How to Cost Effectively and Reliably Build Infrastructure for Machine Learning

Osman Sarood
(Mist Systems)

Strata
DATA CONFERENCE

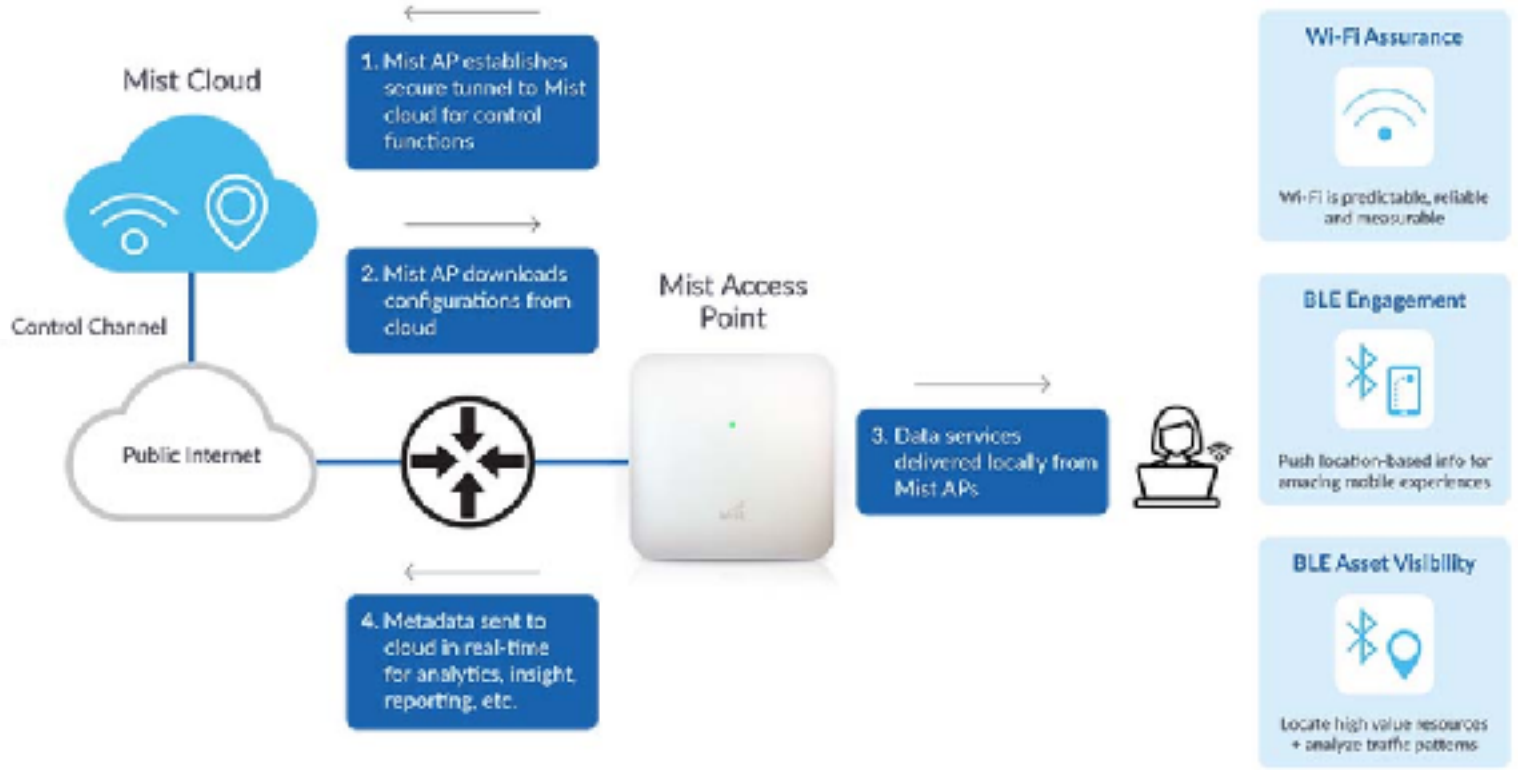
Mist
AI is in the AIR

About Osman

- Fan of performance, reliability, load balancing, scheduling and cost
  - PhD @ UIUC (High Performance Computing)
  - Authored 20+ research papers
  - Speaker at several academic and industry conferences

- Currently @ Mist System
  - Managing infrastructure, data platform and operations teams
  - Tech lead for infrastructure, data platform and operations
  - Software Engineer
  - Seen several 10X growth spurts :)

What's Mist?


Mist Cloud
1. Mist AP establishes secure tunnel to Mist cloud for control functions
2. Mist AP downloads configurations from cloud
Control Channel
Public Internet
Mist Access Point
3. Data services delivered locally from Mist APs
4. Metadata sent to cloud in real-time for analytics, insight, reporting, etc.
Wi-Fi Assurance
Wi-Fi is predictable, reliable and measurable
BLE Engagement
Push location-based info for amazing mobile experiences
BLE Asset Visibility
Locate high value resources + analyze traffic patterns

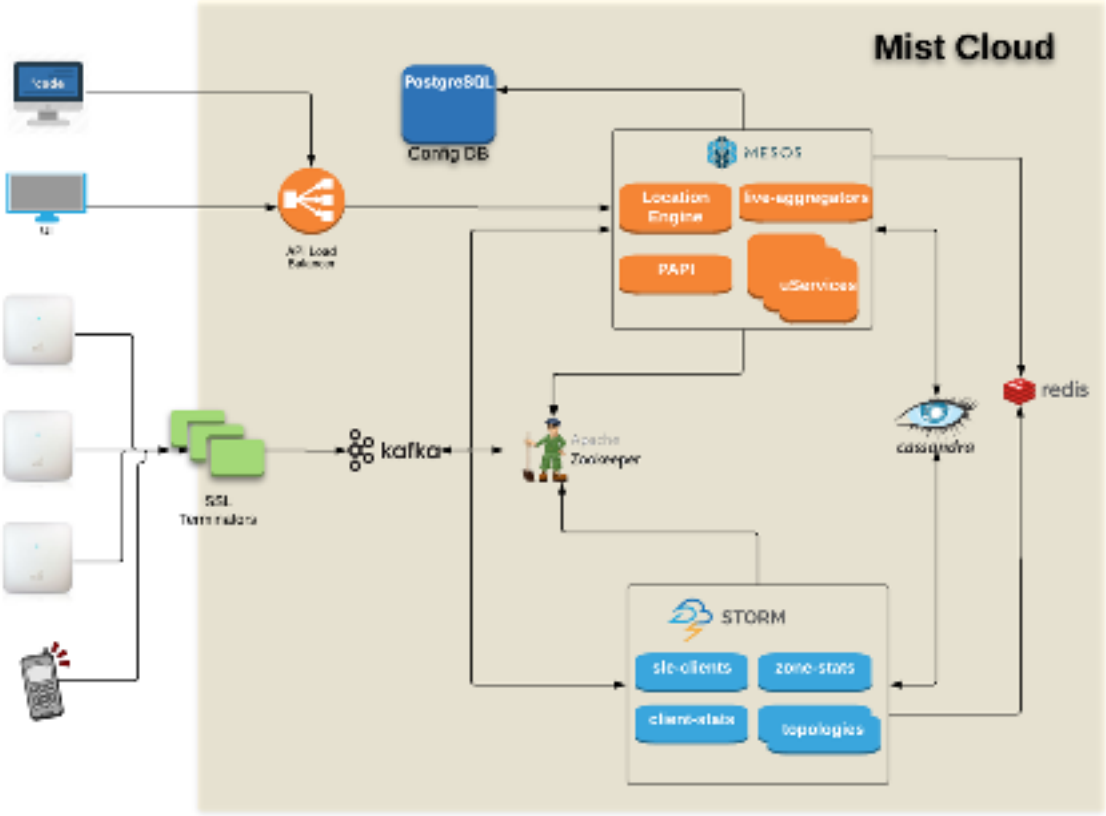
Mist Architecture

1 TB+

500+ partitions

10 Billion+ Msgs

10's TB+

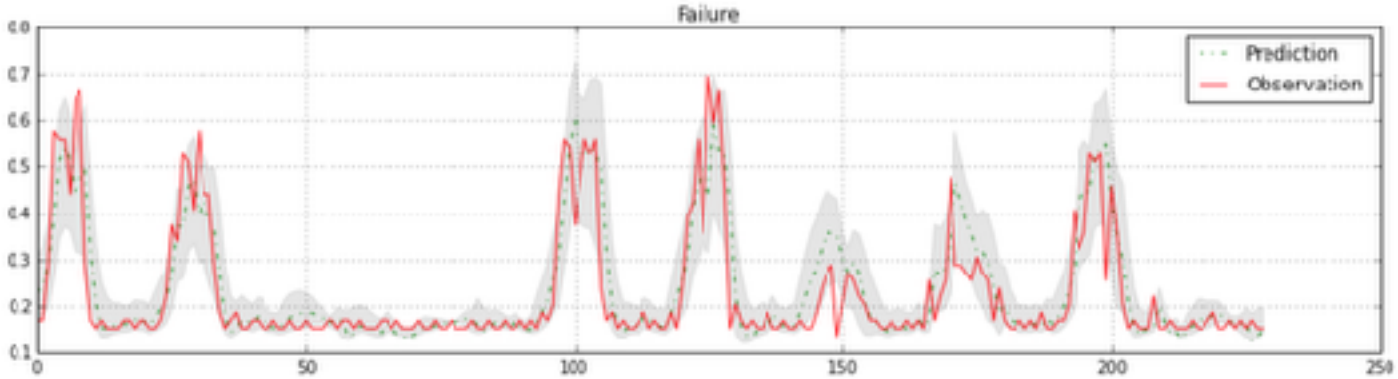


Marvis: Virtual Network Assistant - Inference Engine

- Based On:
  - Probabilistic programming
  - Bayesian statistics
- Feeds on preprocessed streaming data from:
  - Apache Storm
  - Live-aggregators

Anomaly Detection

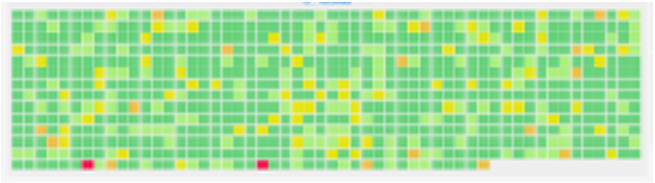
- Consumes preprocessed-aggregated data
- Intelligent spatio-temporal monitoring
- Analyzing network data across multiple dimensions
- Seasonal ARIMA and Tensor Flow models



What to expect?

