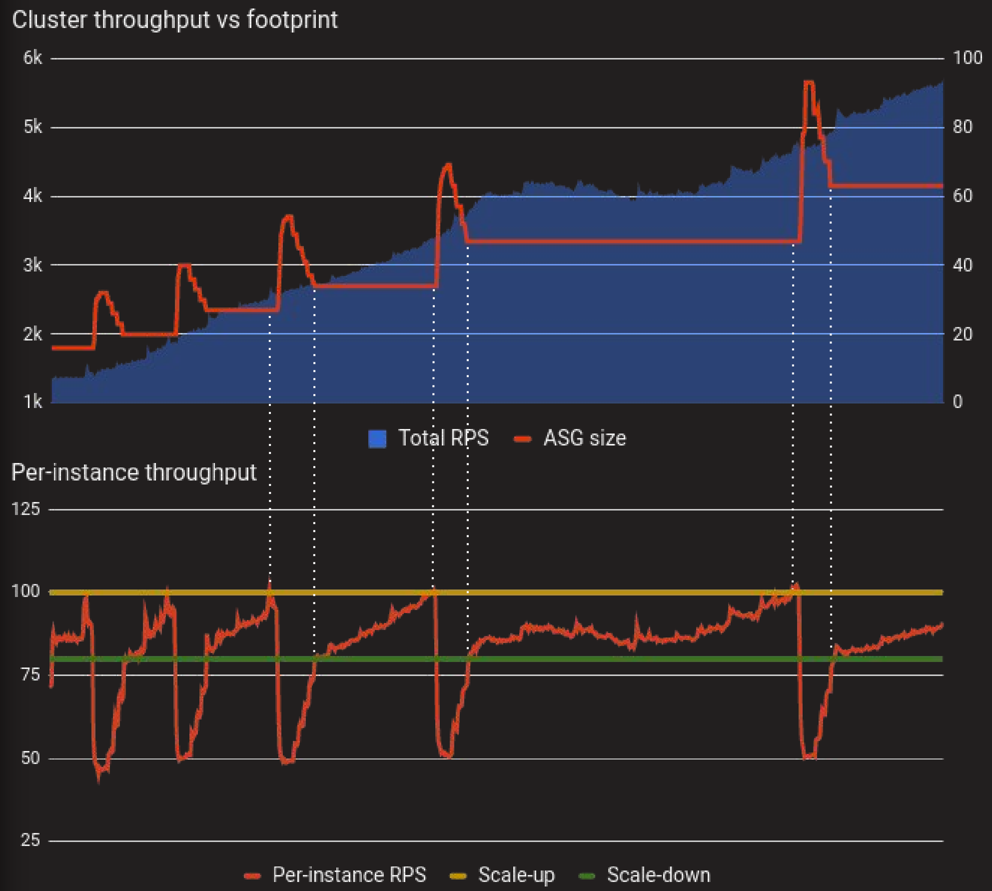


Problem: scaling amounts too small, cooldown too long

Effect: scaling lags behind the traffic flow. Not enough capacity at peak, capacity wasted in trough

