Feedback Computing

Challenges and Opportunities in Cloud Architectures

Dr. P. Simon Tuffs
Life360.com

10’th International Workshop
Feedback Computing
Seattle, April 13 2015

Abstract

As computing has moved to the cloud, the tradeoff between performance, availability and cost has become increasingly complicated. Distributed cloud based architectures are complex, nonlinear, stochastic and time-varying, and are often inherently unstable under heavy load. Companies that move to the cloud need to maximize performance subject to constraints on availability and cost. This paper surveys some of the challenges involved in designing and operating micro-service cloud systems, and explores constraints to their stability and efficiency. These constraints will be illustrated with examples drawn from Netflix and Life360.

Bio

Dr. Simon Tuffs began his career in computing by researching self-tuning control systems at the University of Oxford. He then moved into the software engineering industry and has held a wide range of positions developing and deploying commercial systems. His current role is leveraging analytics and feedback to high-performance location services in the cloud.

Aviation History

1910:

performance (*)

cost (*)

stability (-)

source: http://upload.wikimedia.org/wikipedia/en/f/f5/Howard_Wright_Biplane_(1910).jpg

Aviation in 1940:

1940:

performance (***)

cost (**)

stability (-)

http://upload.wikimedia.org/wikipedia/commons/thumb/3/31/Ray_Flying_Legends_2005-1.jpg/600px-Ray_Flying_Legends_2005-1.jpg

Aviation Today:

2010:

performance (*****)

cost (*****)

stability (-----)

(coordinated feedback is critical)

http://upload.wikimedia.org/wikipedia/commons/d/da/Blue_Angels_Flying_in_Delta_Formation_at_Miramar.jpg

The Blue Angels Don’t Automate...

because…

sometimes they

do this!

http://upload.wikimedia.org/wikipedia/commons/d/da/Blue_Angels_Flying_in_Delta_Formation_at_Miramar.jpg

A Relevant History

- Avionics
  - 1910: Wright Biplane
    - Many people required to launch, one to fly, unstable but forgiving
  - 1940: Spitfire
    - Performance increase, fewer to launch, one to fly, unstable
  - 2010: Blue Angels
    - Performance at scale, many many to launch, many to fly, very unstable.

Application Computing in 1994

1994:

performance (*)

cost (*)

availability (*)

http://www.augustinas.net/Nuotraukos/10%20Kompiuteris.jpg

Application Computing in 2004

Sun E-10000:

performance (***)

cost (*****)

availability (****)

http://www.slac.stanford.edu/comp/unix/farm/E10000/about.html

Cloud Computing Today

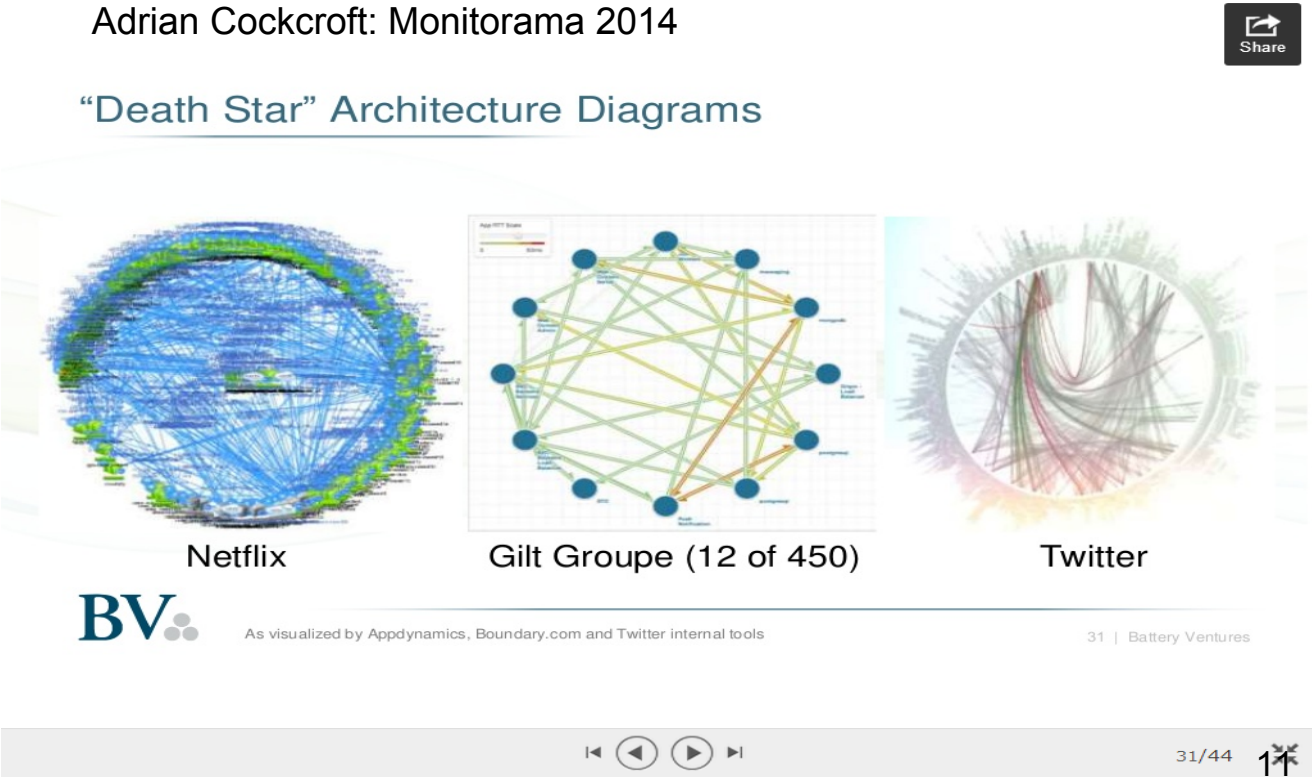
Cloud:

performance (*****)

cost (***)

availability (?)

Adrian Cockcroft: Monitorama 2014



In Computing…

- Evolution, at a compressed timescale:
  - 1990: IBM-PC, one person to launch and maintain, stable (ish)
  - 2000: Sun E10000, a few people to launch an maintain, serves many, does not scale, stable.
  - 2010: The Cloud: many people to launch and maintain, scales, unstable.

Challenges In Cloud Computing

- Efficiency:
  - Demand (Performance)
  - Capacity (Cost)
  - Availability
- Availability:
  - Architecture
  - Deployment
  - Operations

Micro-Services and Efficiency in the Cloud

scaling, feedback, feedforward
operating efficiently

Micro-services Architectures

- A micro-service is a computational entity which implements a REST API in a scalable way
  - ideally a few hundred lines of clean business logic
- A cloud computing infrastructure of cooperating micro-services connected in a dependency graph
  - dynamic, ephemeral instances
  - distributed, networked dependencies
  - distributed, resilient persistence
  - low latency request/response (<<50ms), or async

Micro-services Architectures

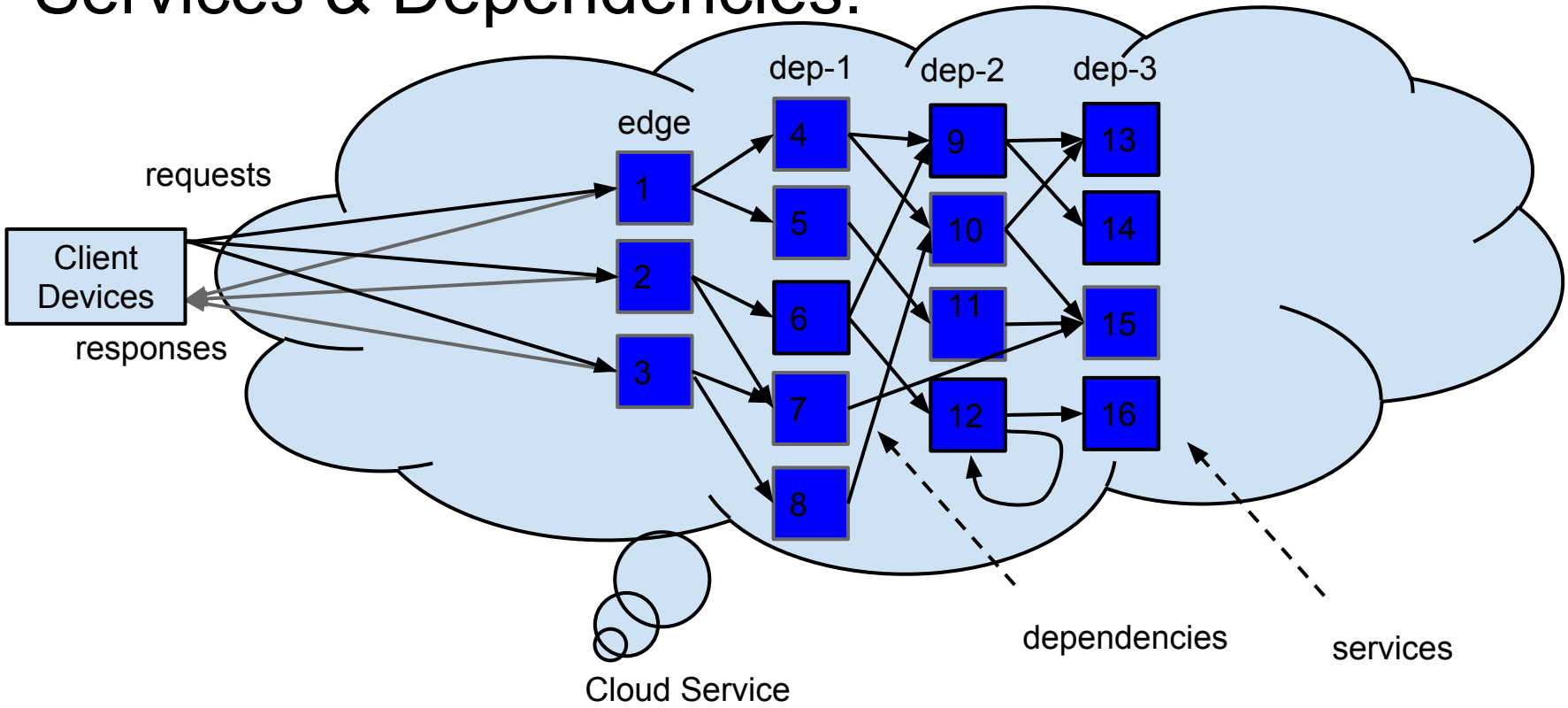
- contd...
  - wide, not deep
  - scalable up and out
  - resilient to failure
  - behave like dynamic systems from the controls domain
    - lags, delays, gains, nonlinearities, multiple variables

Examples of Microservices

- A cloud system may contain >100 microservices
  - Authentication Service
  - Identity Service
  - Account Service
  - Device Management Service
  - Subscriber Service
  - Digital Rights Management Service
  - Discovery Service
  - Configuration Service
  - … etc.

Microservices Architecture

Services & Dependencies:



Microservices and Latency

- A major mode of failure under load is timeouts due to latency
  - service dependencies accrue latency
  - services timeout and send errors back to clients
  - retries by clients make things ***worse***

Timeouts: Who, Why, Where

- who: clients, services
- why: avoid getting stuck, avoid exhausting resources
- where: everywhere

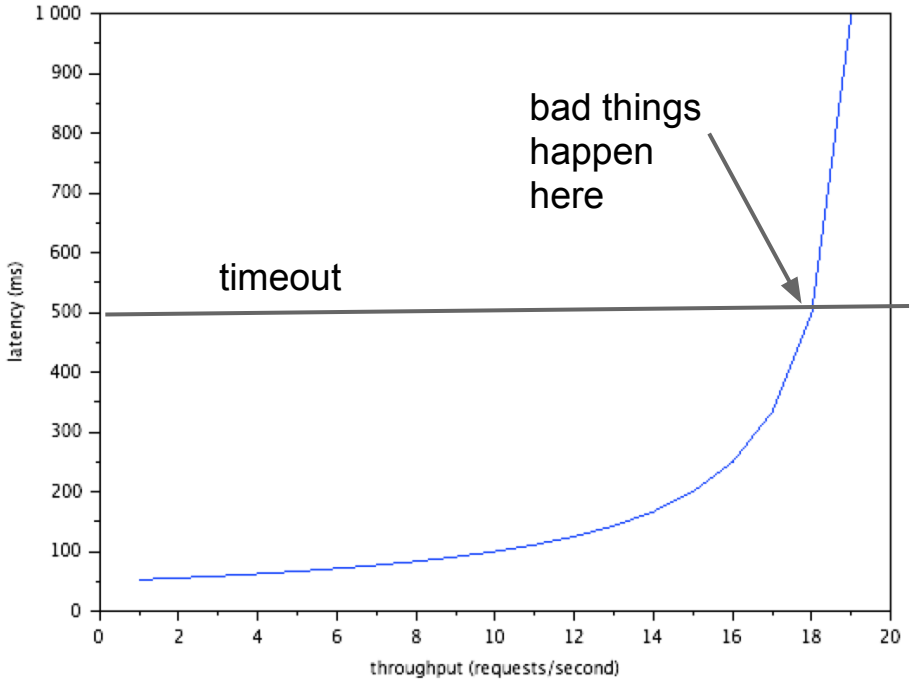
timeouts => errors

Latency Limits To Performance

- M/M/1: for a typical single server, with randomly arriving packets:
  - poisson arrival/exponential service/1cpu -- a very good model for the cloud
  - latency = 1/(C-L)
    - C = capacity (requests/second)
    - L = load (requests/second)
    - C -> L: latency -> ∞

source: http://en.wikipedia.org/wiki/M/M/1_queue

M/M/1 Load Latency



Request/Latency Curve

But it can be worse: more coordinated arrival times (scheduling)