- Minimize cost using the volatile AWS Spot instances
- Ensuring reliability amidst unpredictable server terminations
- Container right-sizing for reliability and autoscaling
- How to monitor real-time applications under chaotic conditions

Heat Map for CPU Load (Hotspots)



AWS EC2 Contract Types

- On-demand
- Reserved
- Spot

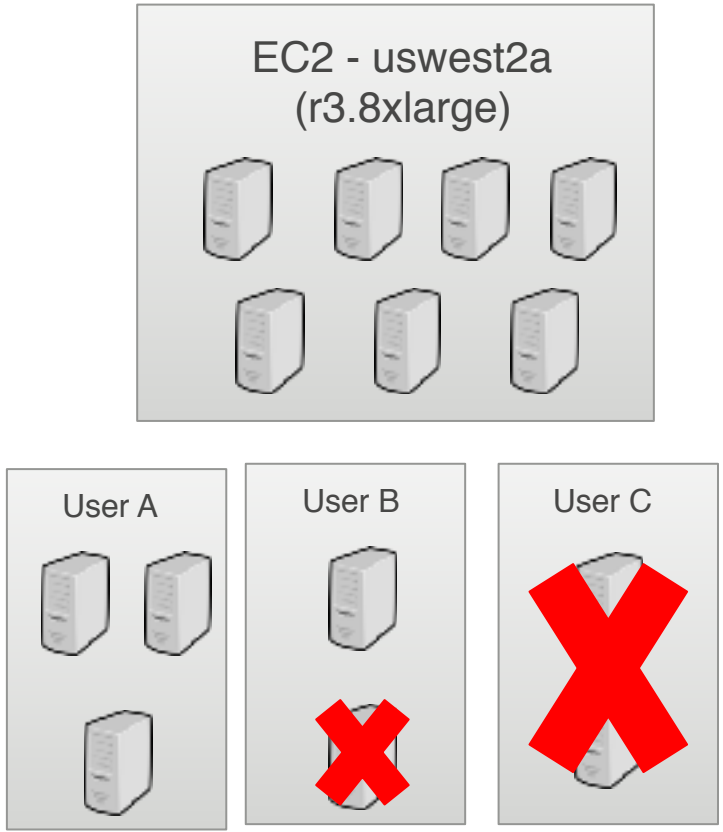
Contract Type	Cost	Reliable	Commitment Required?
On-demand	Very High	Yes	No
Reserved	High	Yes	Yes
Spot	Low	No	No

Spot Instances

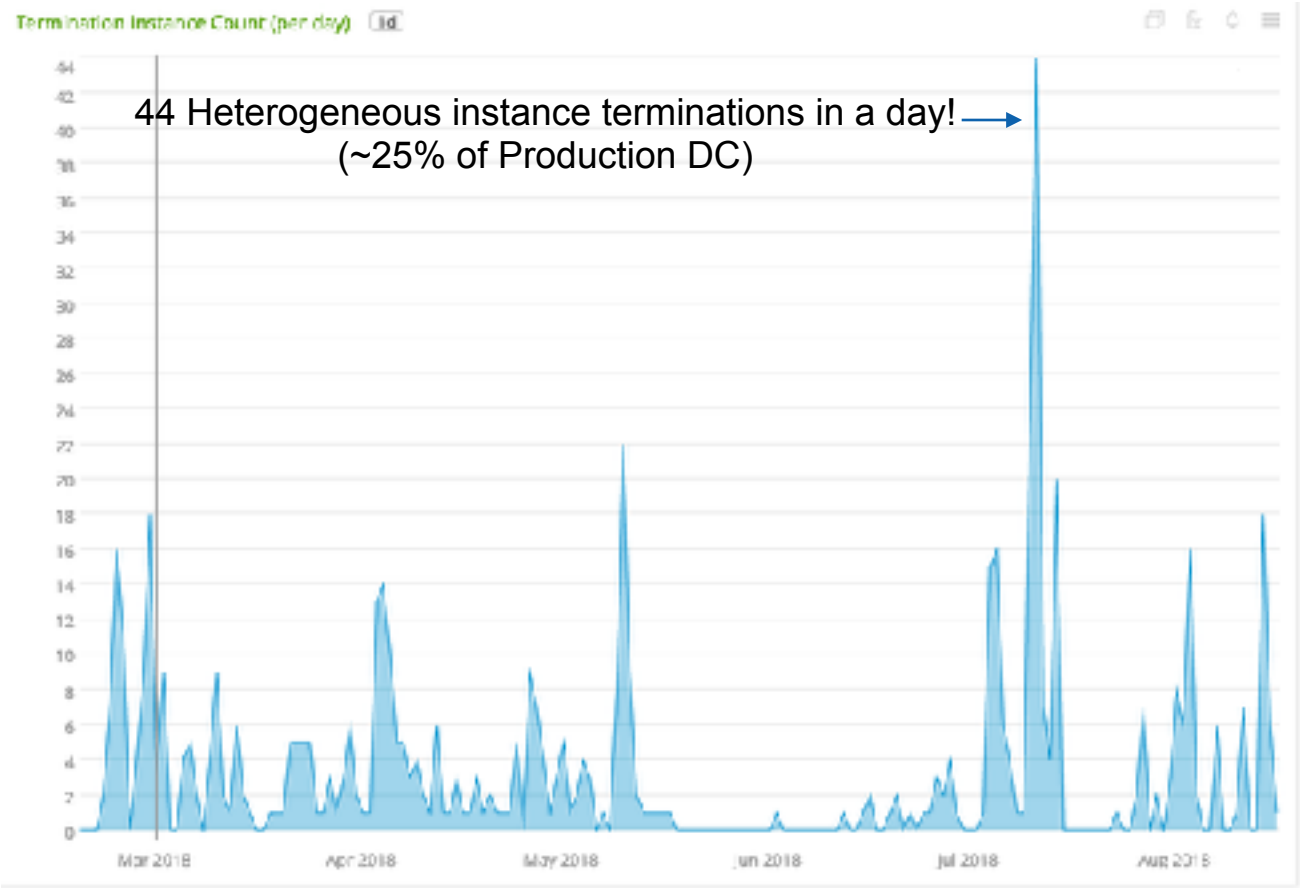
- Unused AWS EC2 capacity
- Bidding based
- If used correctly, they can be ~ 80% cheaper
- Be careful! Naive usage may end up costing more than on-demand

How does Spot Pricing Work?

- Three users make the requests
  - A: 3 machines bidding @ $10
  - B: 2 machines bidding @5
  - C: 4 machines bidding @0.4
- User ‘A’ gets 3 machines
- User ‘B’ gets 2 machines
- User ‘C’ gets *just* 2 machines
- Spot price is the bid price of last fulfilled request, i.e., $0.4 (every pays $0.4)
- On-demand demand increases by 3:
  - 3 spot machines terminated
  - New spot price: $5

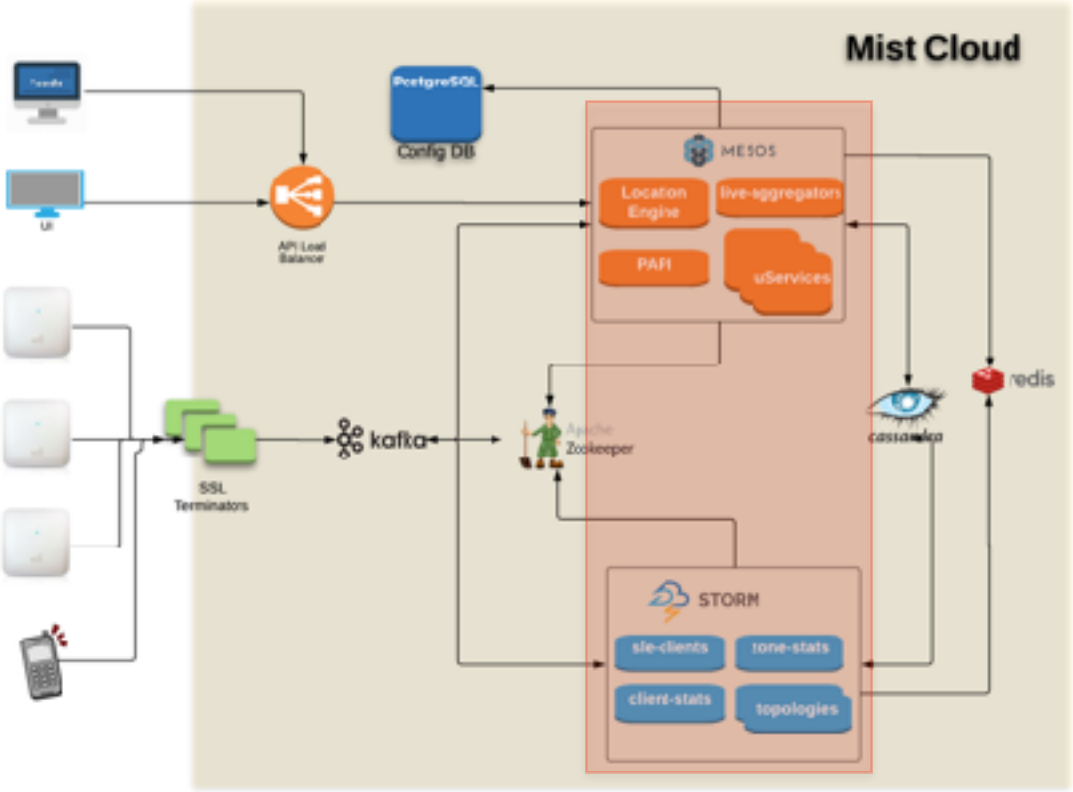


Spot Market Volatility



Mist Architecture Running On Spot

- 100% on Spot
- 80% of Datacenter
- Huge savings $$$
- Allows us to do more
- Forces reliability



Cost Benefits of Using Spot Instances

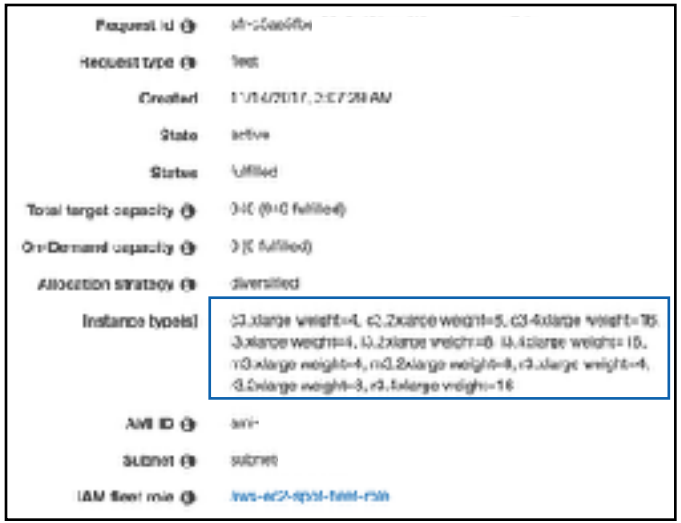
System	vCPUs	Percentage of total EC2 cost	Savings compared to On-demad	Savings compared to RI
Storm	1096 (31%)	17%	74%	60%
Mesos	1864 (52%)	30%	75%	62%
Total Cores	3590		56%	35%

17% non-spot infra constitutes 53% of our cost!

