Aperture

An algorithm for non-cooperative, client-side load balancing.

Ruben Oanta
@rubenoanta

SERVICE-TO-SERVICE LOAD BALANCING

capacity utilization
safely make use of aggregate capacity of replicas

failure management
route around replicas when they inevitably fail

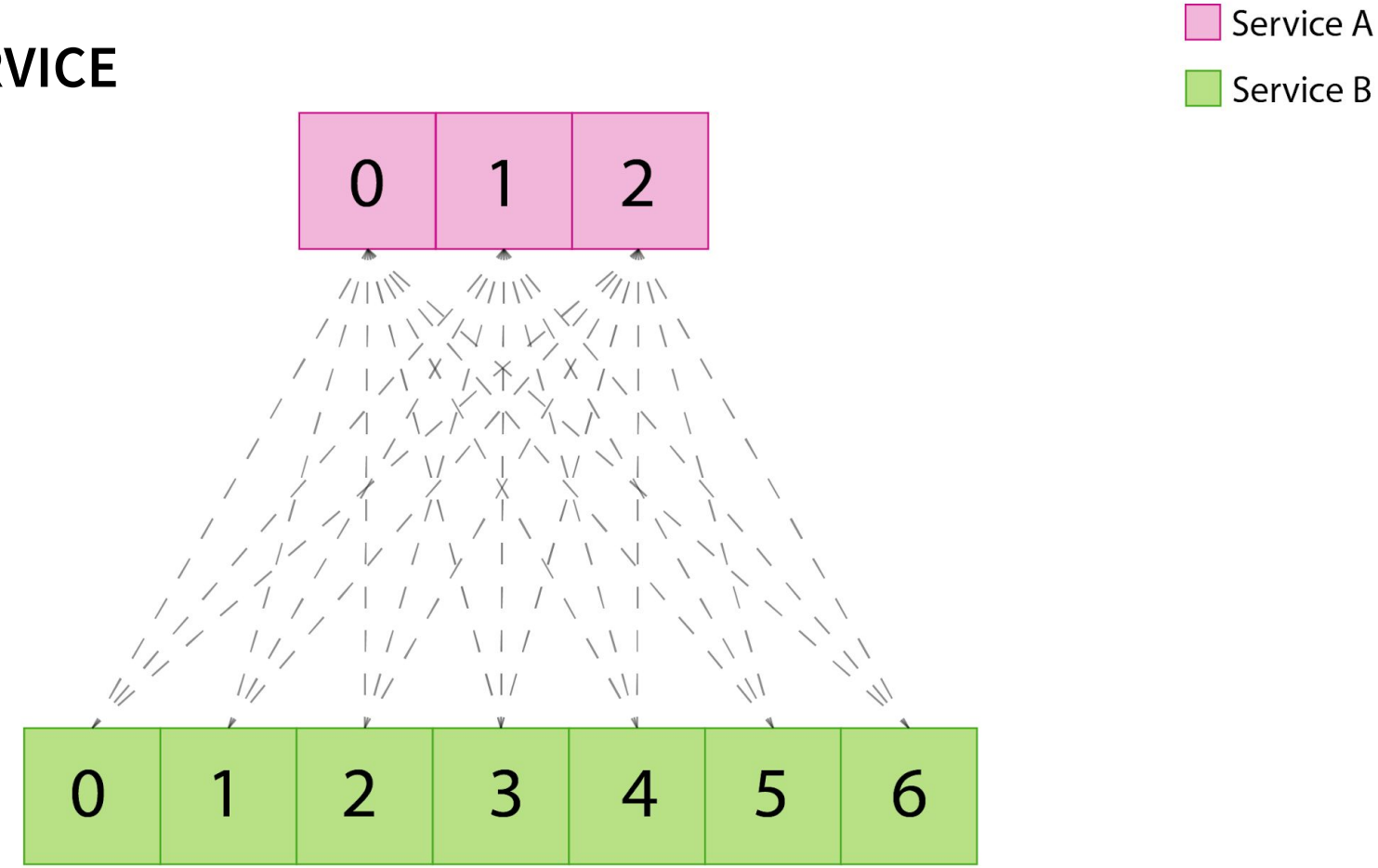
SERVICE-TO-SERVICE LOAD BALANCING

non-cooperative
multiple load balancers which make decisions independently

client-side
embedded within each replica of a service

load balancing
over sessions (OSI L5) and requests (OSI L7)