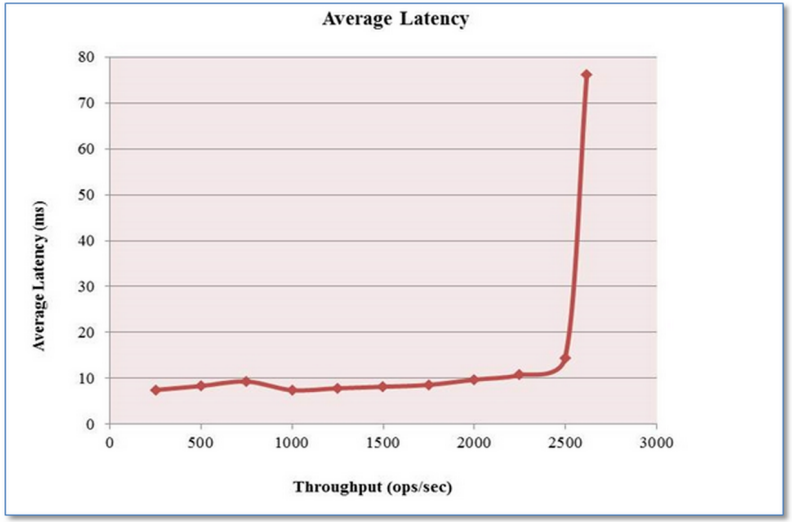


Figure 3: Throughput vs Average Latency

source: http://blog.flux7.com/blogs/benchmarks/littles-law

Operating at 50% capacity

10% transient increase in traffic: no failures

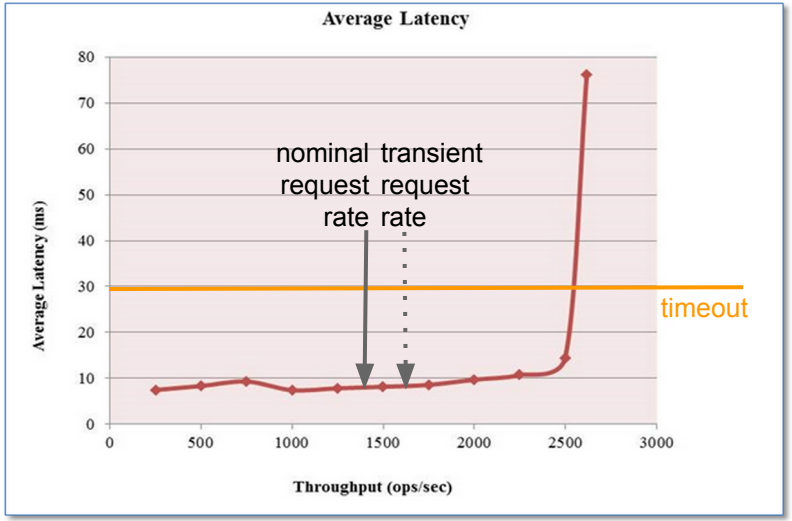
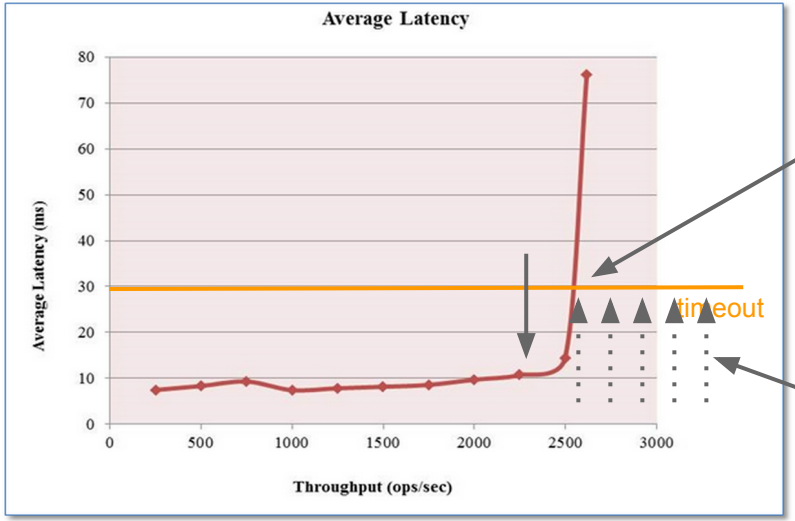


Figure 3: Throughput vs Average Latency

Operating At The Edge

10% transient increase in traffic



50% of all requests are now timing out

100% of all requests now timing out

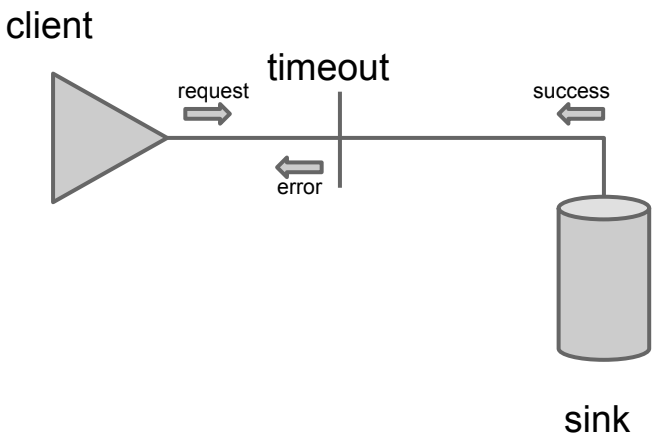
Figure 3: Throughput vs Average Latency

Feedback and Feedforward Systems

thundering herds/scaling up

Thundering Herd

Request/Response



- client sends events at some average random rate
- client waits for response indicating success
- errors cause retries at a constant rate for a given period of time
- timeouts turn latency into errors.

Thundering Herd

- Clients which retry on error result in an unstable system
  - due to positive feedback interacting with load/latency nonlinearities)

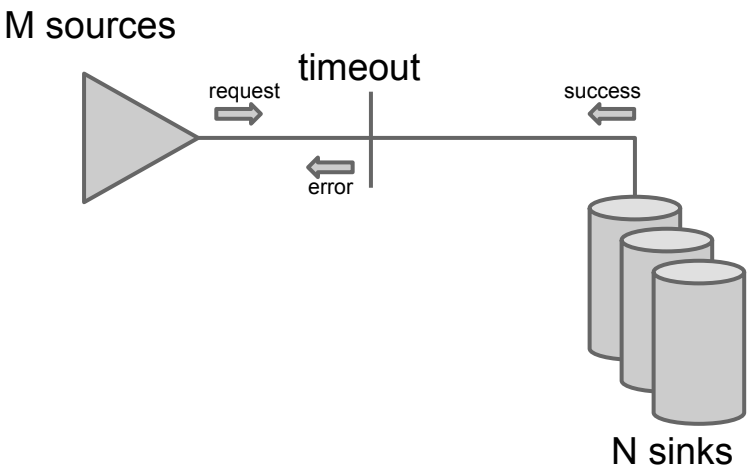
Thundering Herd

Consequences

- as server reaches request capacity, latency increases
- clients begin to timeout and retry
- this increases the request load, worsening latencies
- implicit positive feedback loop.
- open-loop unstable

Scaling Up Against the Herd

Autoscaling



- M >> N
- when average load on sinks exceeds threshold, add a new sink
- assume requests evenly distributed across sinks

Question: How well does this perform?

Scaling up Against the Herd

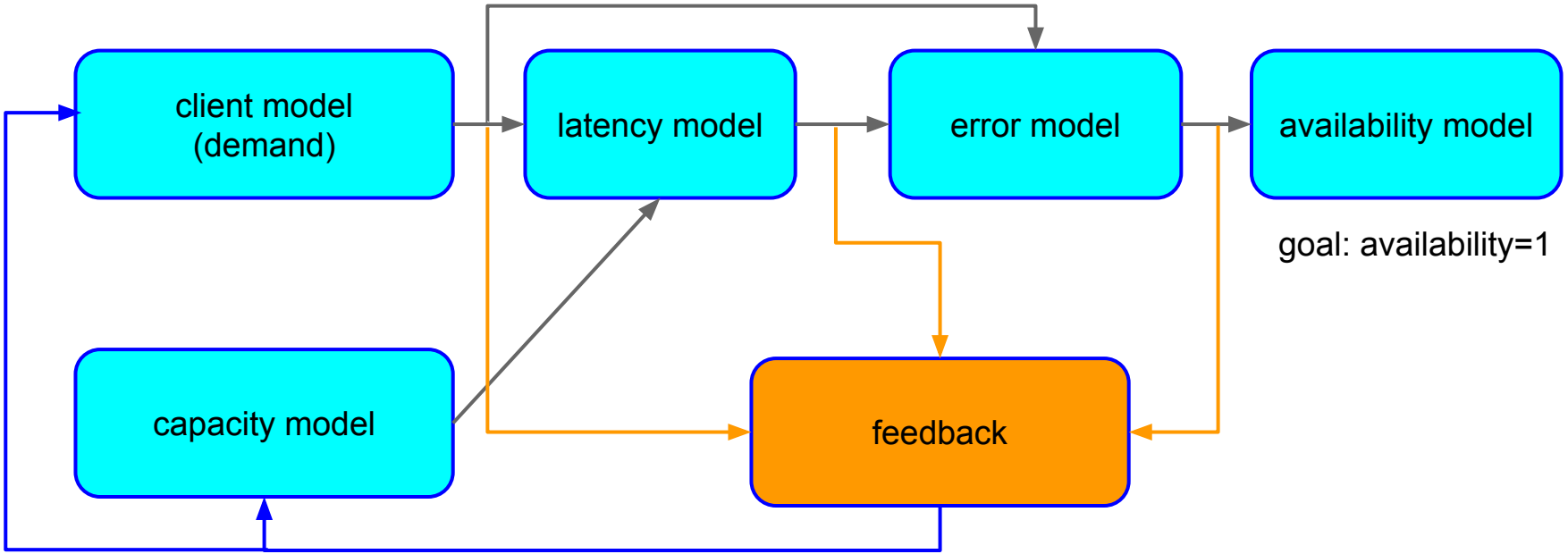
Consequences

- services stays away from overload, errors due to latency minimized
- services are consistently over-provisioned, this matters!
- delays in provisioning (start-times) break this model.