*Includes EBS cost as well

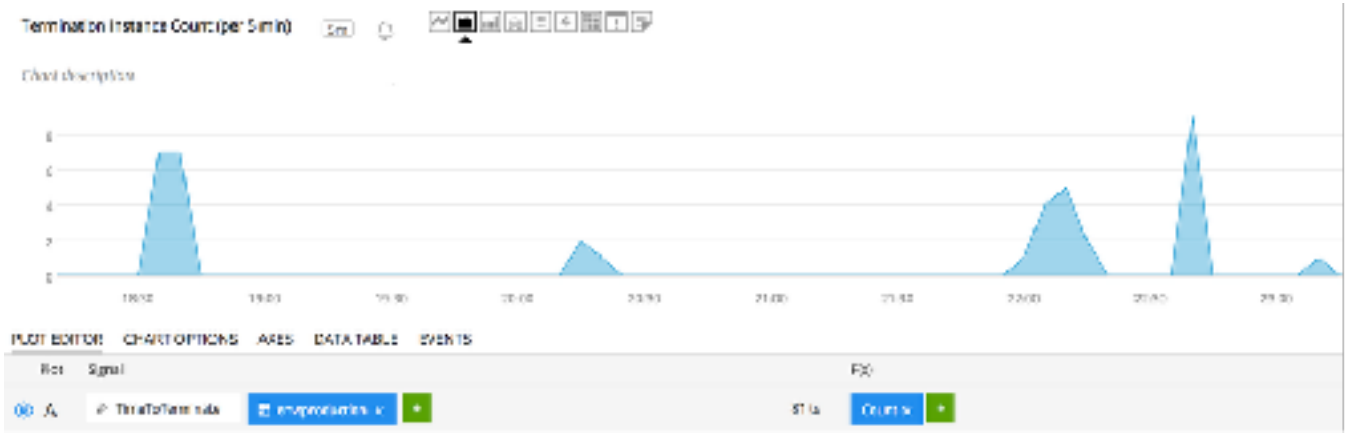
How to Use Spot Instances?

- Spot market = (instance type, AZ)
- Prices for difference spot markets are independent
- AWS Spot Fleet
  - Uses spot instances
  - Ensures capacity
- Caution: Cost saving scheme of AWS Spot Fleet is risky!
- Diversify across markets!
- Applications can restart



How Much To Overprovision?

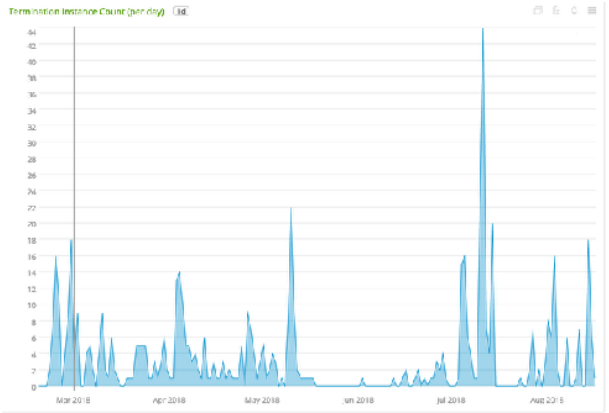
- Lead time for spot instance: < 5 mins
- Max number of instances terminated in 5 mins = 9
- Overprovision by 9 * 3 = 27 (where 3 is a magic multiple)
- The bigger the cluster the better it is



Unreliable Servers Help Building Reliable Systems!?

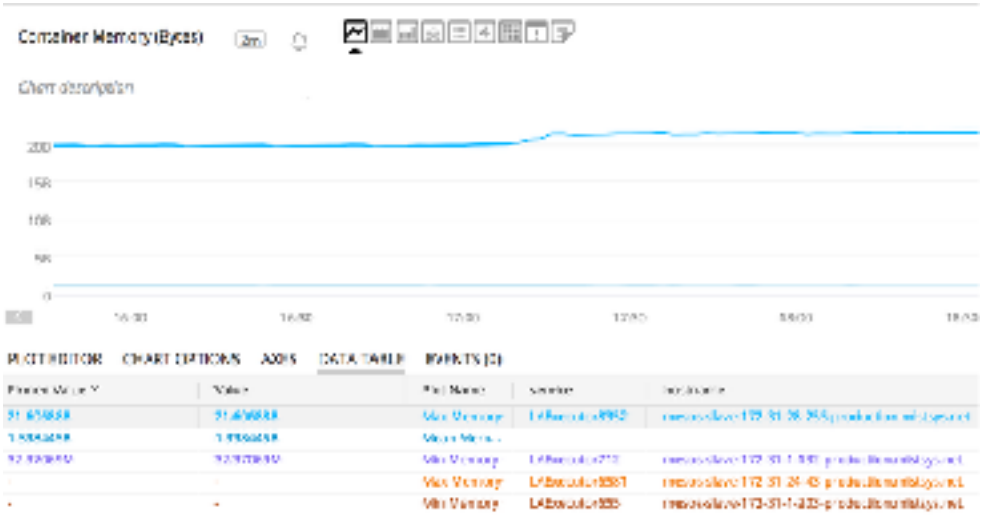
- Chaos Monkey
  - Pioneer of chaos engineering
  - Controlled Chaos (can be stopped and resumed)

- Use of Spot Instances
  - Uncontrolled `Real' Chaos
  - Can't be turned off and depends on spot prices
  - Enables building reliable infrastructure. Wait what?
  - Can be volatile



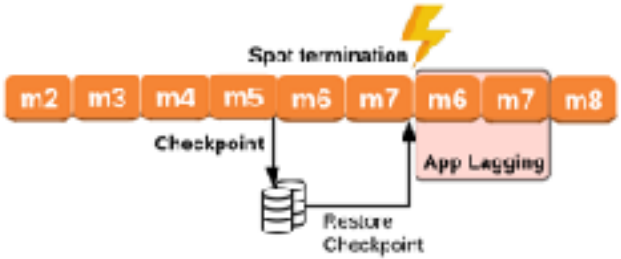
Coping with Spot's Unreliability

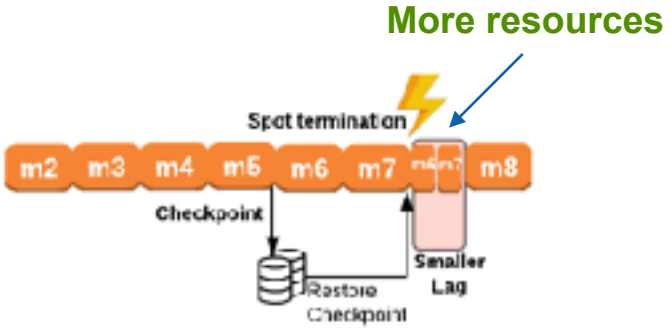
- Mist Systems relies on both stateful and stateless applications
- Live-aggregators:
  - 1.2+ TB in-memory state
  - ~ 1000 Mesos containers
  - CPU Utilization 60%-80%
  - Per-container memory footprint
    - minimum: 32MB
    - maximum: 21GB
    - mean: 1.3GB