Remedy: increase scaling amounts, migrate to step policies

Common mistakes - twitchy scaling

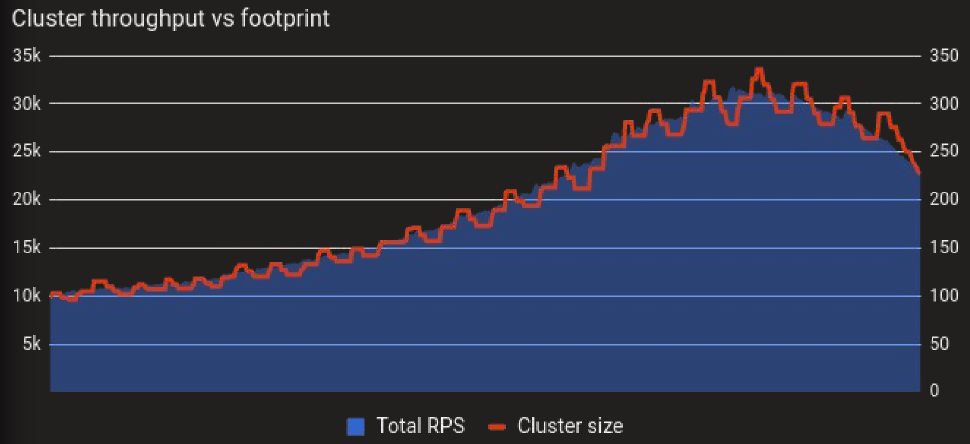


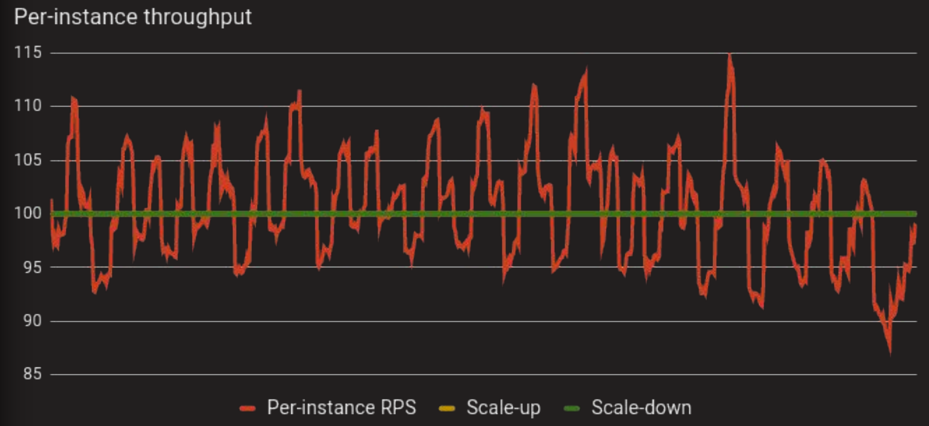
Problem: Scale-up policy is too aggressive

Effect: unnecessary capacity churn

Remedy: reduce scale-up amount, increase the # of eval periods

Common mistakes - should I stay or should I go





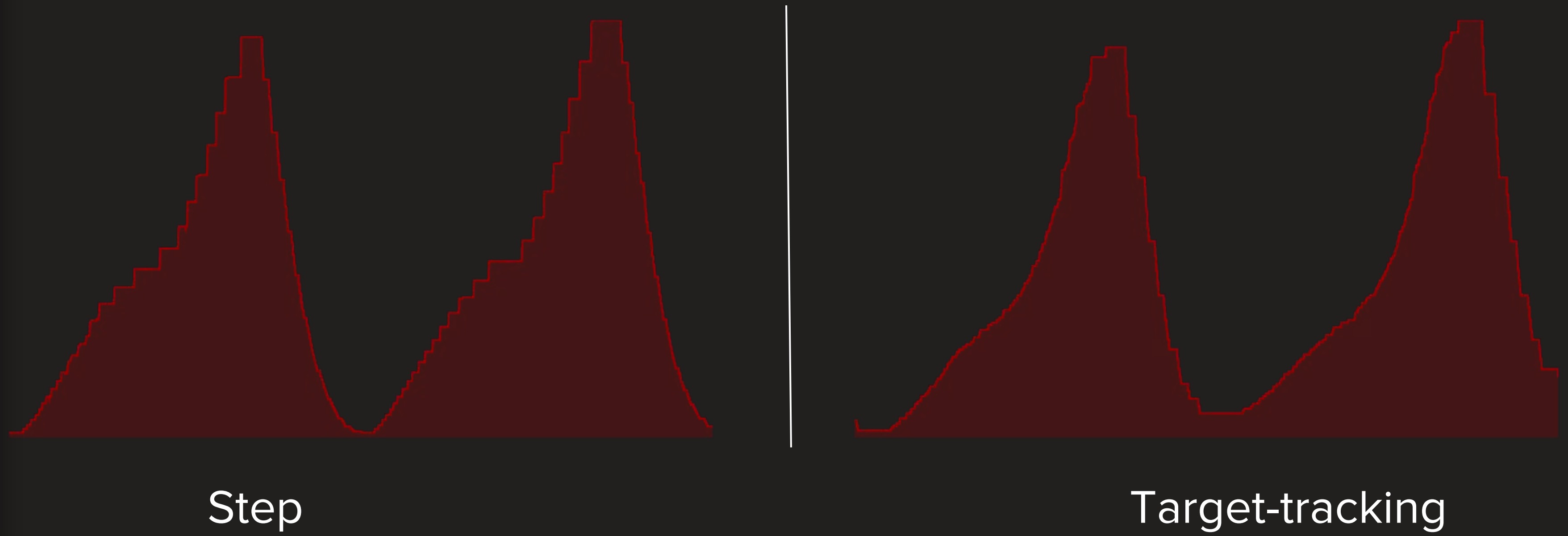
Problem: -up and -down thresholds are too close to each other