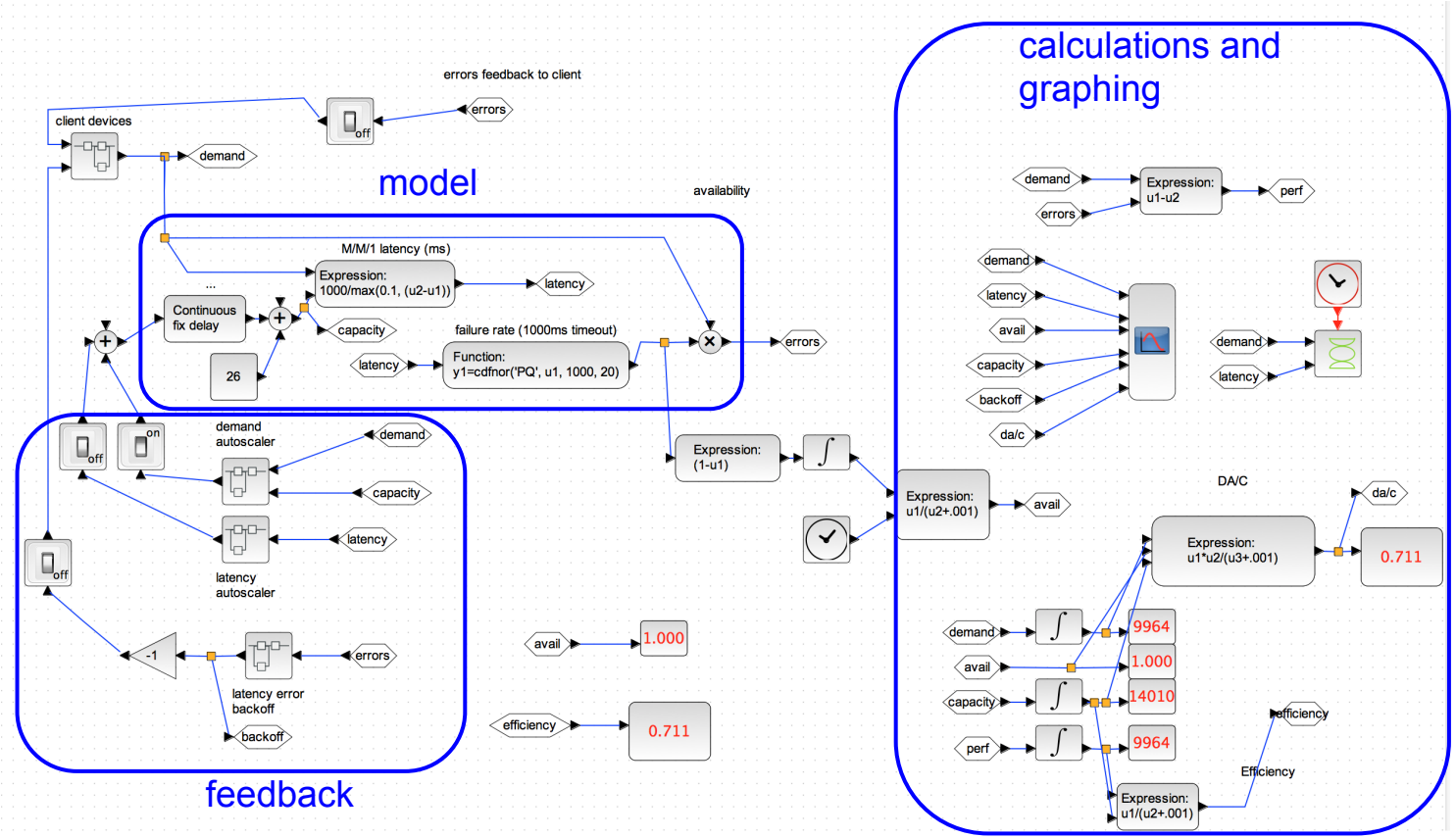
Simulating the Cloud

efficiency versus availability
strategies and tactics

Simulating The Cloud



SciLab Cloud Model



Parameters

- nominal capacity = 26 rps
- demand = (20 + 10*sin(.05t) + norm(0.5)) rps (overloaded)
- various feedforward/feedback compensators.
  - open-loop
  - demand throttling
  - latency-error backoff
  - demand autoscaling
  - latency autoscaling

Measurements

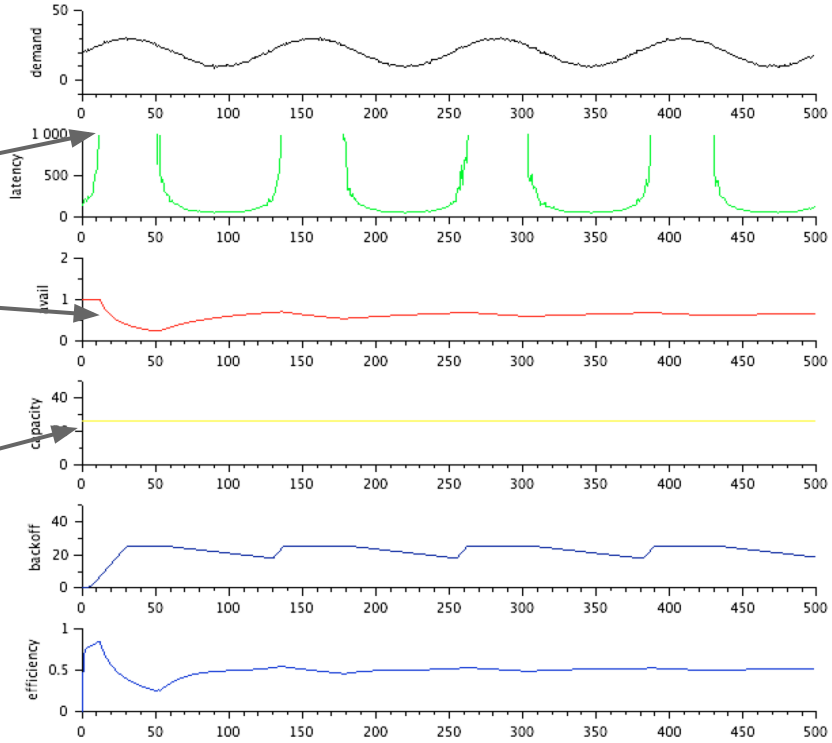
- Demand
- Latency
- Availability
- Capacity
- Efficiency