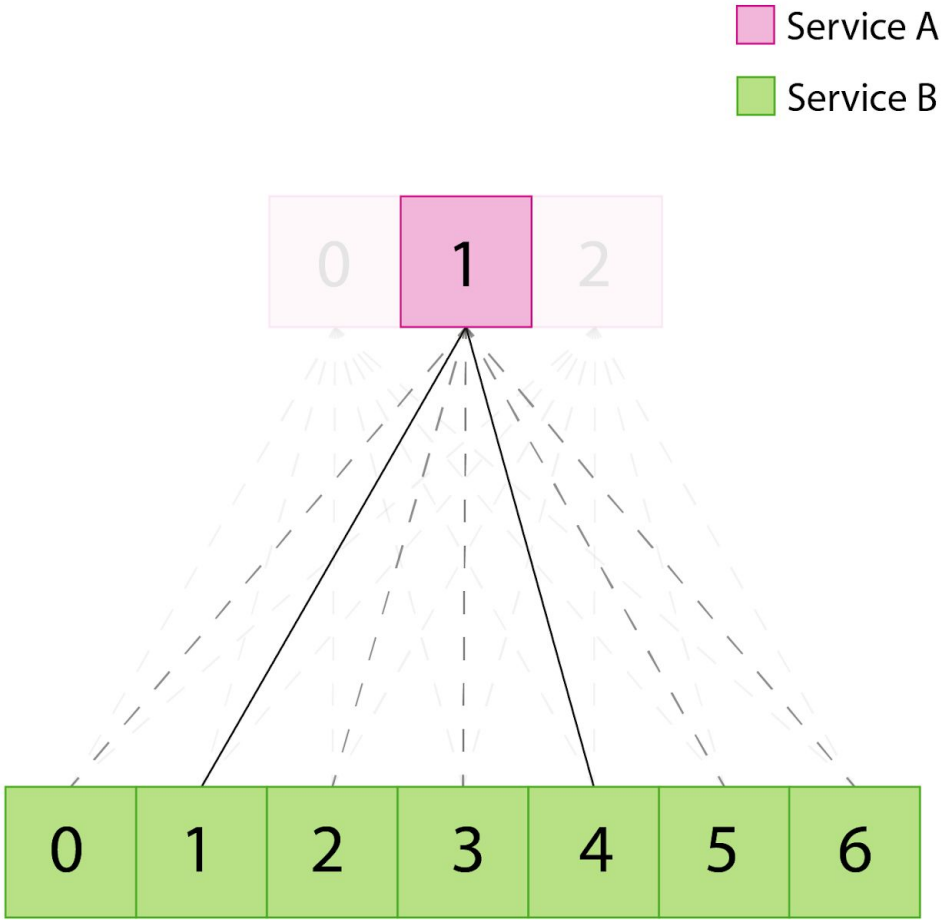
EXAMPLE SERVICE TOPOLOGY



PER REQUEST: P2C

1. Select two instances uniformly and randomly.

2. Of the two, select the instance with the fewest number of outstanding requests.



PER REQUEST: P2C

fair request distribution
request load is even with homogenous replicas

efficient
fully concurrent, constant time for selection + comparison

decoupled selection + comparison
allows for sophisticated interpretation of load

PER SESSION: IT'S A MESH!

wasted resources
everyone talks to everyone

no isolation
independently discover the same problems

low concurrency
poor load metric performance without concurrent requests

How can we reduce the number of sessions?