Effect: constant capacity oscillation

Remedy: move -up and -down thresholds farther apart

AWS target tracking - your best bet!

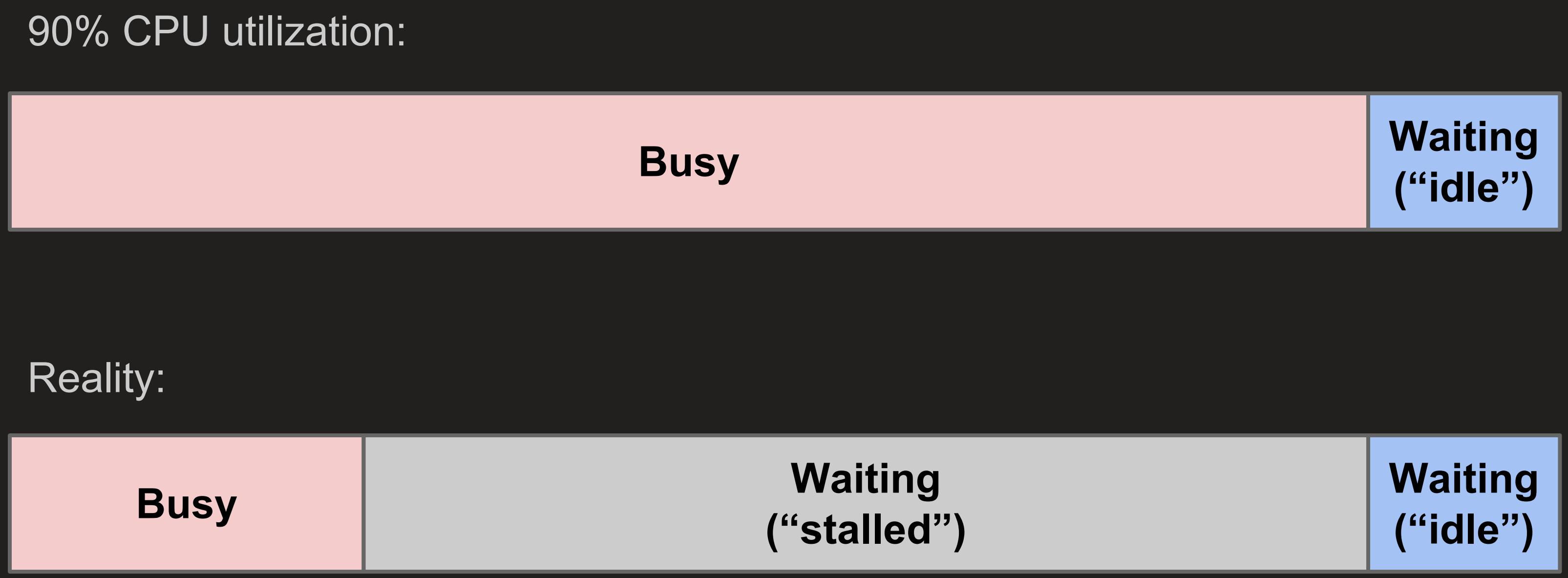
- Think of it as a step policy with auto-steps
- You can also think of it as a thermostat
- Accounts for the rate of change in monitored metric
- Pick a metric, set the target value and warmup time - that’s it!