Open Loop Behavior

latency > 1000 ms

availability drops

fixed capacity



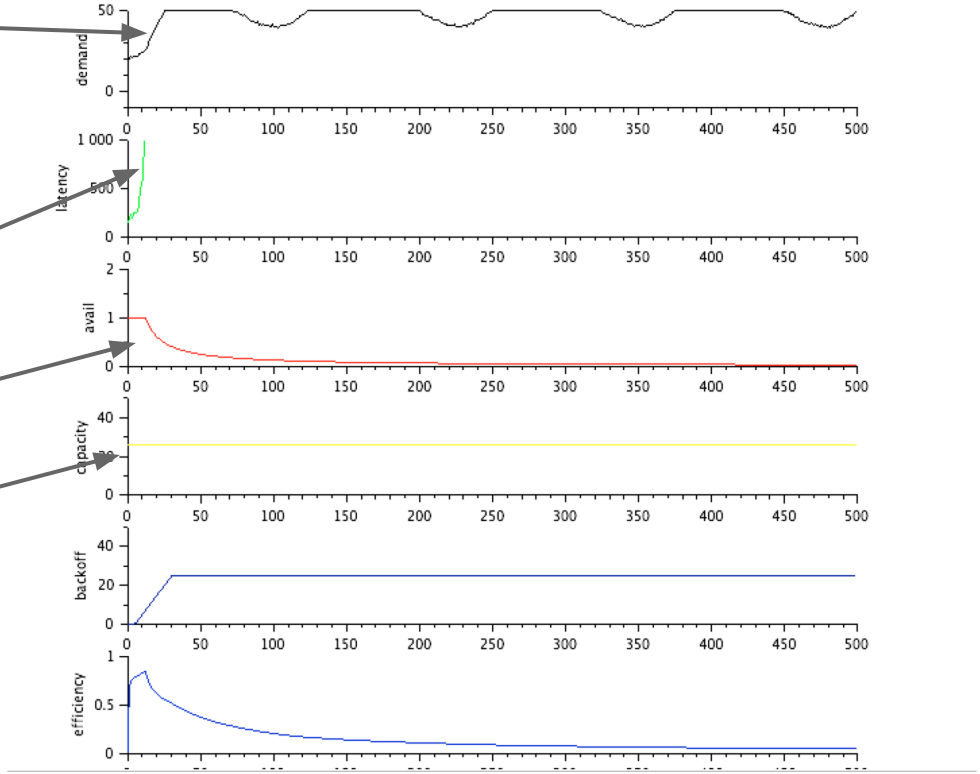
Thundering Herd

error-based retries cause thundering herd

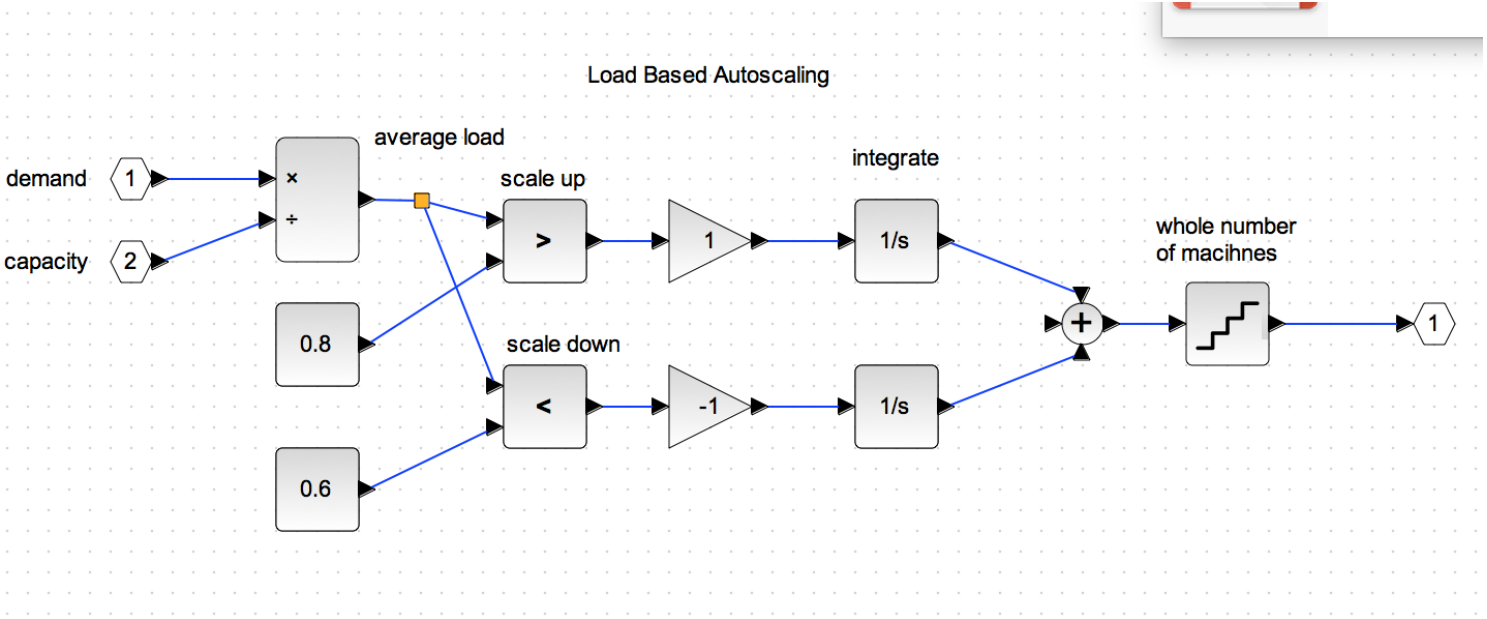
permanently overloaded

unavailable

fixed capacity



Load-Based Autoscaler

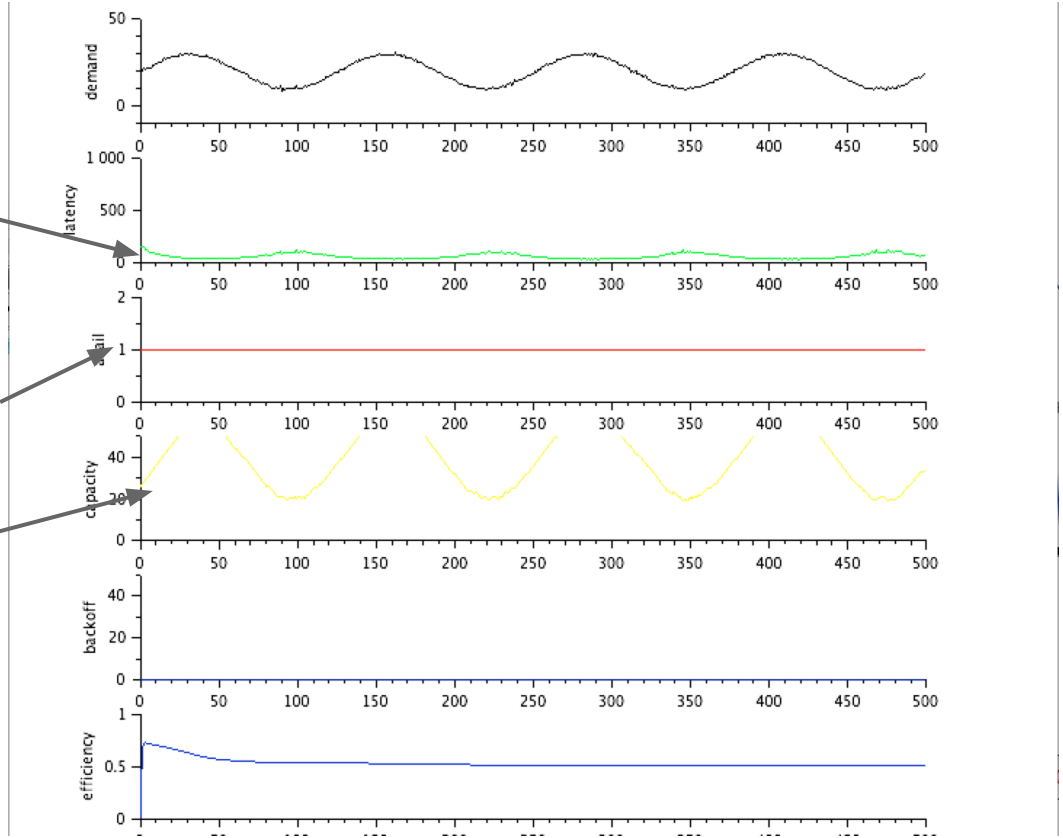


Load-Based Autoscaling

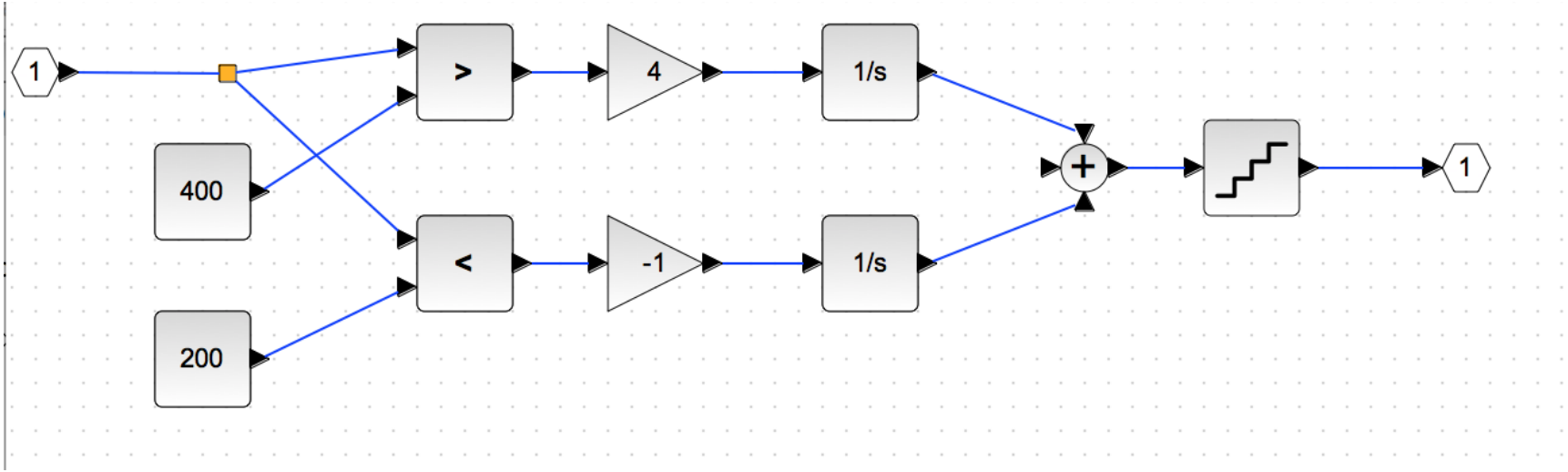
latency <<< 1000 ms

100% available

variable capacity, over provisioned



Latency Autoscaler



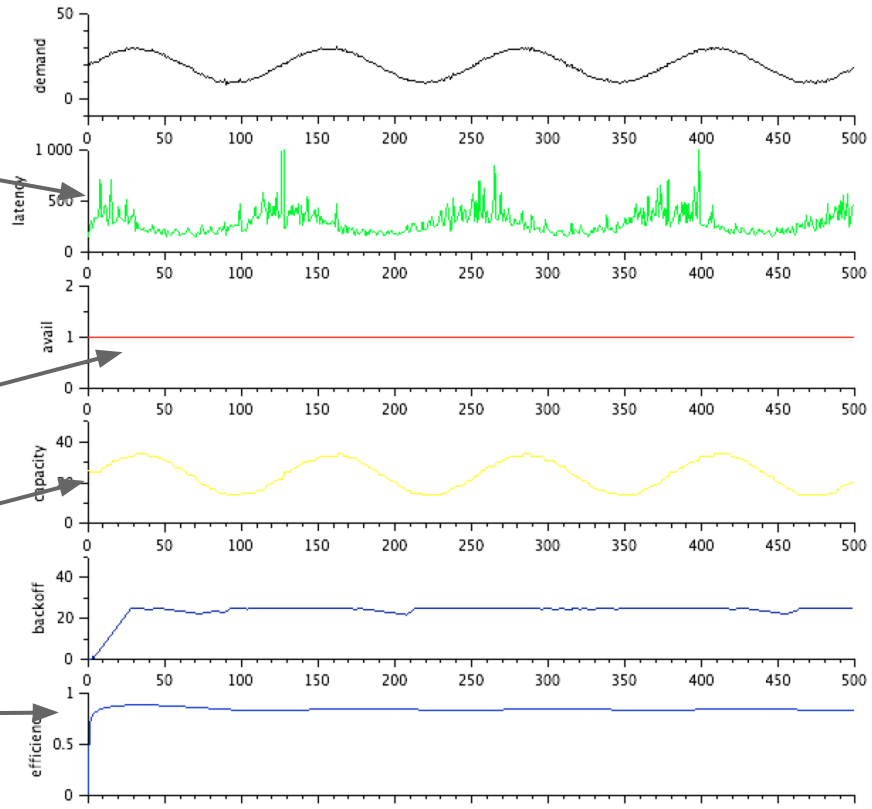
Latency-Based Autoscaling

latency < 1000 ms

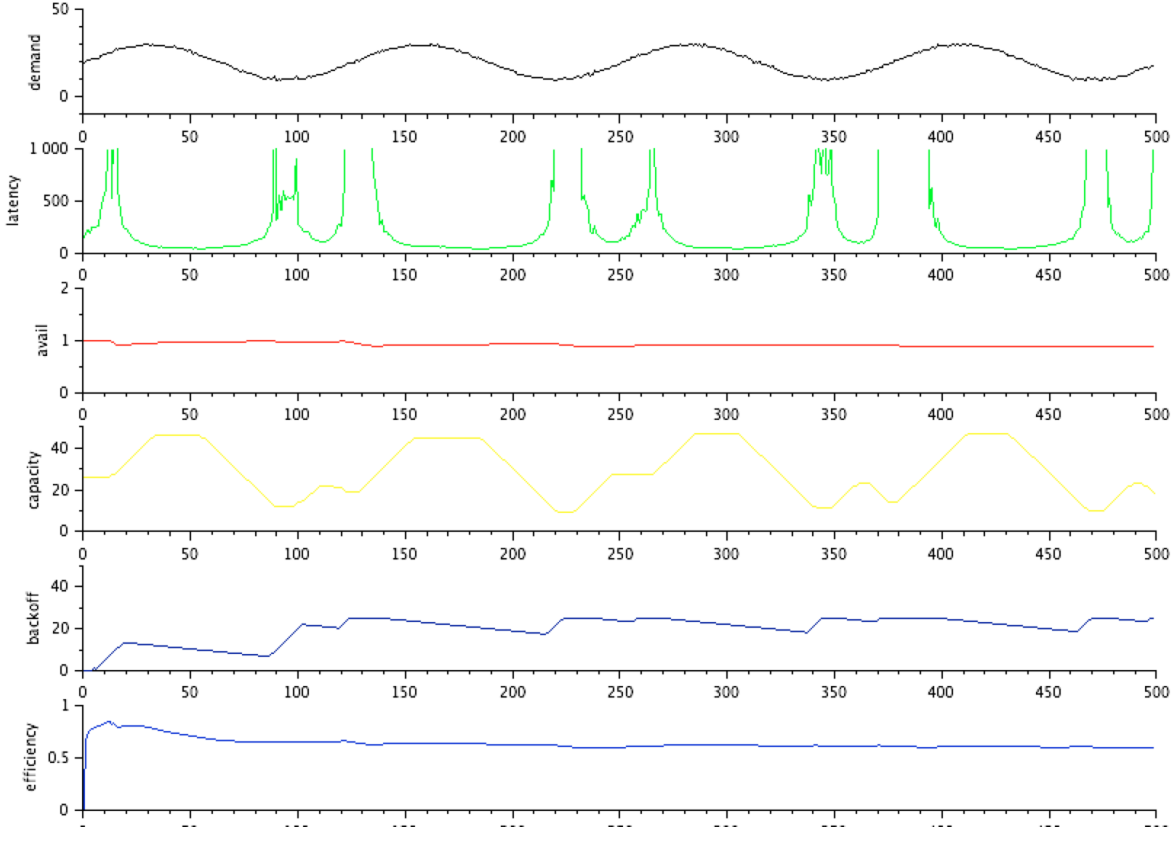
99.9% available

variable capacity

most efficient



Demand Autoscaling/ Provisioning Delay



Shedding Load

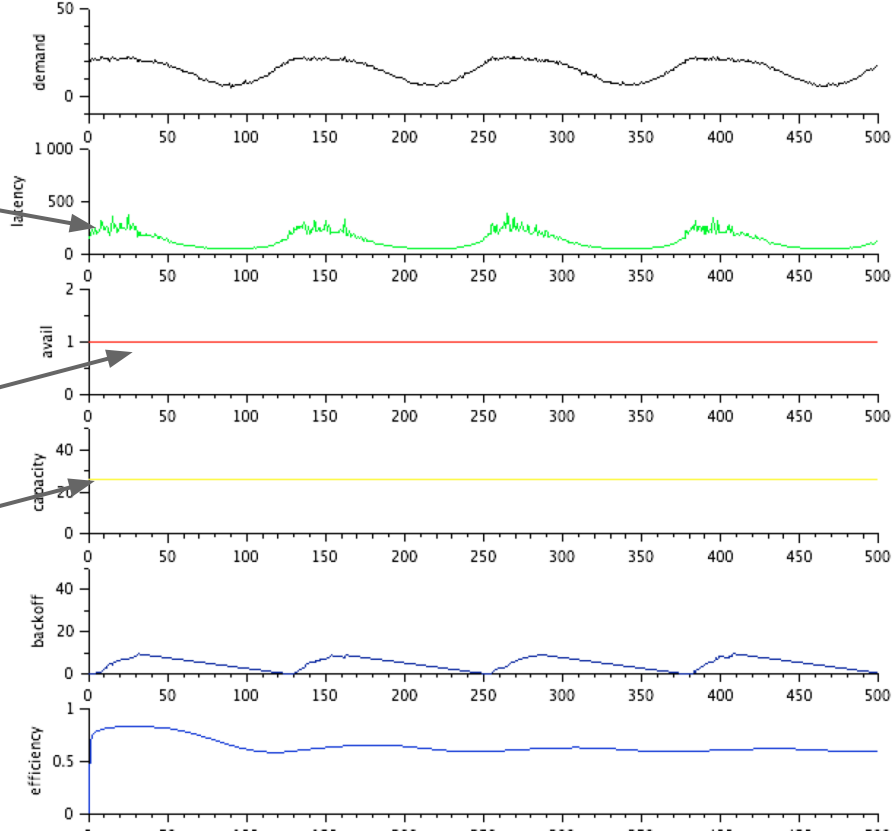
a viable strategy when clients can be slowed down