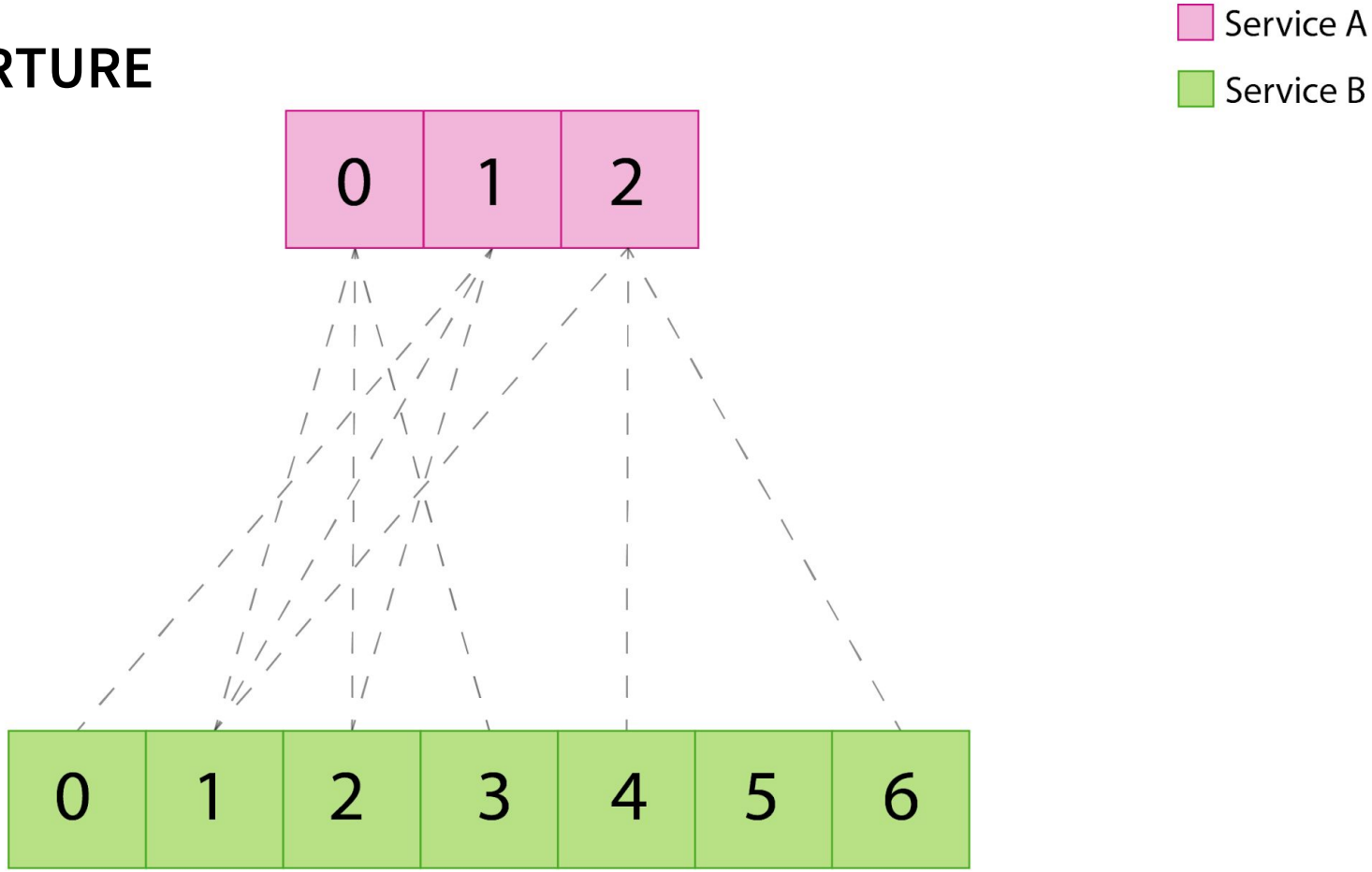
RANDOM APERTURE

high concurrency
aperture is smallest subset to satisfy concurrency

dynamic
can grow or shrink based on feedback controller

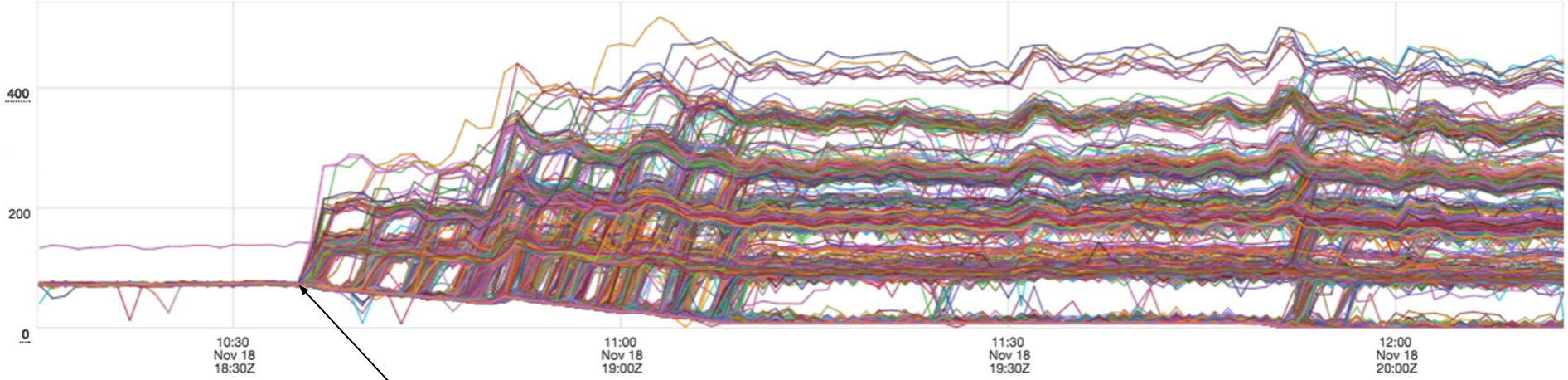
random
replicas selected in aperture window randomly

RANDOM APERTURE



RANDOM APERTURE: UNFAIR

RPS PER SERVER / TIME



clients deployed random aperture

RANDOM IS STATISTICAL

Results in a load distribution that closely resembles a binomial distribution. Minimizing the “banding” requires tuning which