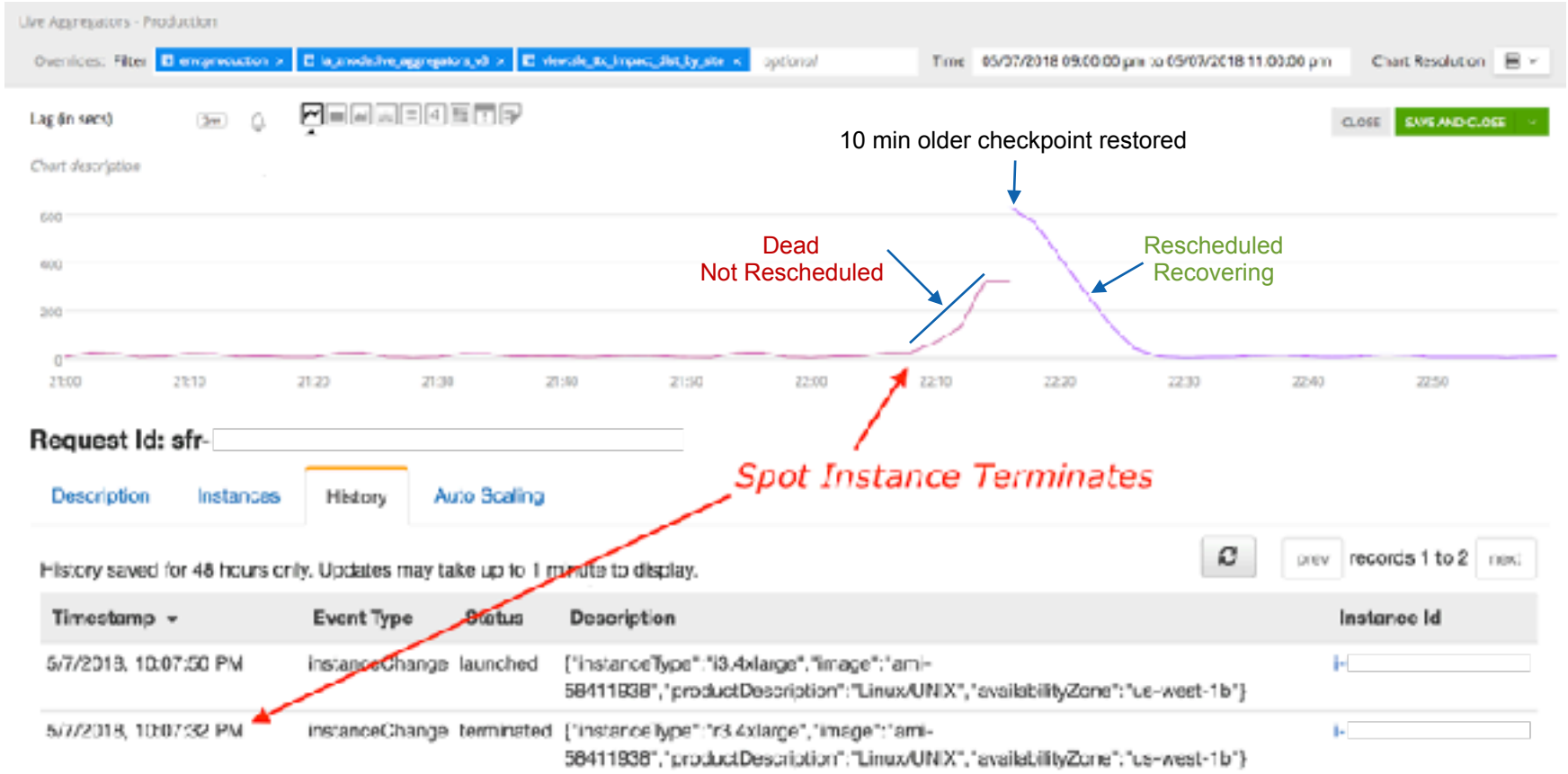
Coping with Spot's Unreliability: Checkpointing

- How far a realtime application is lagging?
- Checkpointing is critical!
- Each platform we use has a mechanism for checkpointing:
  - Storm: Redis
  - Mesos:
    - Live-aggregators: S3
    - Real-time: Redis
    - API: Stateless



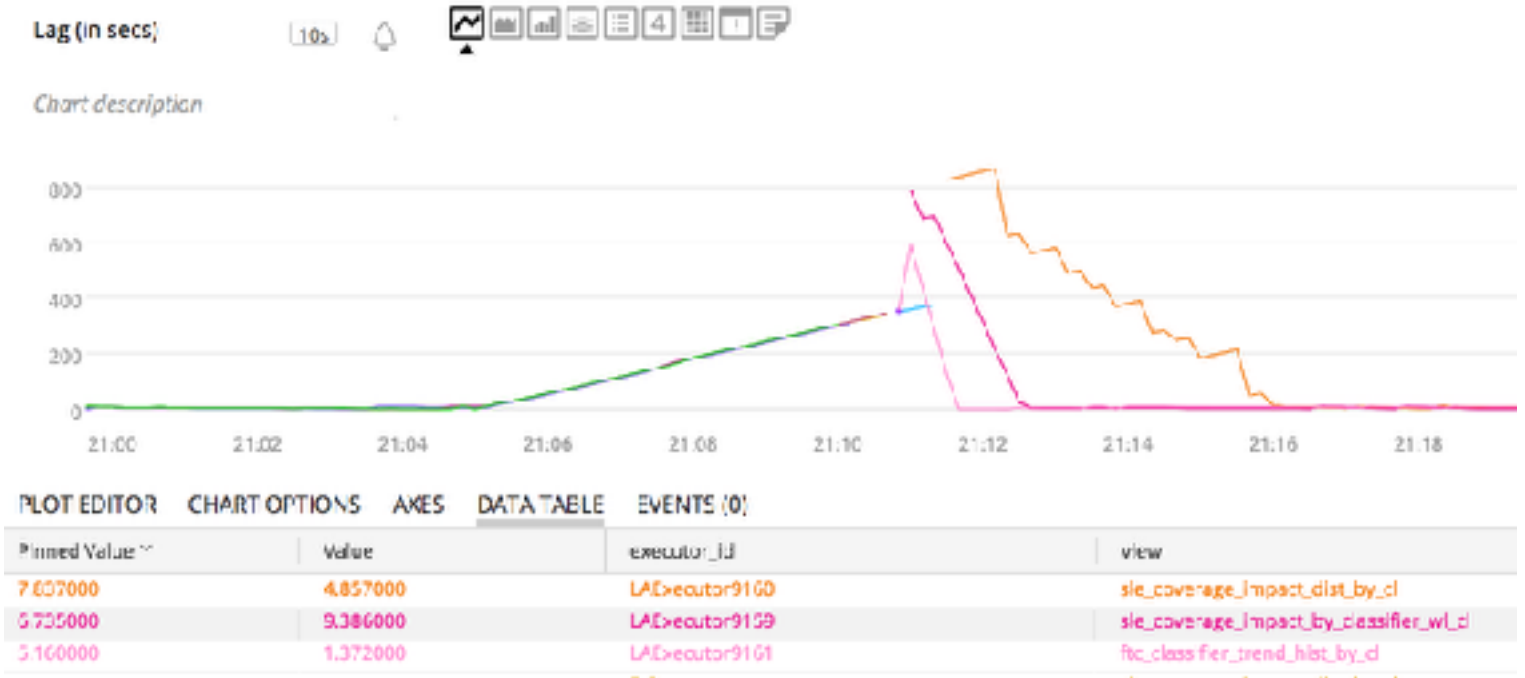


Spot Termination: Lag and then Recovery



Difference in Recovery Times

- High Message Rate
- Type of computation
- Amount of data/msg



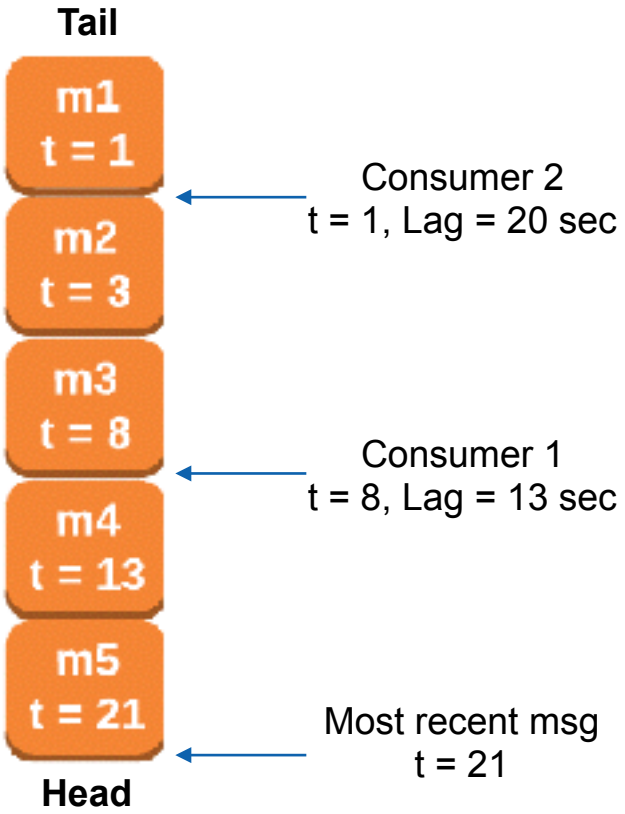
Why Do We Monitor?

- Detecting the problem: Is a real time application falling behind?
- Troubleshooting:
  - Resource bottlenecks
  - Bottlenecked on an external system
  - Application code errors
- Use SignalFX, Graphana, Graphite and Cloudwatch