Latency Error Feedback

latency << 1000 ms

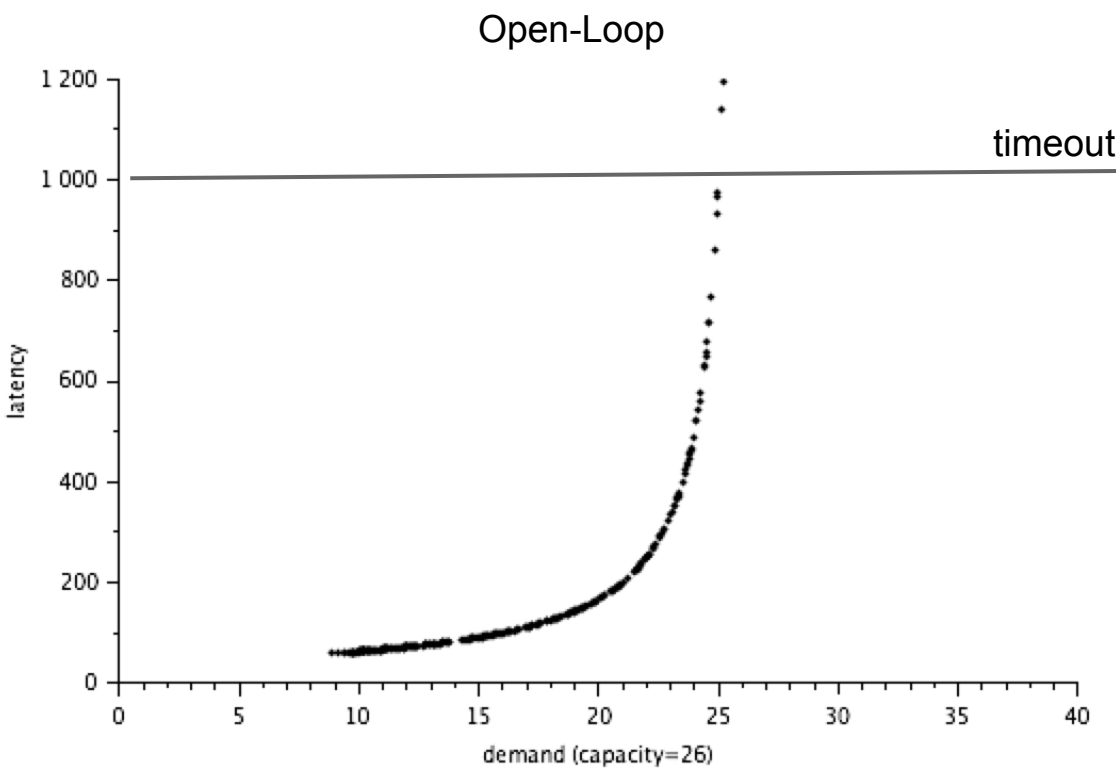
100% available

fixed capacity

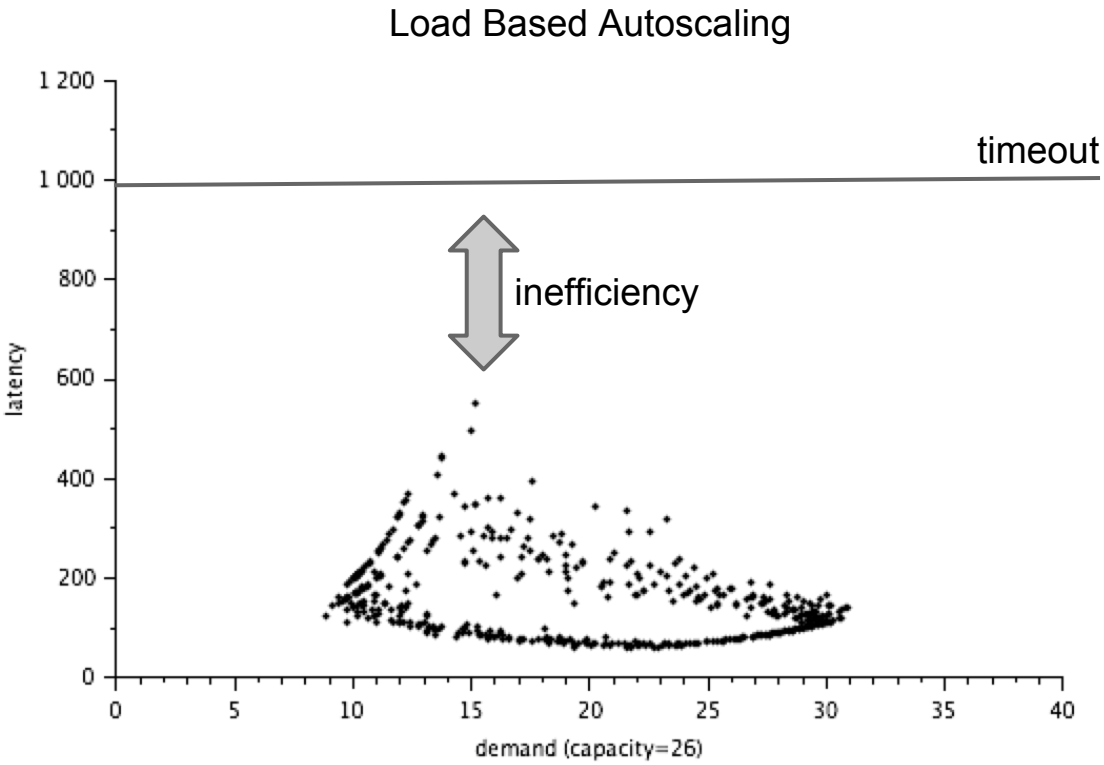


Characteristic Load/Latency Graphs

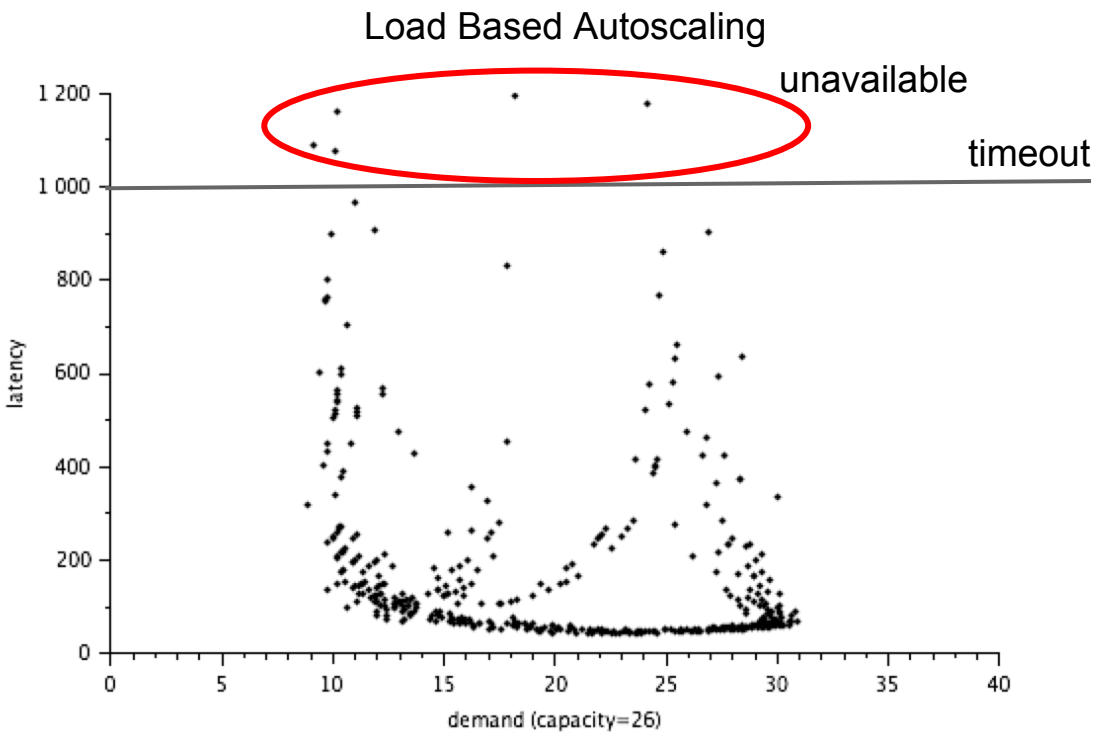
Load/Latency Curves



Load Based Autoscaling



Effect of Provisioning Delay

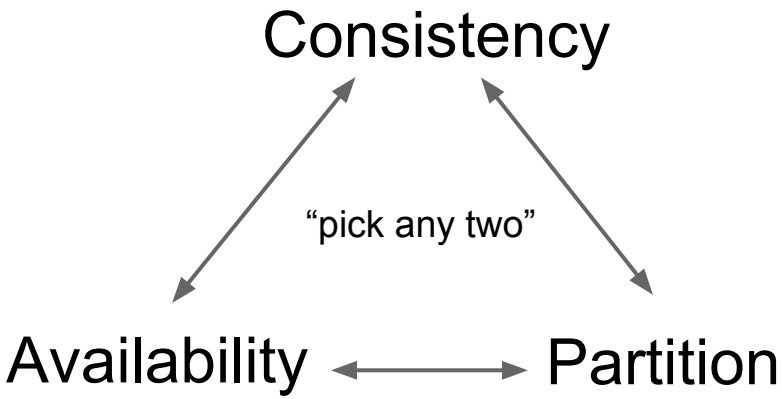


Limits To System Performance

- We have the CAP Theorem in Computing
- What are the limits to Efficiency?

CAP “Theorem”

- a distributed system cannot simultaneously provide consistency, availability and tolerance of network partition

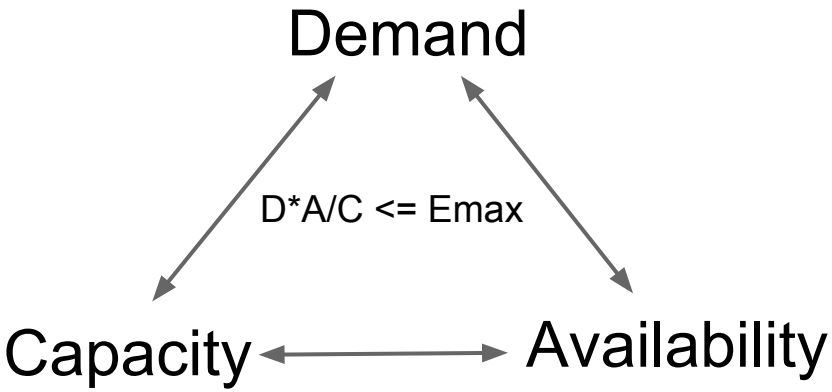


Define Efficiency

Efficiency = Demand * Availability / Capacity

Efficiency “Theorem”?

- Demand*Availability/Capacity <= Emax
  - to meet demand and availability, you must provision more.
  - to improve availability at a given capacity, you must accept reduced demand.
  - The constant Emax depends on the architecture, and represents a a constraint on system *Efficiency*.



Simulation Efficiencies

- some architectures achieve better Efficiency than others, and are therefore measurably ‘better’.

System	Demand	Availability	Capacity	Efficiency
Open Loop	9964	0.669	12977	0.408
Error Retry	23255	0.024	12977	0.021
Latency Error F/B	7630	1	12977	0.588
Demand Autoscaling	9964	1	14010	0.711
Latency Autoscaling	9964	0.999	11934	0.834

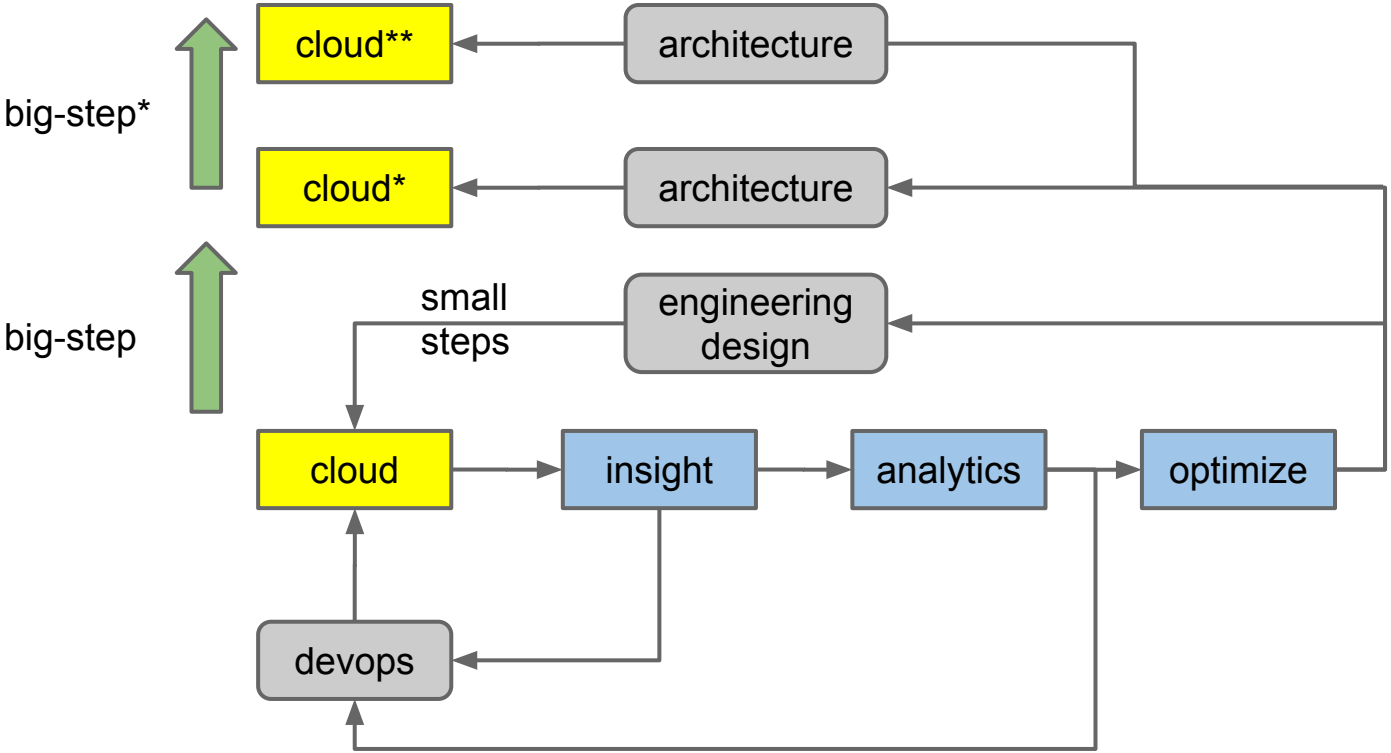
no availability

highest efficiency, lowest capacity three-nines availability

Feedback Systems

Layers and Layers

Continuous Improvement



Layers

- feedback systems:
  - insight = dashboards, operations
    - e.g. reconfigure machines, diagnose failures
  - analytics = statistics, signal processing
    - e.g. change provisioning capacity
  - engineering design
    - e.g. redesign systems, deploy new code, make things a little better
  - architecture
    - e.g. make big changes, e.g. change from SQL to noSQL data stores, big improvements in efficiency

Challenges

- getting the big wins while staying available
- deploying new code (canaries)
- breaking monoliths into microservices while continuing operations
- latency and errors
- latency and measurement (eventually consistent metrics).