can only be eliminated when the aperture is the size of all the backend replicas.

CONFIGURED RANDOM APERTURE

WolframAlpha computational intelligence.

binomial distribution(500000, 0.001)

Browse Examples Surprise Me

Input:

binomial distribution	number of trials	$n = 500\,000$
	probability of success	$p = 0.001$

Open code

Statistical properties:

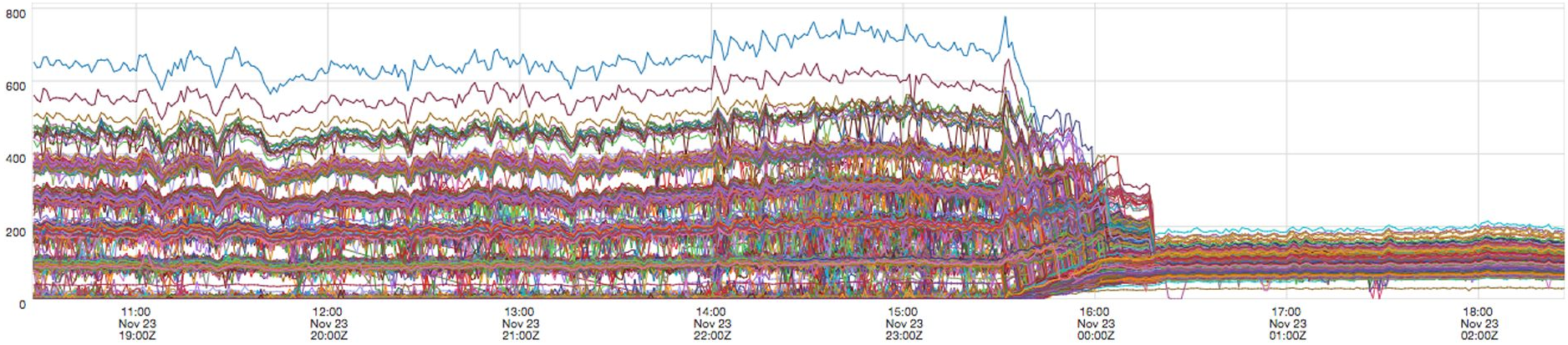
More

mean	500
standard deviation	22.3495
variance	499.5
skewness	0.0446542
kurtosis	3.00199

Probability density function (PDF):