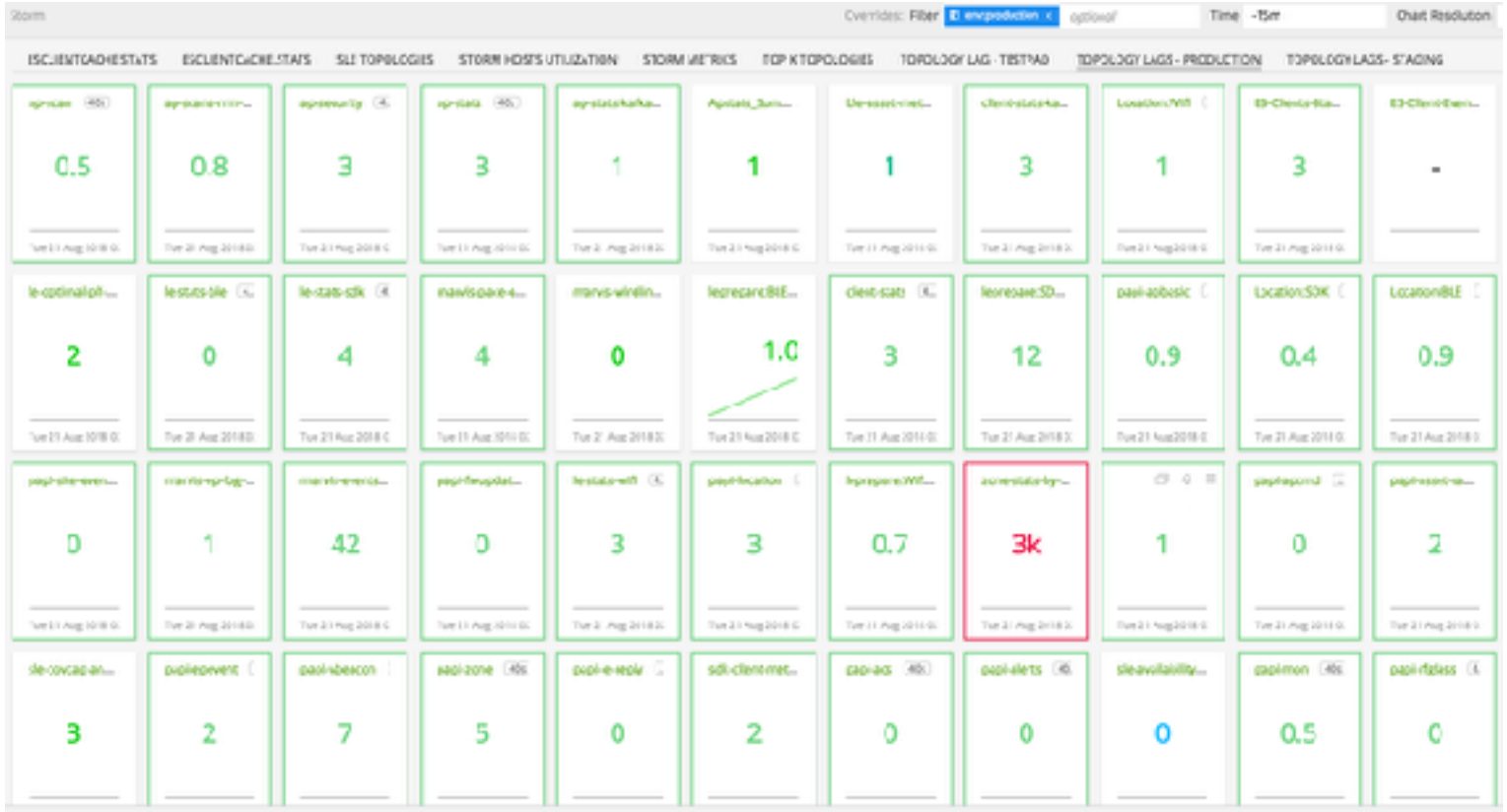
SignalFx

Detection: Realtime Lag

- How far behind is a realtime service/ topology?
- Difference between Kafka timestamp between:
  - Most recent message
  - Last consumed message for an application
- How much lag is acceptable? nano secs, millisecs, secs?



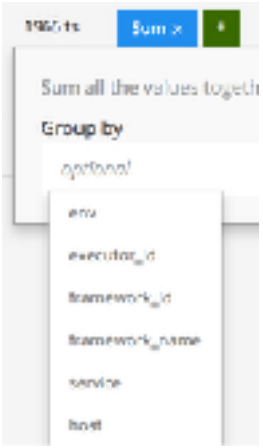
Realtime Lags



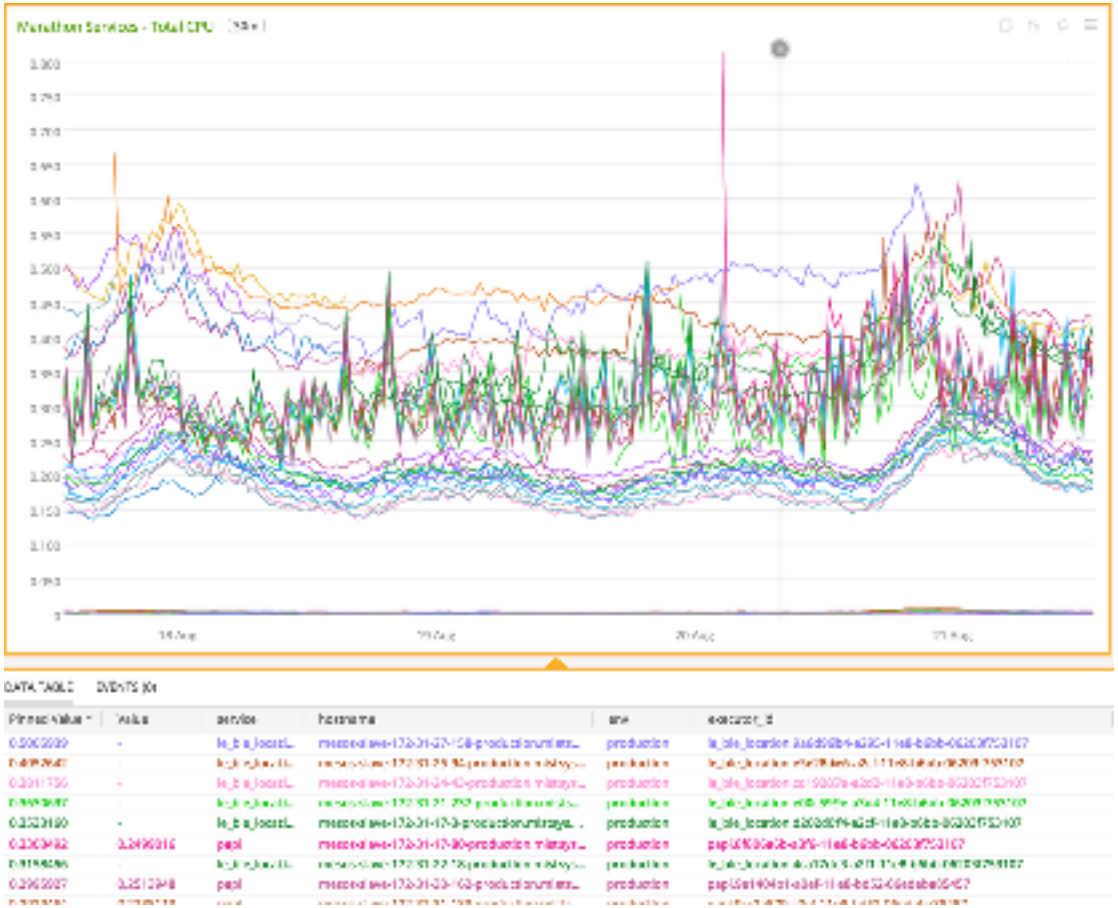
Attribution: Resource Monitoring

- Track key metrics per container/ worker
  - CPU, Memory, Network IO
  - CPU Kafka stream
  - Timing individual code blocks
- Write metrics using different dimensions to filter and aggregate
  - Host name
  - Container ID
  - Service/Application name
  - Environment
  - Instance Type