Why Does Efficiency Matter?

$/customer during growth will kill cloud startups

Define Efficiency

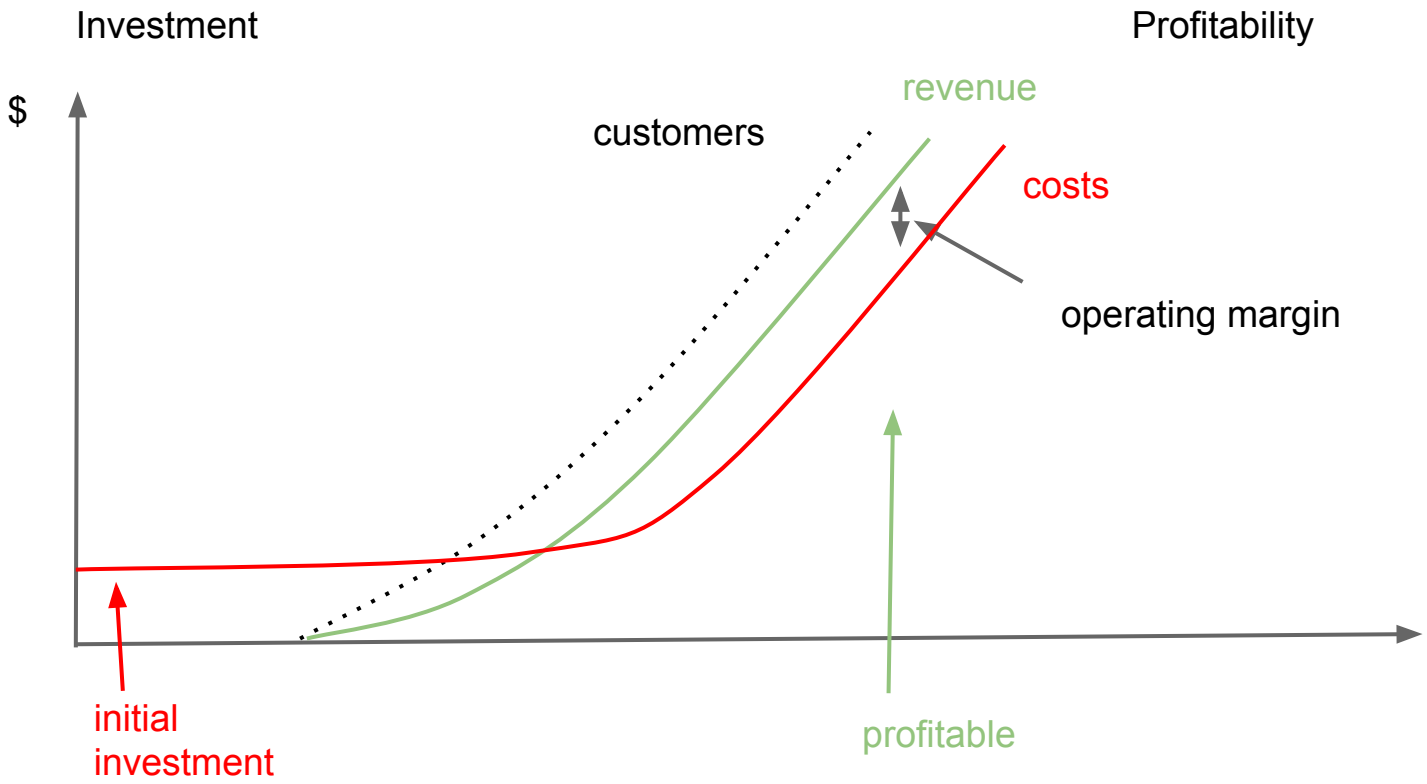
- one definition:

  #customers-served/$cost-to-serve

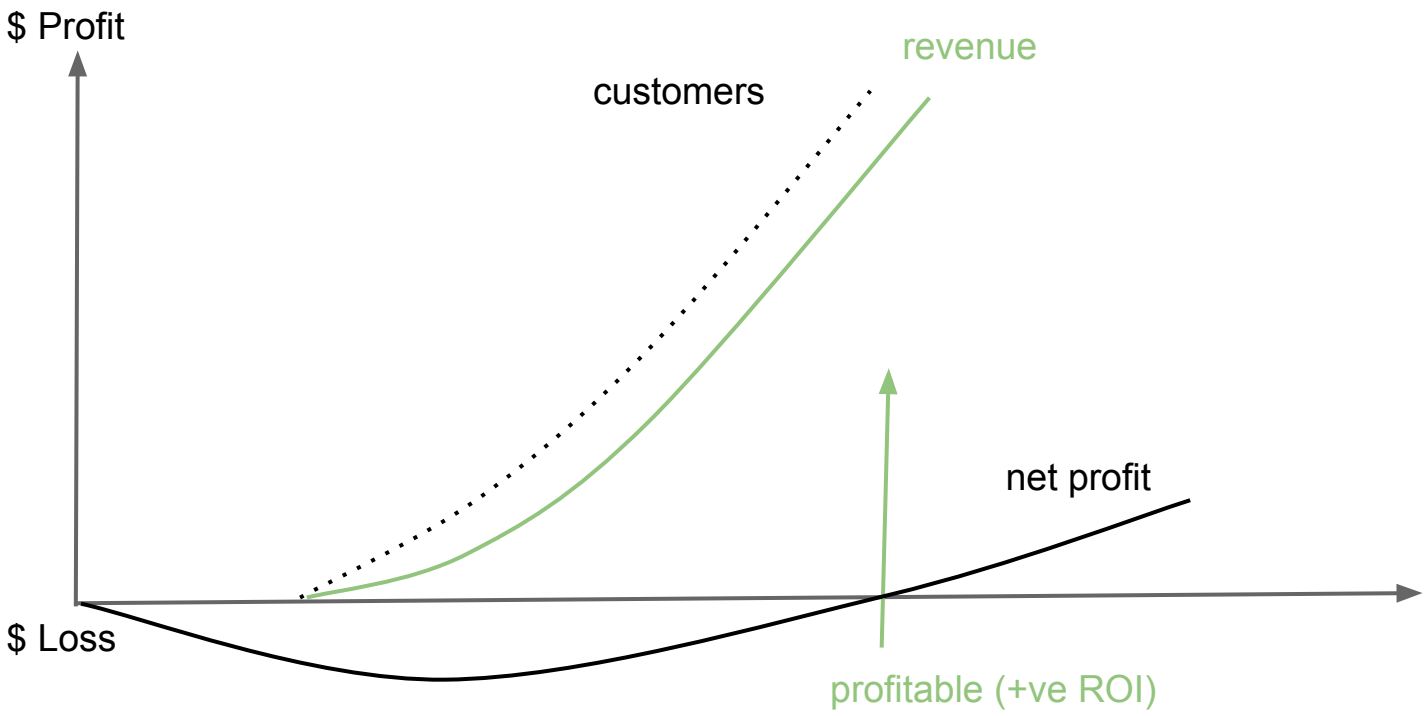
Conventional Business Dynamics

- initial fixed-cost investment
- linear customer growth
- costs scale with customers
- revenues scale with to customers
- cutomer revenue drives costs
- investment = barrier to entry
- predictable from initial investment to profit