CONFIGURED RANDOM APERTURE

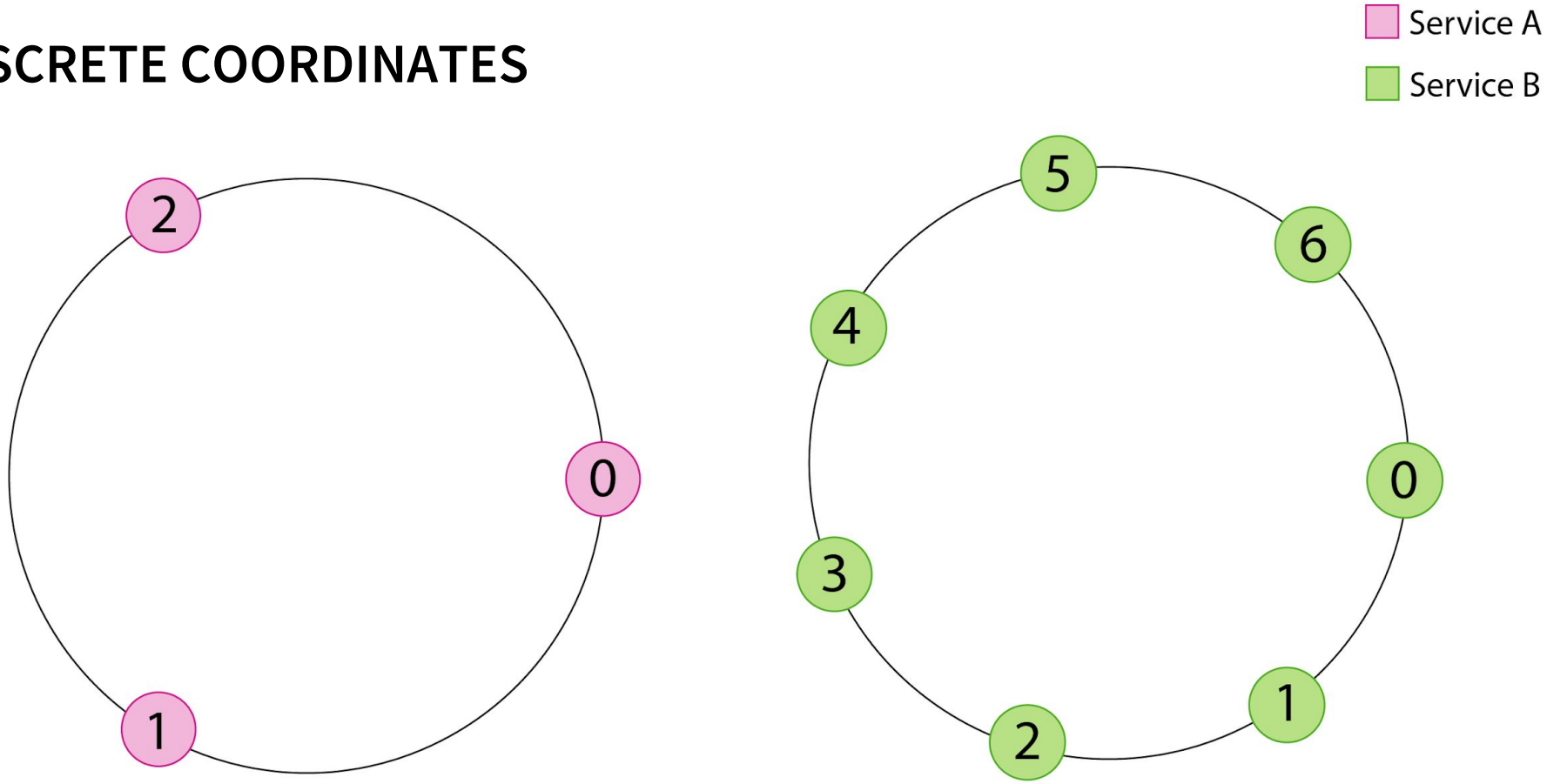
RPS PER SERVER / TIME



Distributing the configuration burden for core pieces of infrastructure will likely converge to poorly configured infrastructure.