Netflix
PMCs on the Cloud

Brendan

90% CPU utilization:


90% CPU utilization:
Busy
Waiting (“idle”)
Reality:
Busy
Waiting (“stalled”)
Waiting (“idle”)

```
# perf stat -a -- sleep 10

 Performance counter stats for 'system wide':

     80018.188438      task-clock (msec)    #    8.000 CPUs utilized    (100.00%)
            7,562      context-switches     #    0.095 K/sec            (100.00%)
            1,157      cpu-migrations       #    0.014 K/sec            (100.00%)
          109,734      page-faults          #    0.001 M/sec
  <not supported>      cycles
  <not supported>      stalled-cycles-frontend
  <not supported>      stalled-cycles-backend
  <not supported>      instructions
  <not supported>      branches
  <not supported>      branch-misses

     10.001715965 seconds time elapsed
```

Performance Monitoring Counters (PMCs) in most clouds

```
# perf stat -a -- sleep 10

 Performance counter stats for 'system wide':

     641320.173626 task-clock (msec)         #   64.122 CPUs utilized    [100.00%]
         1,047,222 context-switches          #    0.002 M/sec            [100.00%]
            83,420 cpu-migrations            #    0.130 K/sec            [100.00%]
            38,905 page-faults               #    0.061 K/sec
   655,419,788,755 cycles                    #    1.022 GHz              [75.02%]
   <not supported> stalled-cycles-frontend
   <not supported> stalled-cycles-backend
   536,830,399,277 instructions              #    0.82  insns per cycle  [75.02%]
    97,103,651,128 branches                  #  151.412 M/sec            [75.02%]
     1,230,478,597 branch-misses             #    1.27% of all branches  [74.99%]

      10.001622154 seconds time elapsed
```

AWS EC2 m4.16xl

Interpreting IPC & Actionable Items

IPC: Instructions Per Cycle (invert of CPI)

- IPC < 1.0: likely memory stalled
  - Data usage and layout to improve CPU caching, memory locality.
  - Choose larger CPU caches, faster memory busses and interconnects.
- IPC > 1.0: likely instruction bound
  - Reduce code execution, eliminate unnecessary work, cache operations, improve algorithm order. Can analyze using CPU flame graphs.
  - Faster CPUs.

Intel Architectural PMCs

Event Name	Umask	Event S.	Example Event Mask Mnemonic
UnHalted Core Cycles	00H	3CH	CPU_CLK_UNHALTED.THREAD_P
Instruction Retired	00H	C0H	INST_RETIRED.ANY_P
UnHalted Reference Cycles	01H	3CH	CPU_CLK_THREAD_UNHALTED.REF_XCLK
LLC Reference	4FH	2EH	LONGEST_LAT_CACHE.REFERENCE
LLC Misses	41H	2EH	LONGEST_LAT_CACHE.MISS
Branch Instruction Retired	00H	C4H	BR_INST_RETIRED.ALL_BRANCHES
Branch Misses Retired	00H	C5H	BR_MISP_RETIRED.ALL_BRANCHES

Now available in AWS EC2 on full dedicated hosts (eg, m4.16xl, …)

```
# pmcarch 1
CYCLES        INSTRUCTIONS     IPC BR_RETIRED   BR_MISPRED   BMR% LLCREF      LLCMISS     LLC%
90755342002   64236243785     0.71 11760496978  174052359    1.48 1542464817  360223840  76.65
75815614312   59253317973     0.78 10665897008  158100874    1.48 1361315177  286800304  78.93
65164313496   53307631673     0.82 9538082731   137444723    1.44 1272163733  268851404  78.87
90820303023   70649824946     0.78 12672090735  181324730    1.43 1685112288  343977678  79.59
76341787799   50830491037     0.67 10542795714  143936677    1.37 1204703117  279162683  76.83
[...]
```

https://github.com/brendangregg/pmc-cloud-tools

```
tiptop -                    [root]
Tasks:  96 total,   3 displayed                               screen  0: default

  PID [ %CPU] %SYS    P   Mcycle   Minstr   IPC  %MISS  %BMIS  %BUS COMMAND
 3897   35.3  28.5    4   274.06   178.23  0.65   0.06   0.00   0.0 java
 1319+   5.5   2.6    6    87.32   125.55  1.44   0.34   0.26   0.0 nm-applet
  900    0.9   0.0    6    25.91    55.55  2.14   0.12   0.21   0.0 dbus-daemo
```

Netflix
Performance Meetup

NETFLIX

Netflix
Performance Meetup