Conventional Business



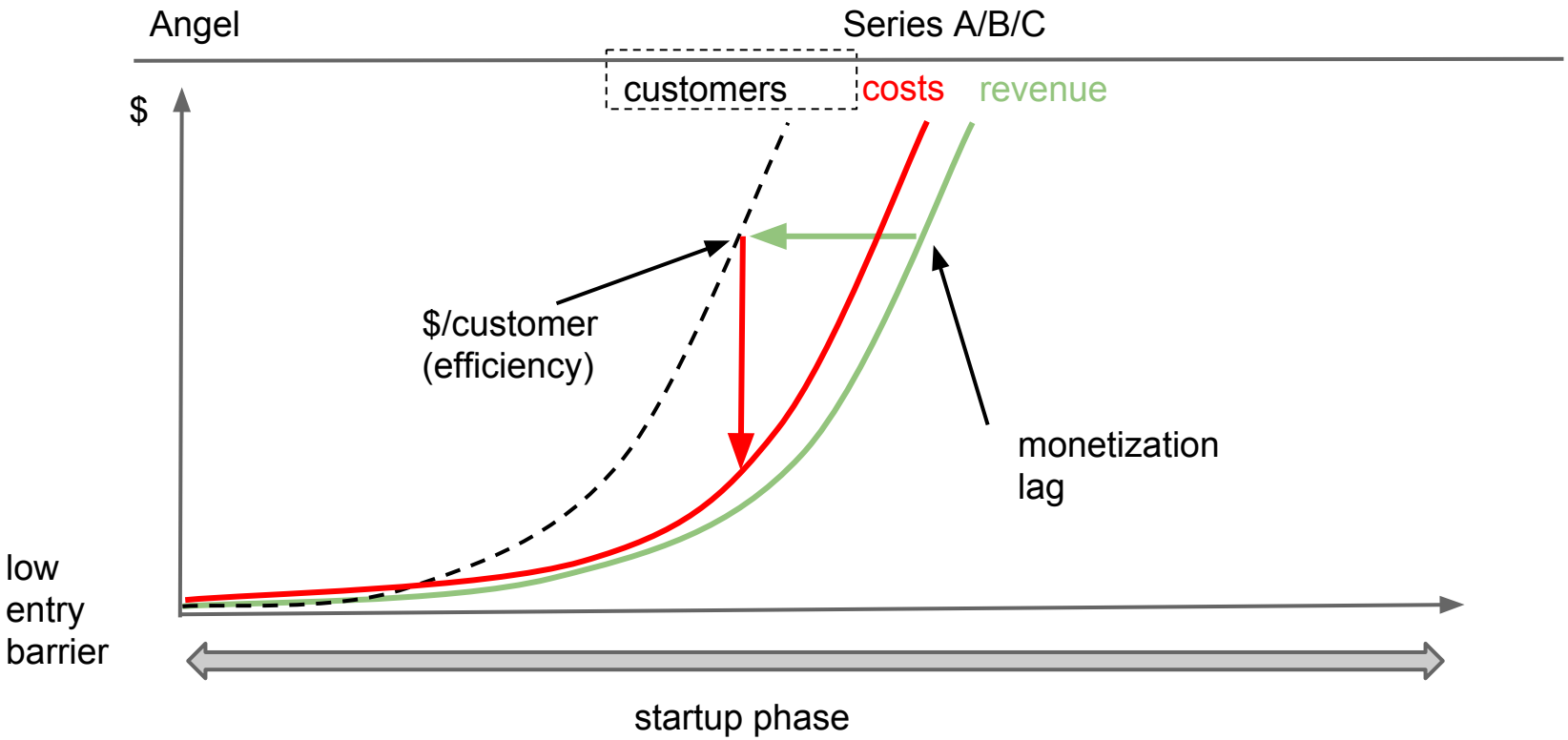
Conventional Business P+L



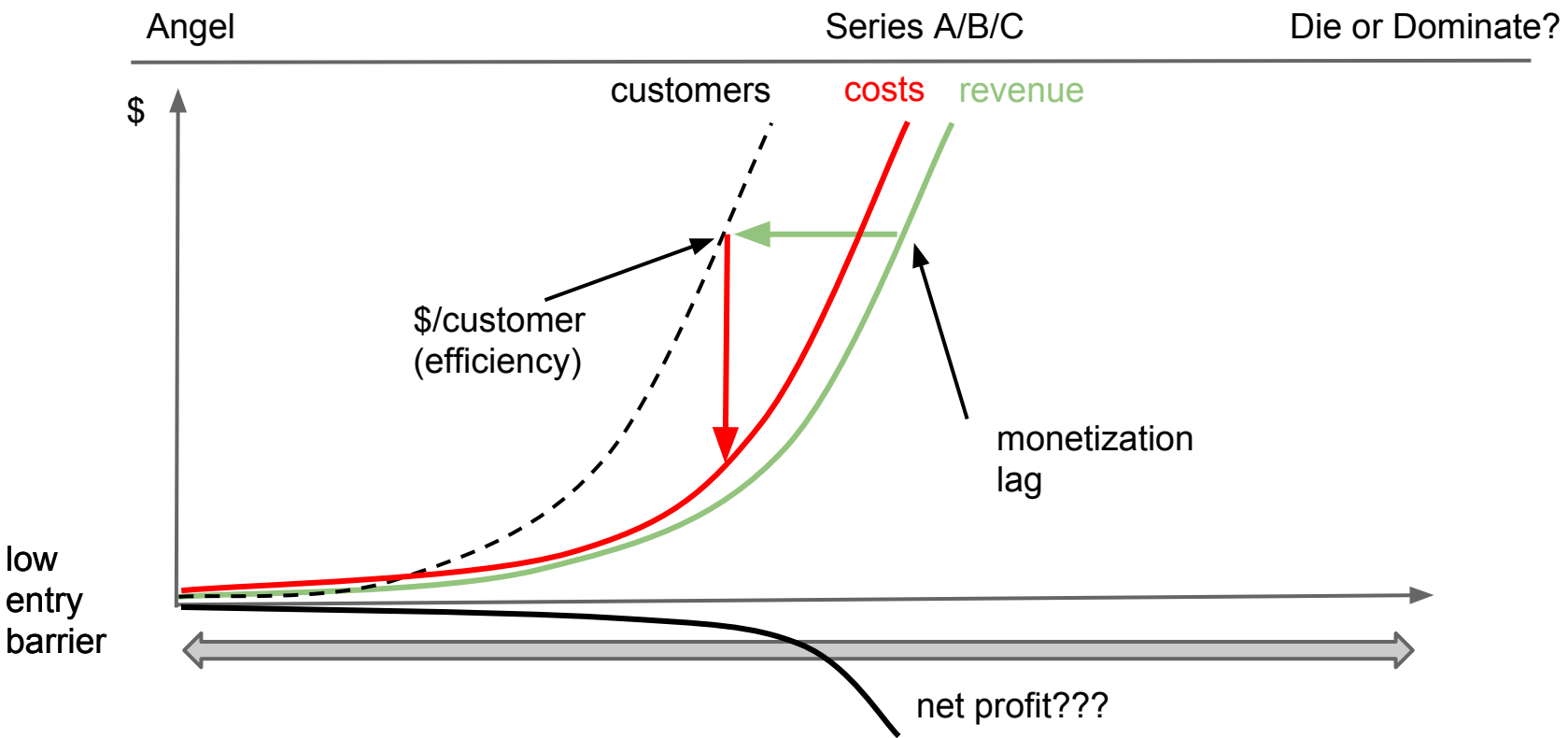
Cloud Business Dynamics

- Easy to get started
- Easy to scale
- Easy to get eaten by the cost
- Easy to get eaten by hungry wannabes.
- Move Fast, Be Efficient

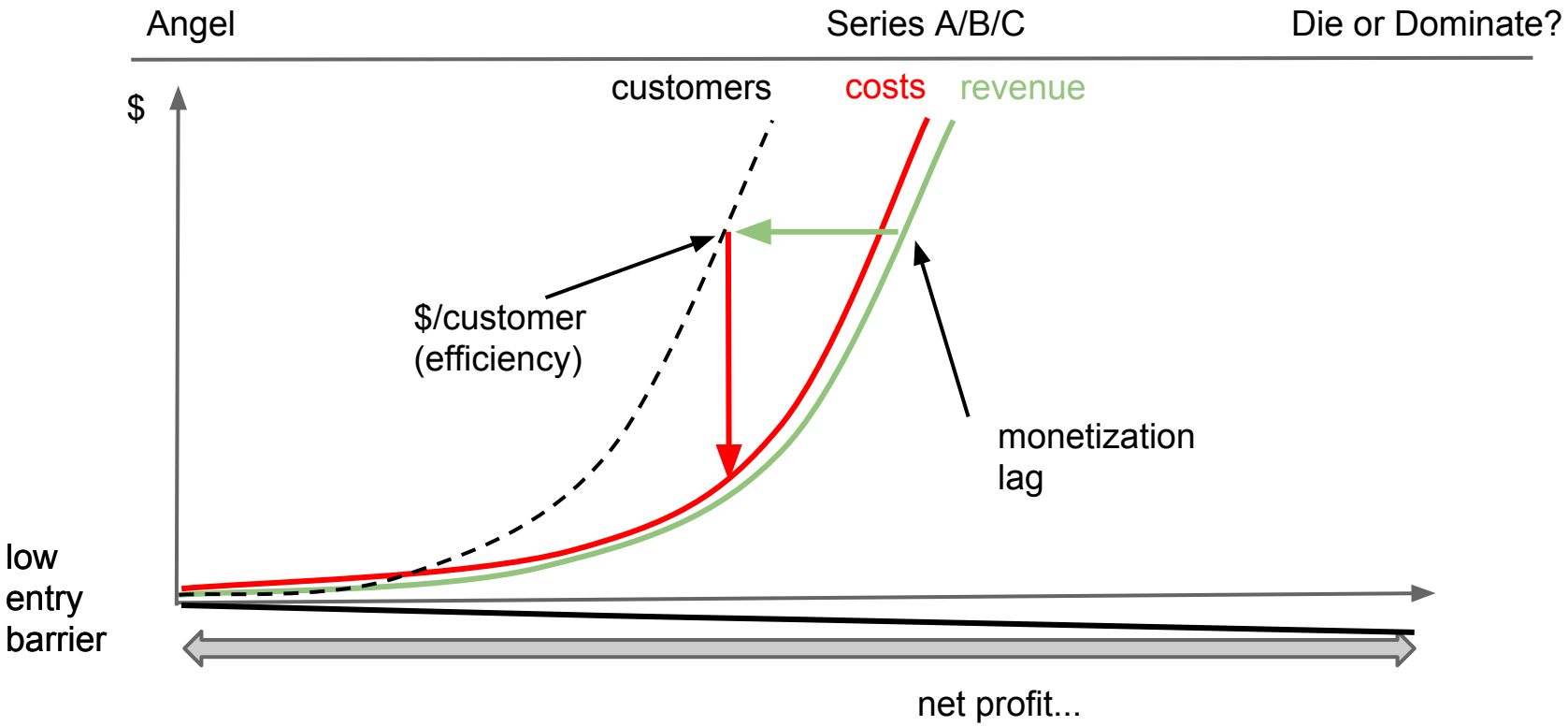
The Cloud Startup Company:



Cloud Startup P+L:



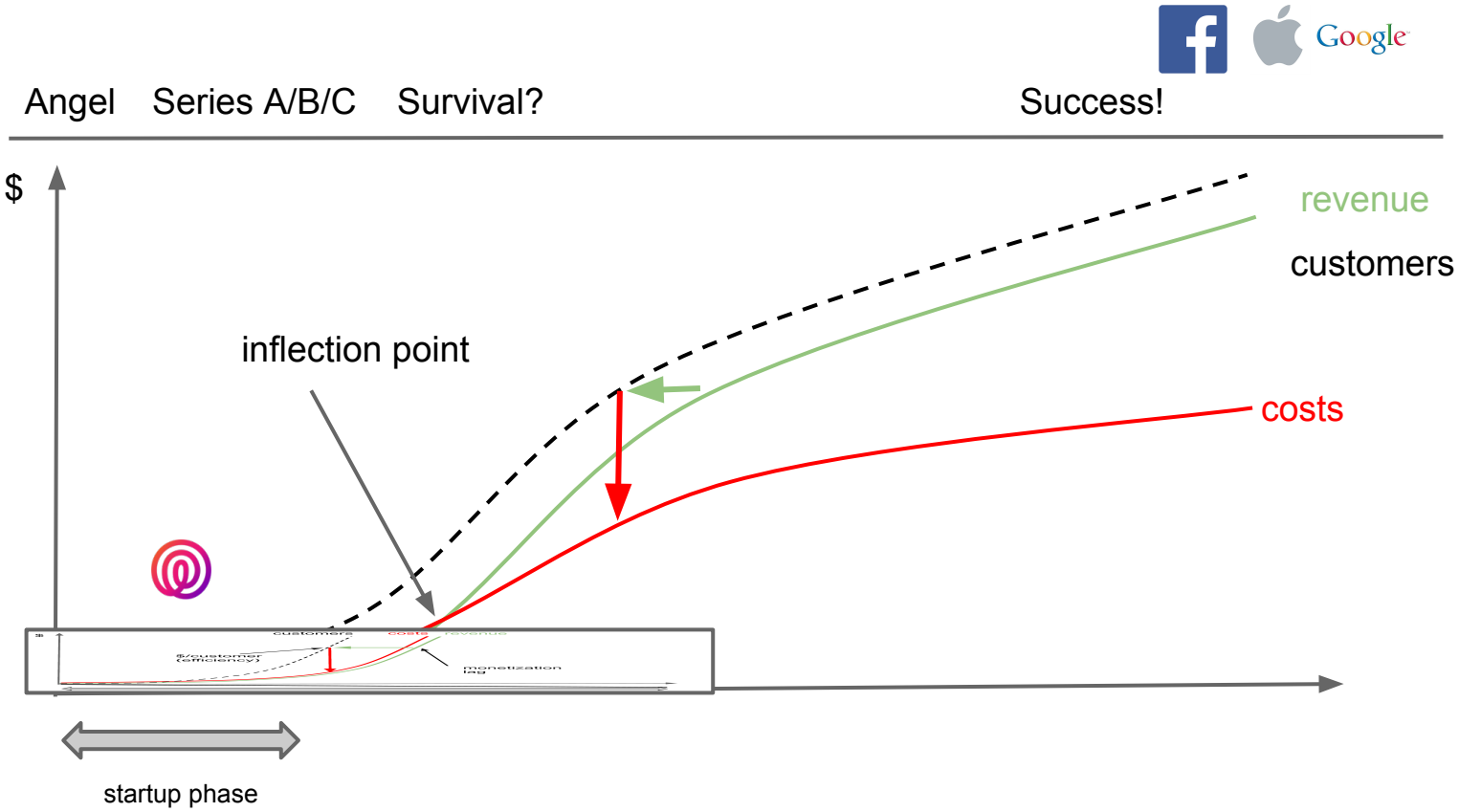
Better Efficiency = Runway



Cloud Business Dynamics

- Low initial costs
- Explosive Customer Growth
- Costs proportional to customers
- Lag from engagement to revenue
- Operating costs can grow exponentially and kill a business.

Crossing the Cloud Chasm:



Crossing the Cloud Chasm*

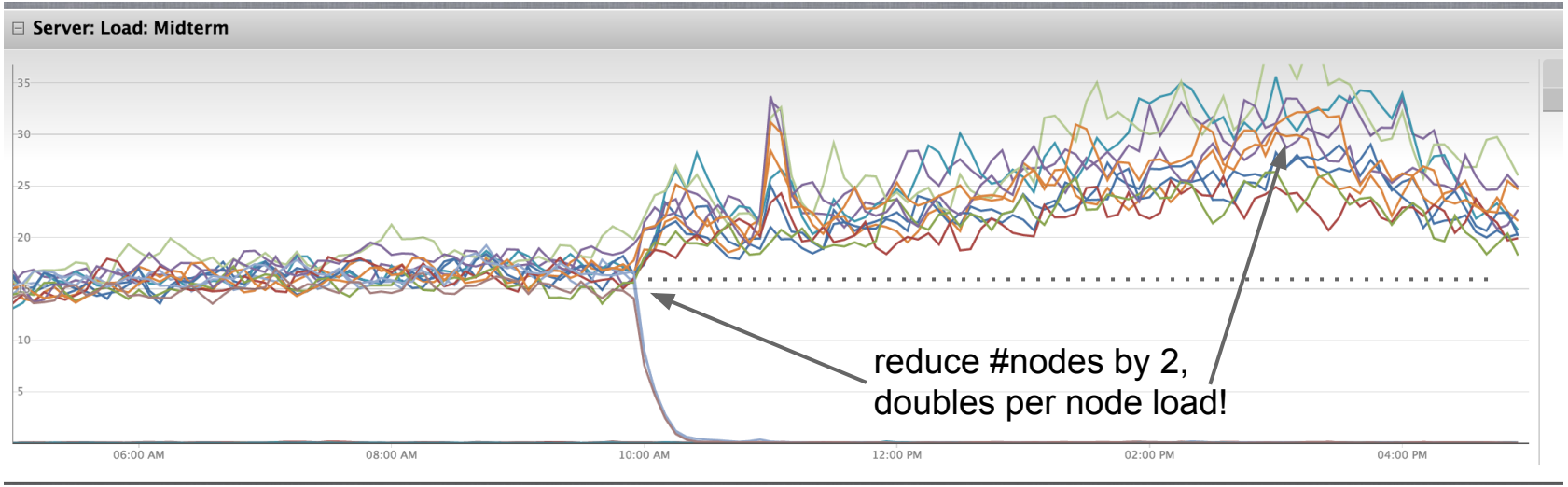
- Driving Revenue is difficult
- Cost containment is difficult
- Efficiency is at least causal.
- Feedback systems can make the difference between survival and success

* this is not the same Chasm as Geoffrey Moore's Chasm (http://crossingthechasmreview.blogspot.com/), but has the same potential for destruction. This chasm has a black hole at the bottom of it, pulling you in.