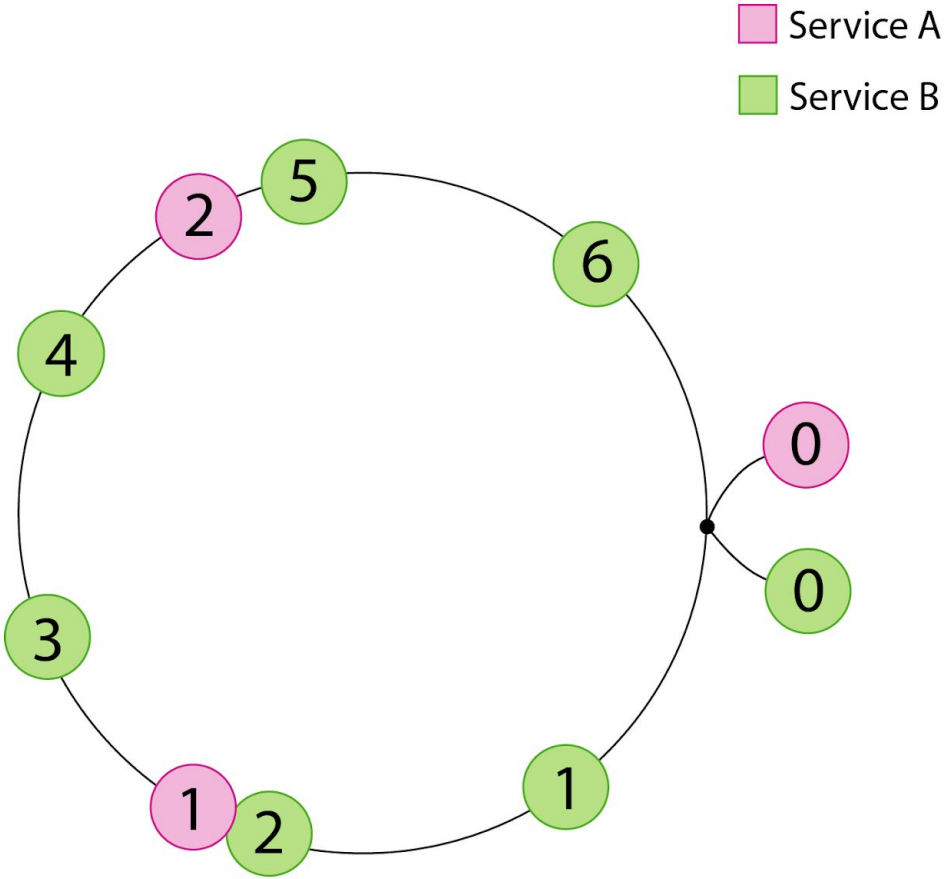
How can we improve aperture?

DISCRETE COORDINATES



COMPOSITE RINGS

0: [0, 1, 2, 3, 4, 5, 6]
1: [3, 4, 5, 6, 0, 1, 2]
2: [5, 6, 0, 1, 2, 3, 4]