signal fx



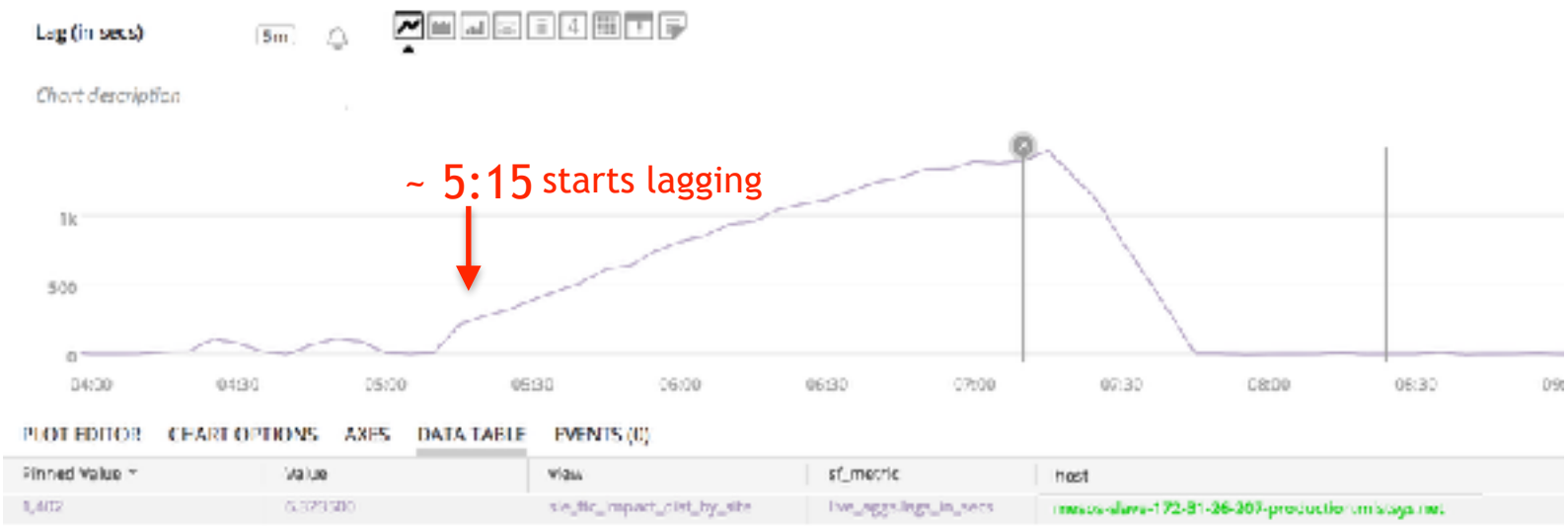
CPU Metrics for Microservices

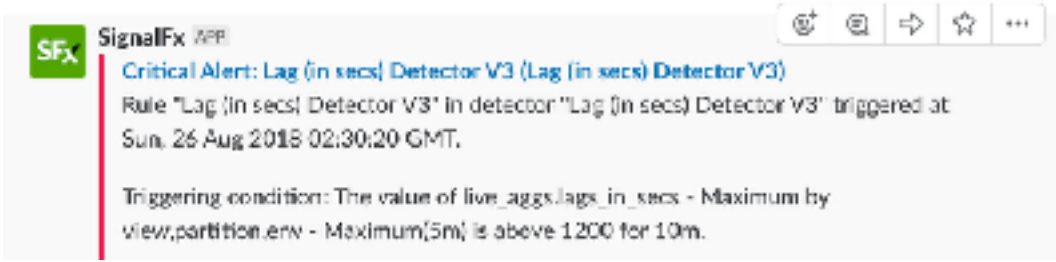


Performance Variations Across Instance Types

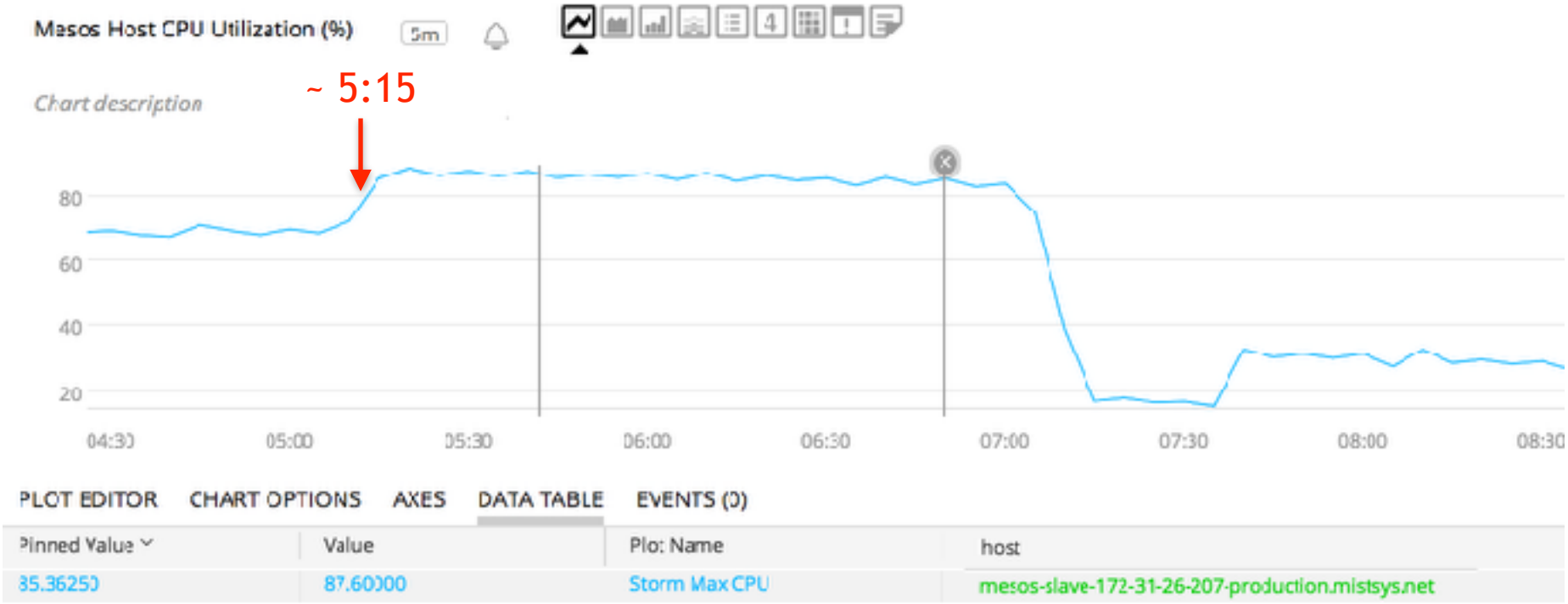
CPU utilization (%) for consuming the same stream

Case Study: Lagging Application

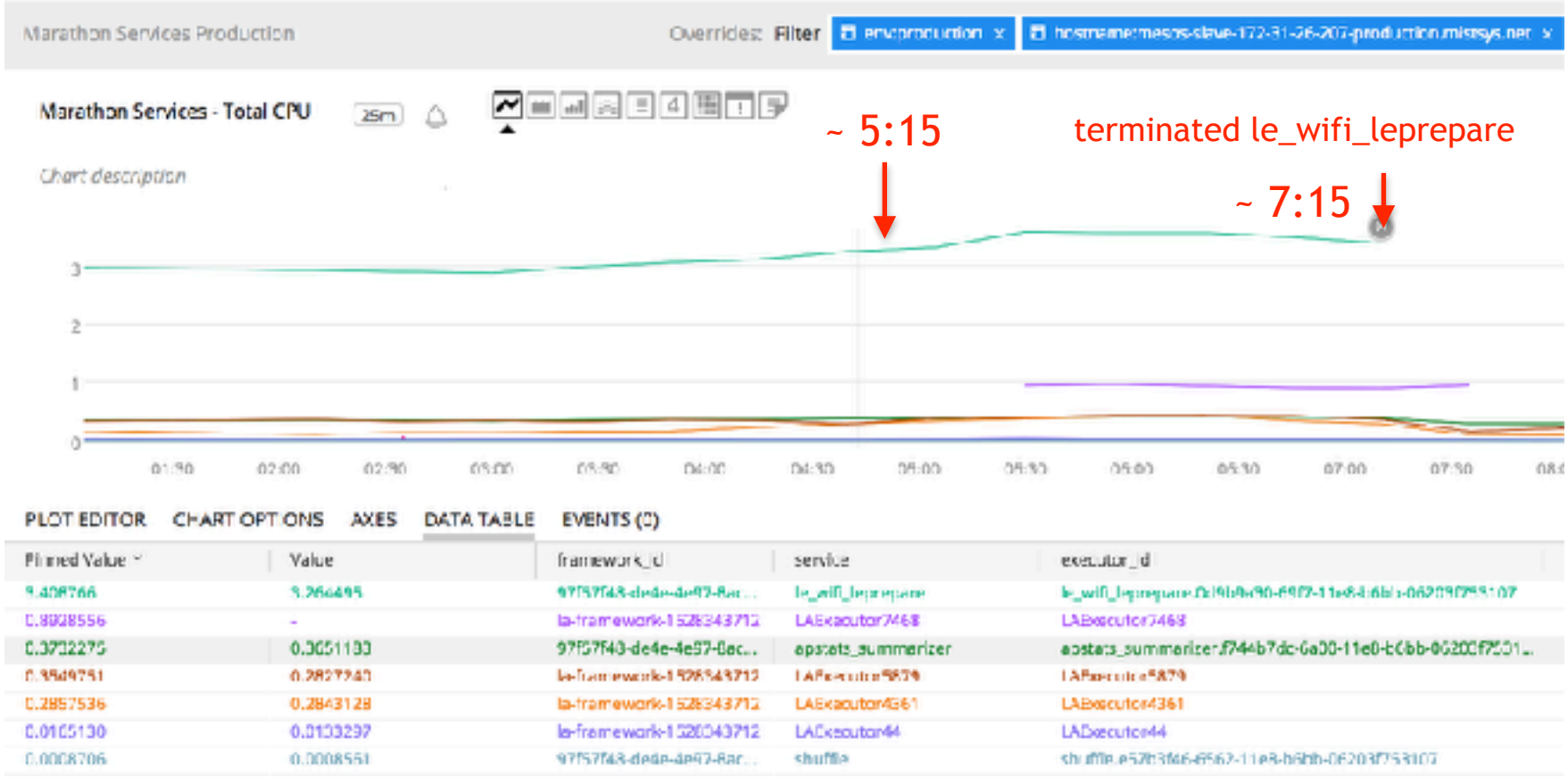




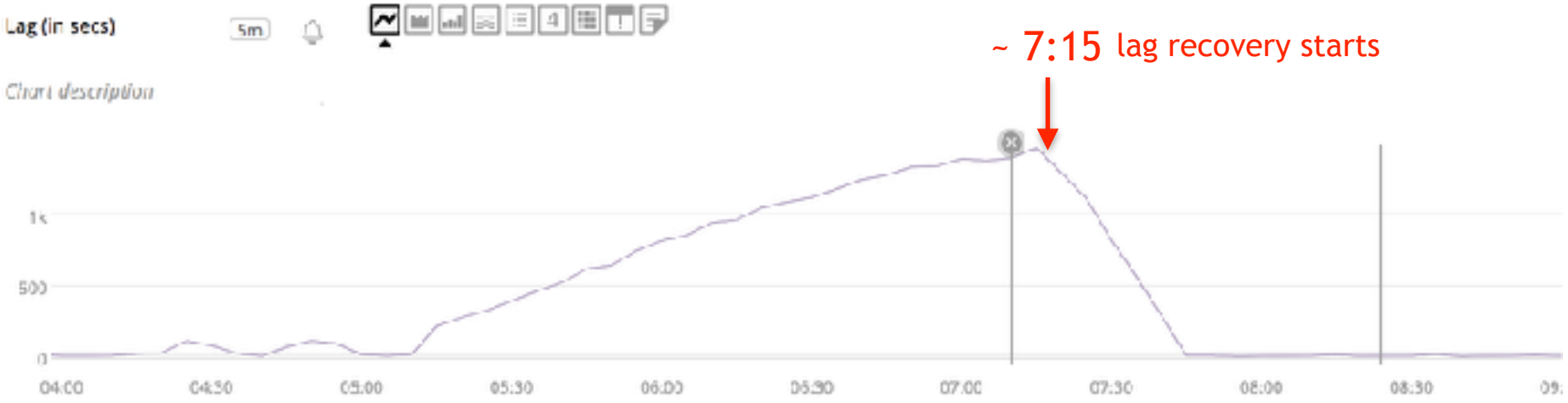
Case Study: High CPU Utilization



Case Study: Host CPU Utilization Details



Case Study: Lag Recovers



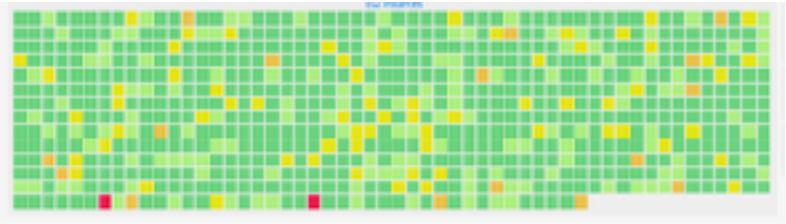
Lying Factor

- Predicting resource requirements is difficult
- How much CPU, memory and network IO?
- Recent resource managers are typically book keepers:
  - Mesos
  - Kubernetes
- Lying factor is the difference between (1 - 2 = -1 cores)
  - Reserved resources (1 cores)
  - Actual used resources (2 cores)