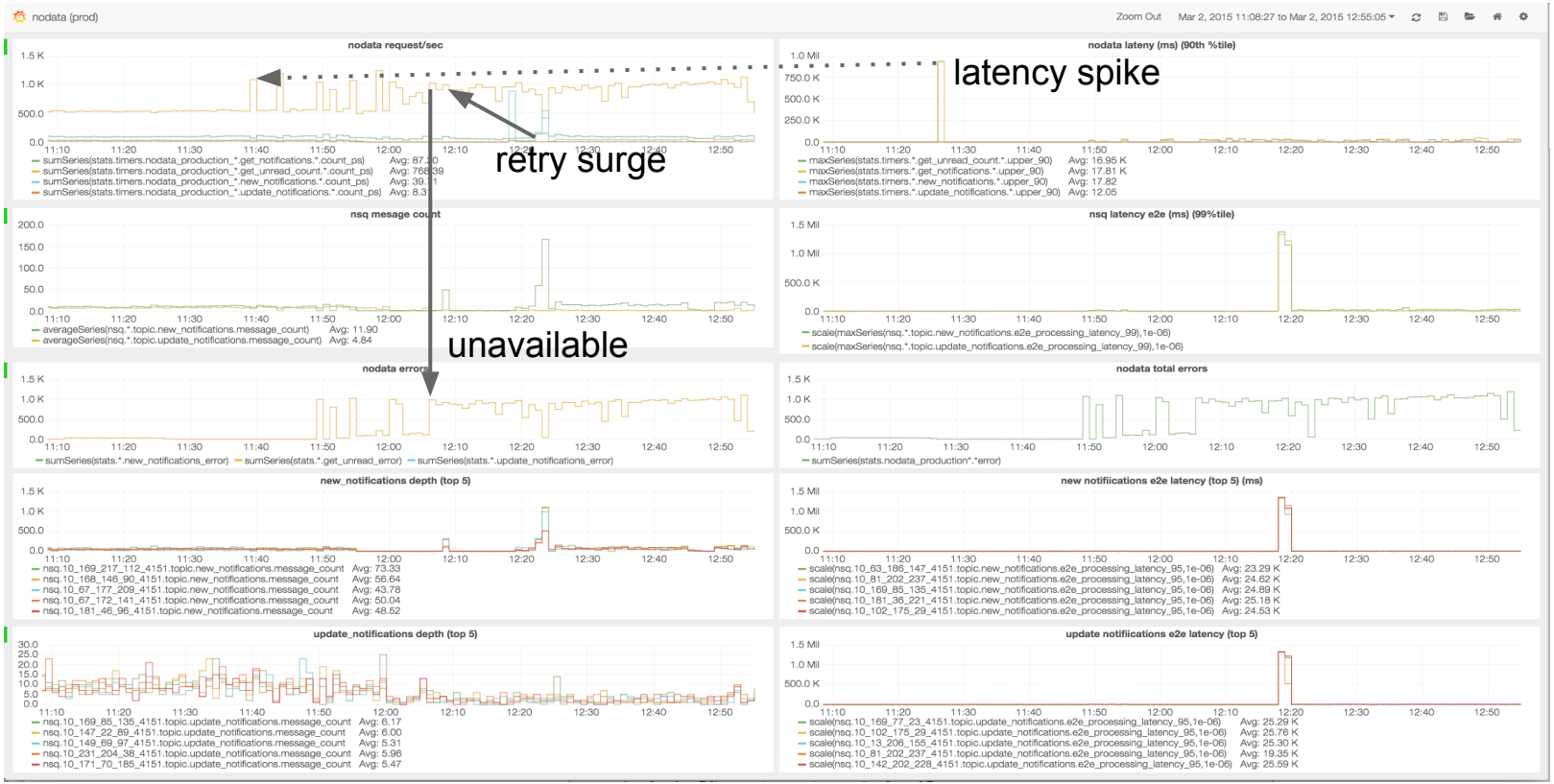
Examples from a Startup

Examples from Life360

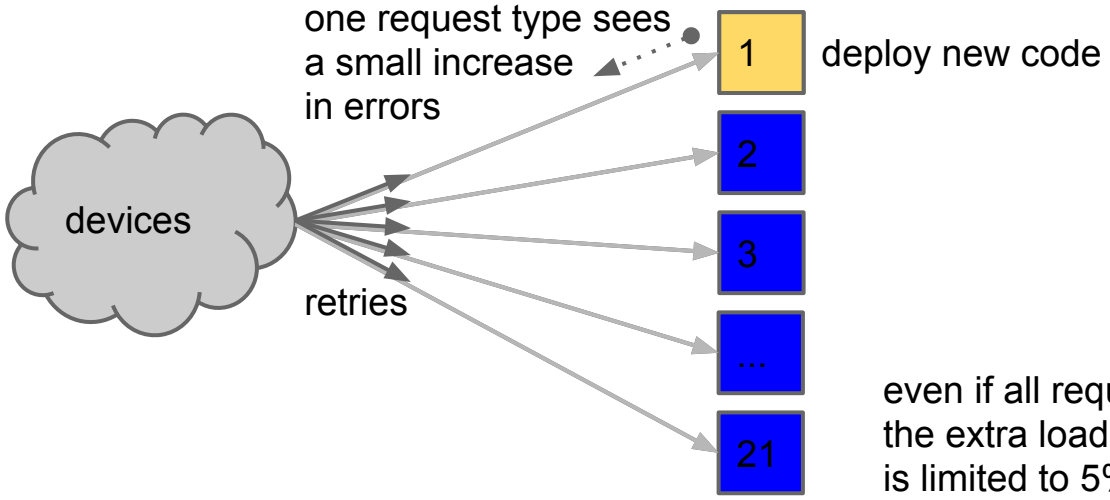
- tuning for cost:



Thundering Herd

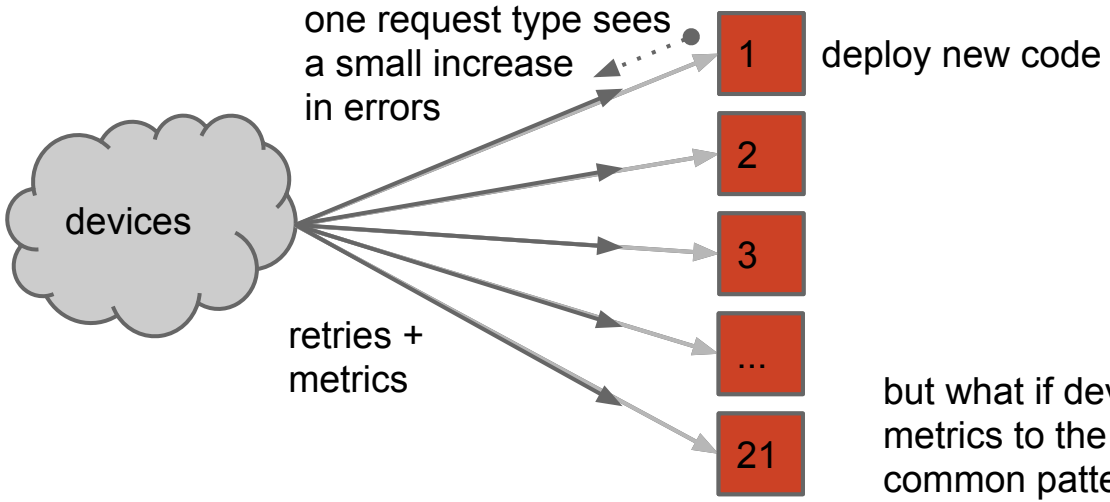


Self Inflicted Herds



even if all requests to #1 fail
the extra load on other machines
is limited to 5% (1/21), right?

Self Inflicted Herds



but what if devices also report failure metrics to the same servers (a common pattern)?

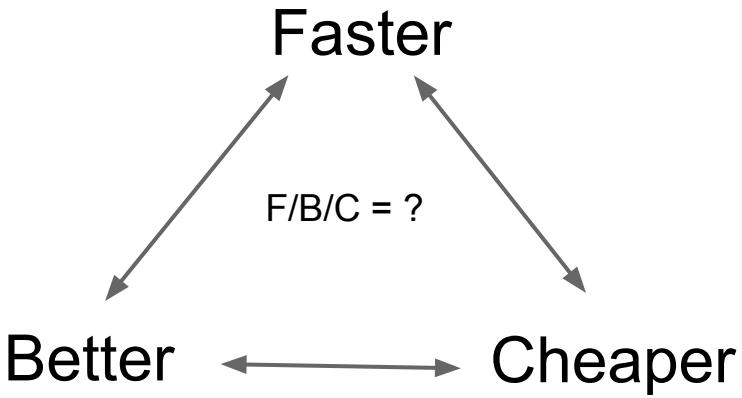
Self Inflicted Herds

- There is now a positive feedback loop
- If the metrics packets are larger than the original requests, then the loop-gain is > 1, and the system will fail during deployment, but not at the *first* canary
- Send your device metrics to a different tier, so if it fails your devices stay up.

What Can We Learn?

Faster/Better/Cheaper

- Can you achieve all three?

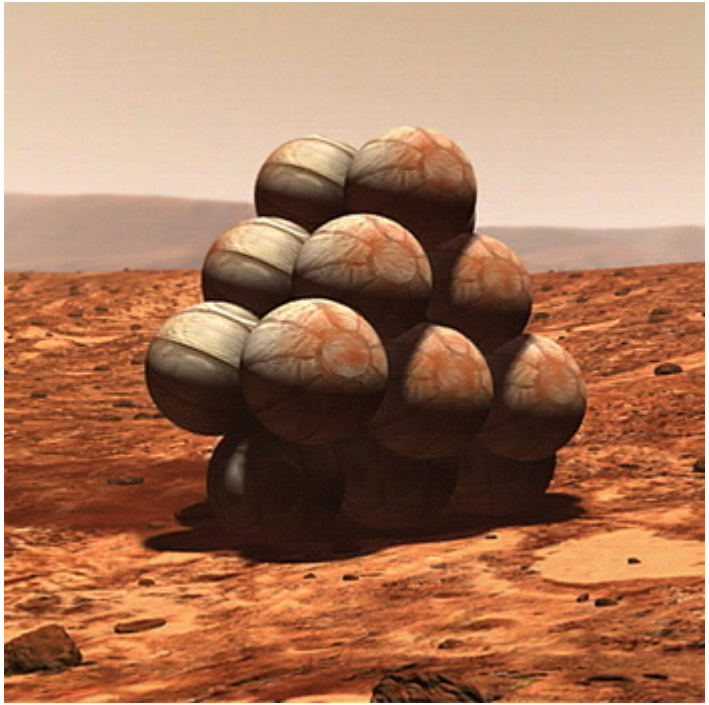


Mars Rover

Faster ✓

Better ✓

Cheaper ✓



http://mars.nasa.gov/mer/mission/images/step15_br350.jpg

Cloud Computing F/B/C?

- design out thundering herds
- analytic computation of efficiency & availability
  - end-to-end trace of latency & errors
- making truly micro micro-services
  - provision fast, run fast, fail fast.
- stream processing data-pipelines

Cloud Computing F/B/C?

- feedback & feedforward on real-time data
  - allows continuous dynamic optimization
  - linearizes inherent nonlinearities
  - makes systems robust
  - improves dynamic performance (scaling)
  - increases efficiency by running closer to the point of instability

A Final Thought

- What should cloud computing really look like?
- And how do we get there?

A Final Thought

http://upload.wikimedia.org/wikipedia/commons/4/42/Marching_band_drummers_in_parade_at_Texas_State_Fair_2007.jpg

A Final Thought

** You'd be surprised at how much goes on in a marching band.*

In one moment, every single member has to think about many different things.

Some of them are for the individual. Am I playing the music correctly? Am I in time with everybody else? Am I in step? Is my instrument held like everybody else's?

Others deal with the band as a whole. Am I in line with everybody else? Am I a good distance from the line in front? Am I a good distance from the other people in my line?

The marching band is very volatile. If one section drags or speeds up, it radiates throughout the rest of the band, causing it to get out of step.

This brings us to the job of the drum major. The drum major is the person at the front of the band responsible for keeping time and choosing when to start and stop the band.

* David Tuffs, High School Marching Band, Trumpet.

This seems an ideal metaphor for a cloud microservices system. The elements of individual and collective feedback, combined with orchestration, achieve a robust and resilient system. We can learn much from this.

Questions?