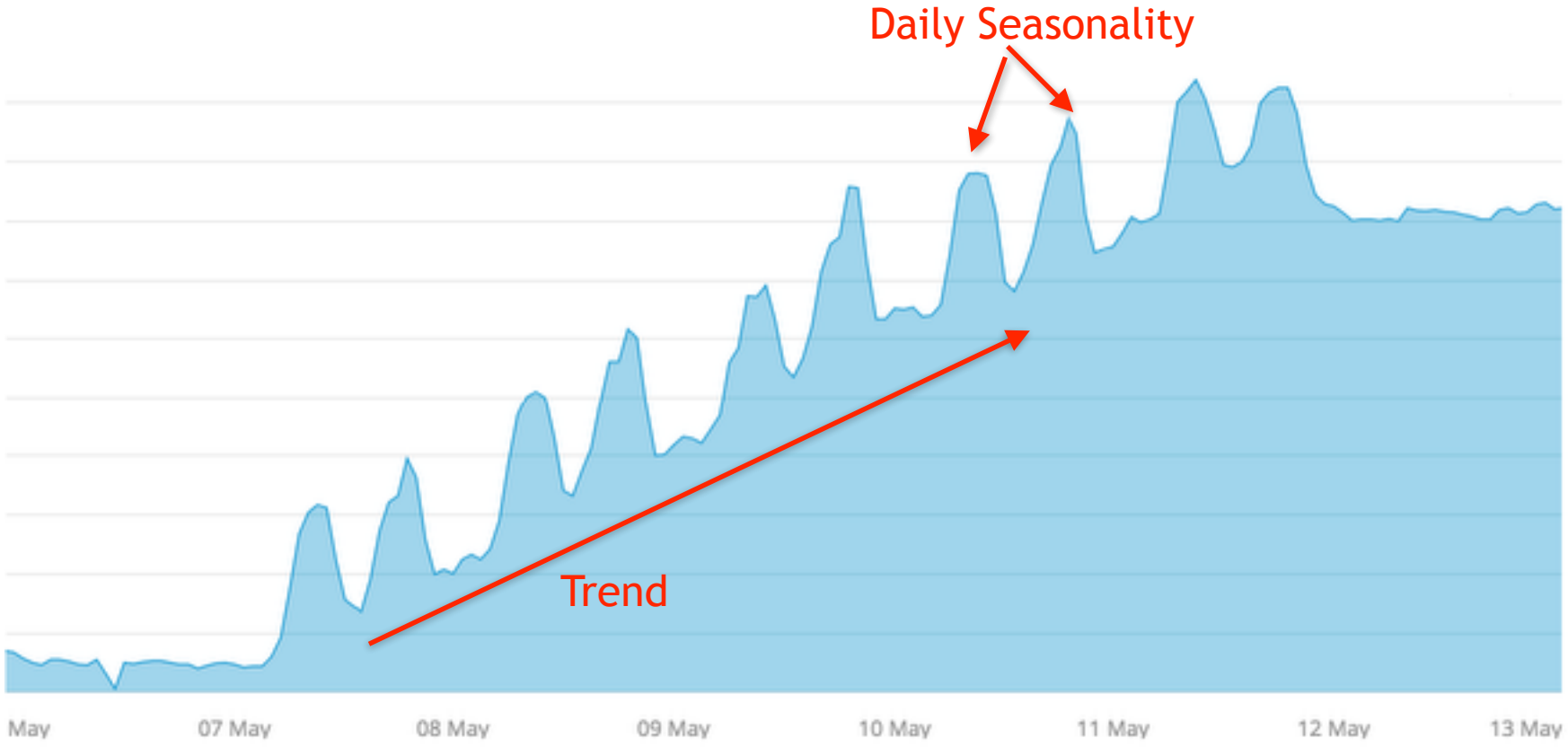
Noisy Neighbor

Right-sizing Containers

- Lying Factor for a container:
  - Negative: Dangerous! Using more resources than it reserved (hotspots)
  - Positive: Resources wasted
- Ideally should be 0 (reserved = actual/consumed)
- Manually update resources using marathon

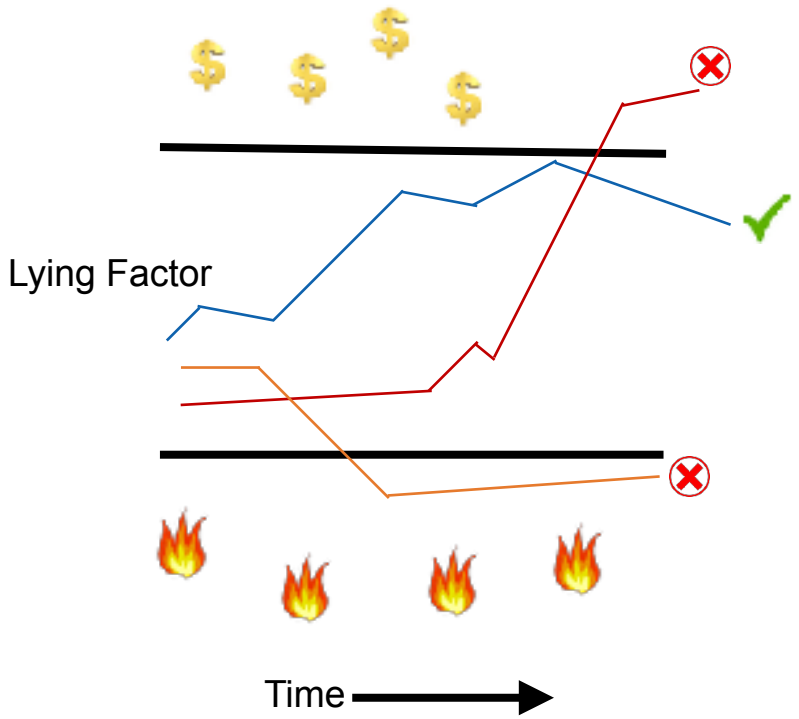


Dynamic Load ($\propto$ Incoming data stream)

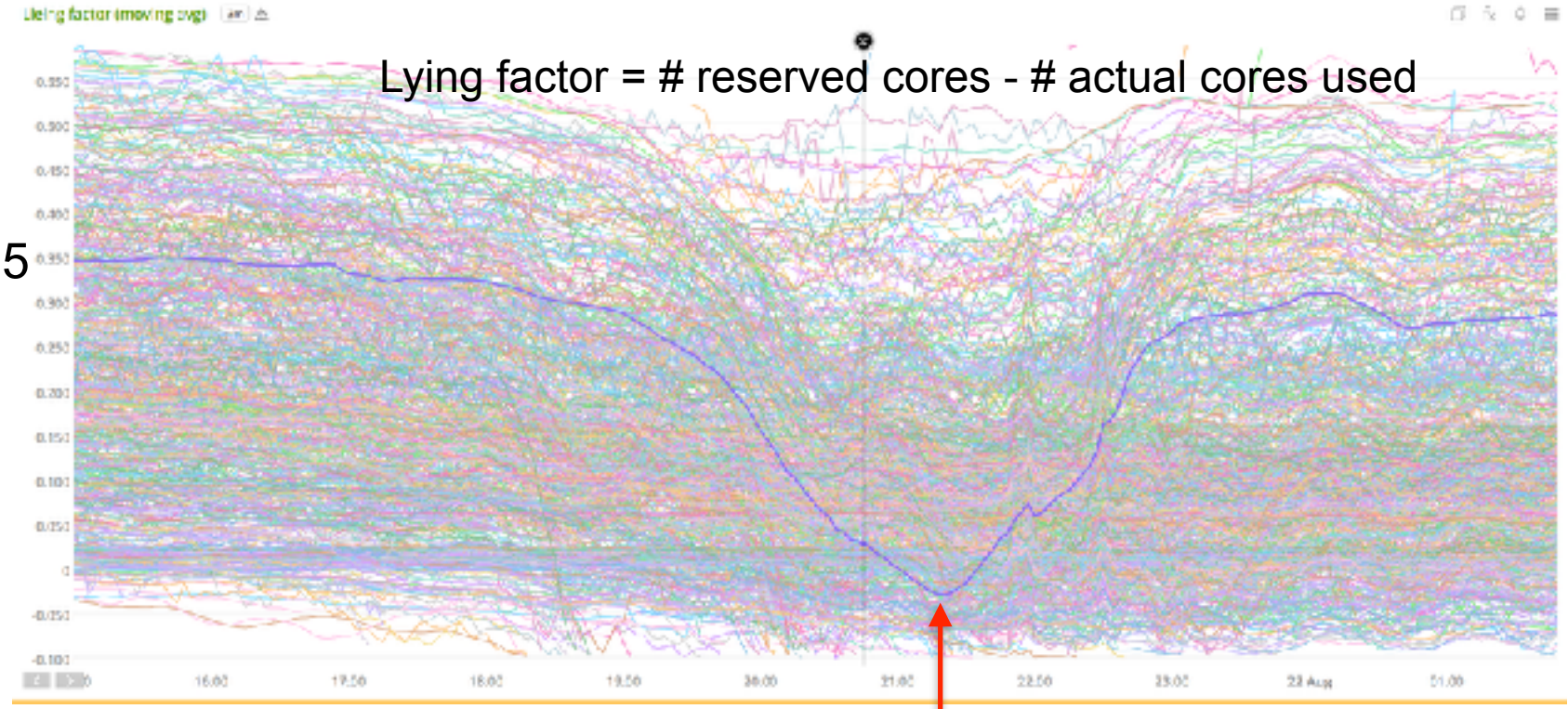


Autoscaling Using Lying Factor

- Changing load common in data streams
- Complicated due to changing load:
  - Seasonality (daily peaks)
  - Trend (load increasing overtime)
- Autoscaling based on lying factor for containers
  - Reduce resources when Lying factor crosses a positive threshold
  - Increase resources when Lying factor below a number
- Maintain Lying factor within a band



Lying Factor vs Load



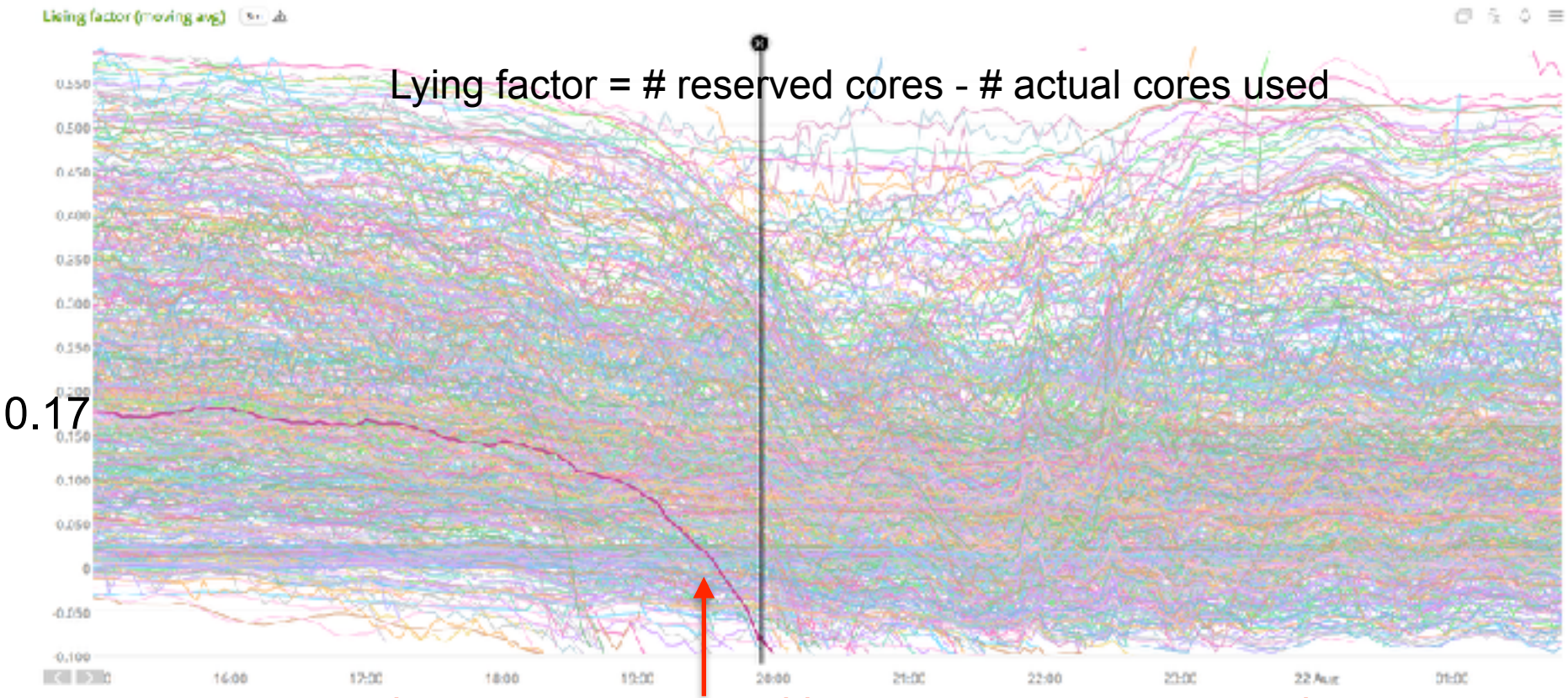
0.35

Upper threshold: 0.6

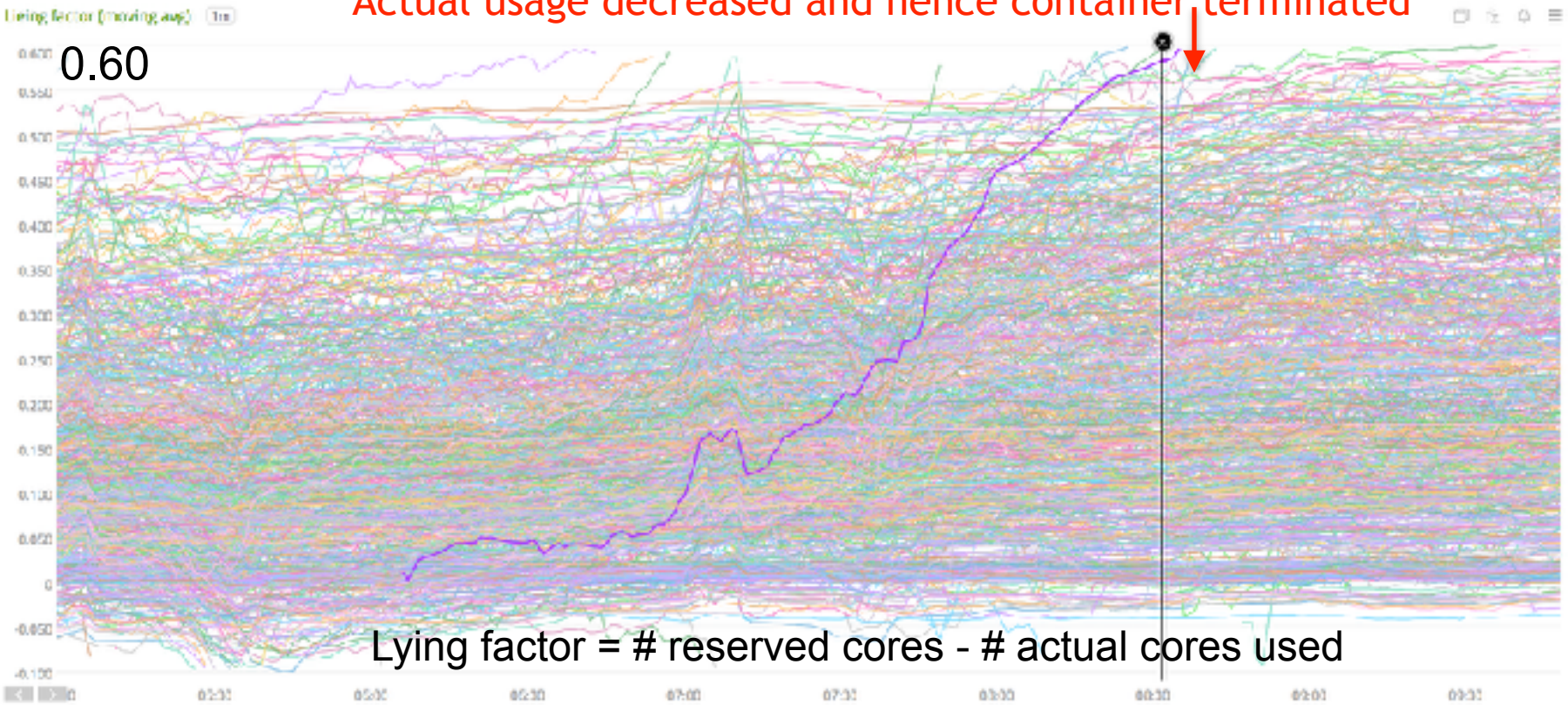
Peak Load

Upper threshold: -0.1

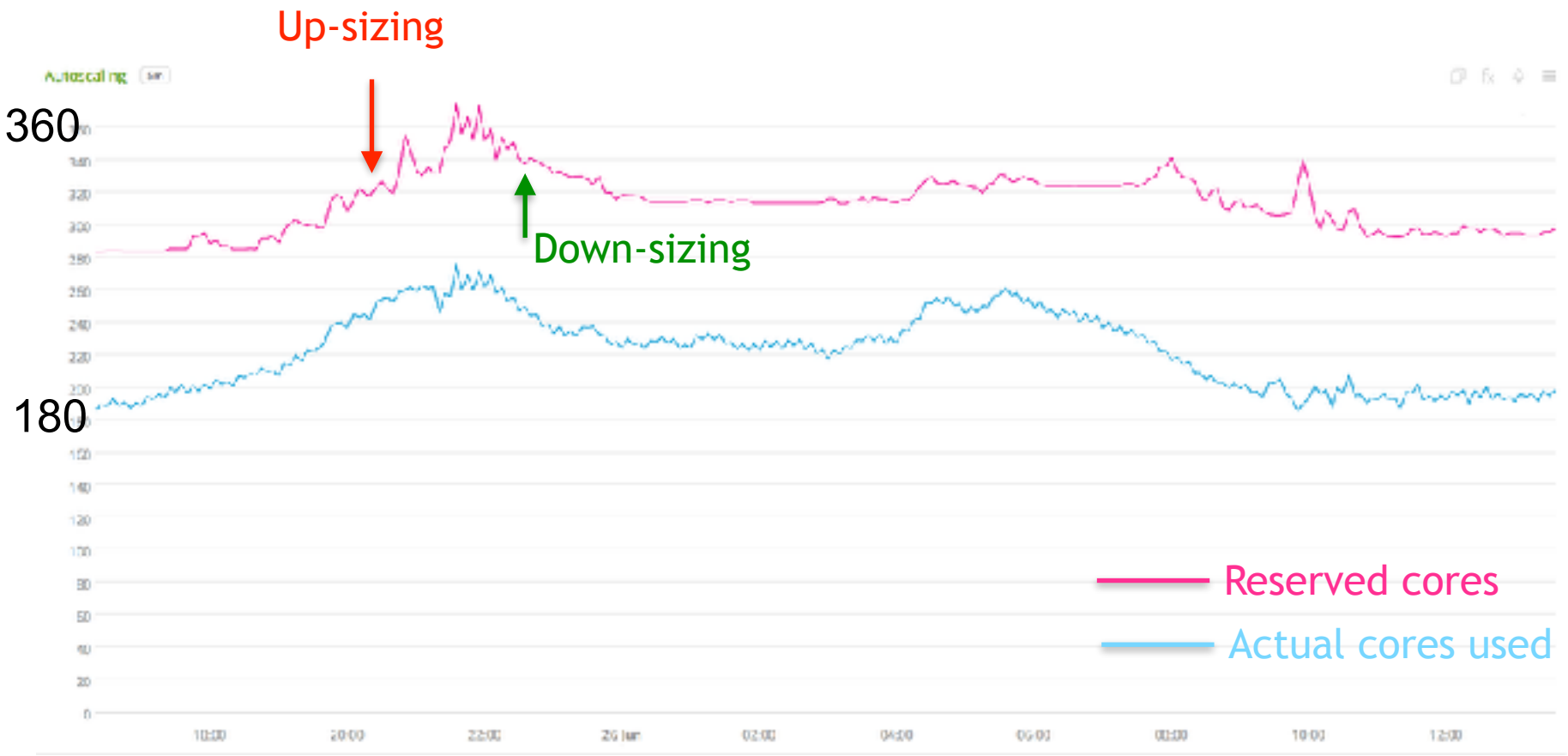
Containers Up-sized Automatically



Container Down-sized Automatically



Autoscaling in Action: Reserved vs Actual Usage



What next?

- Using Spot instances for:
  - Kafka
  - Cassandra
- Intelligently and automatically selecting Spot instance types:
  - Least cost
  - Least volatile
- Autoscaling the remaining applications

Hiring

- Site Reliability Engineer
  - Please email: osman@mist.com
  - Contact me on LinkedIn (Osman Sarood)
  - Fresh position hence not on the website yet 
- Data Scientist
  - Please email: osman@mist.com
  - Contact me on LinkedIn (Osman Sarood)
  - On the website (www.mist.com/careers)

Questions


Mist
AI is in the AIR™