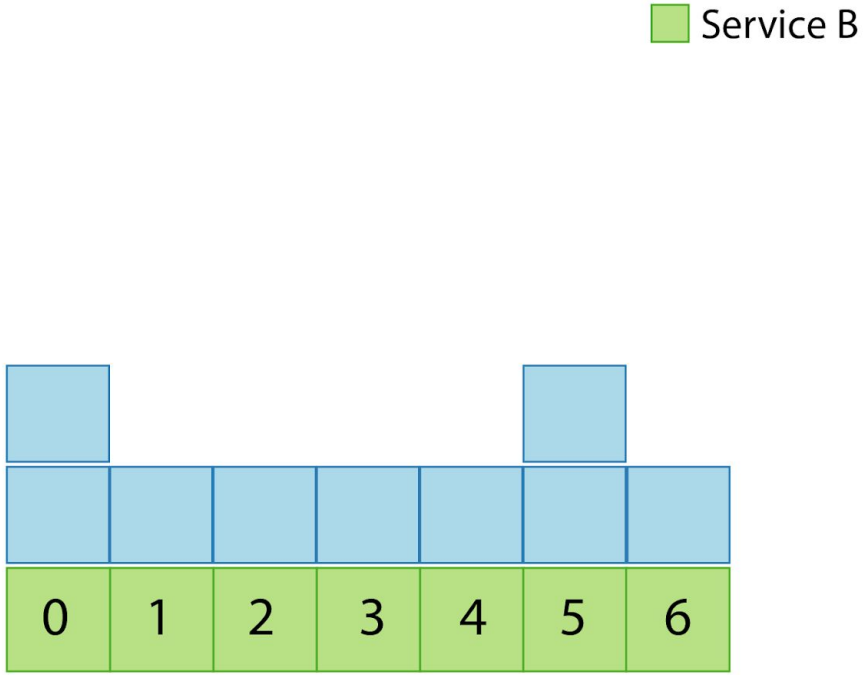
SESSION HISTOGRAM

Service A

0: [0, 1, 2, 3, 4, 5, 6]

1: [3, 4, 5, 6, 0, 1, 2]

2: [5, 6, 0, 1, 2, 3, 4]



MULTIPLE SERVICE RINGS

Service A

0: [0, 1, 2, 3, 4, 5, 6]

1: [3, 4, 5, 6, 0, 1, 2]

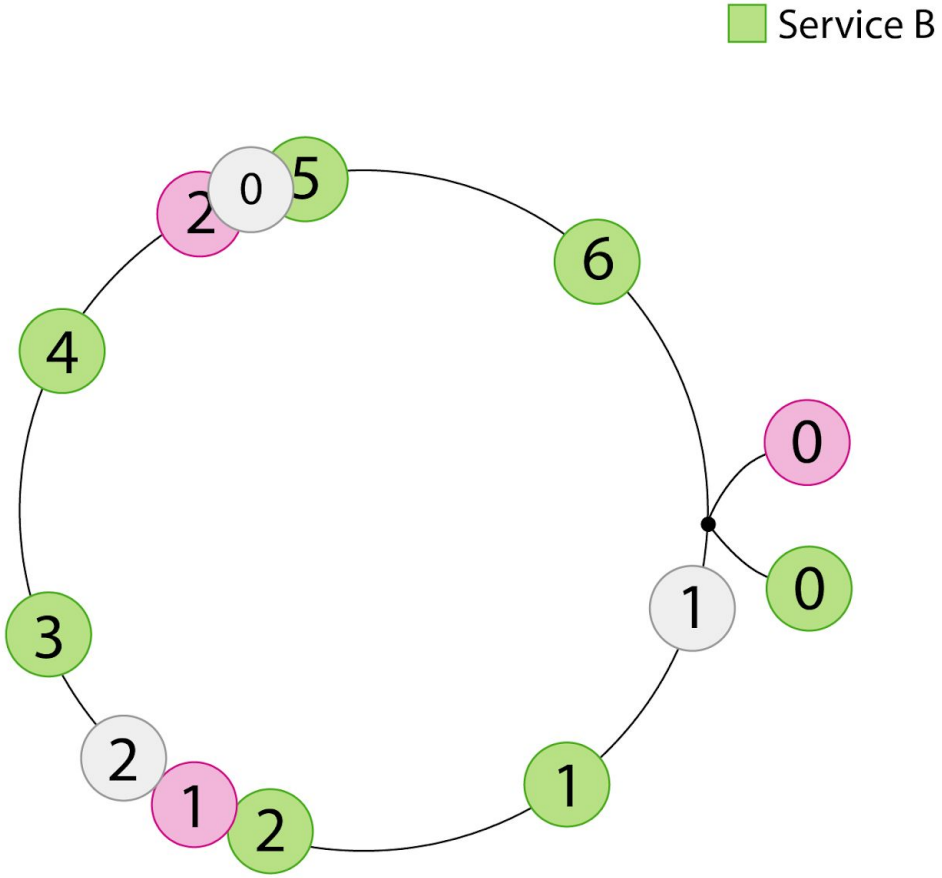
2: [5, 6, 0, 1, 2, 3, 4]

Service C

0: [5, 6, 0, 1, 2, 3, 4]

1: [1, 2, 3, 4, 5, 6, 0]

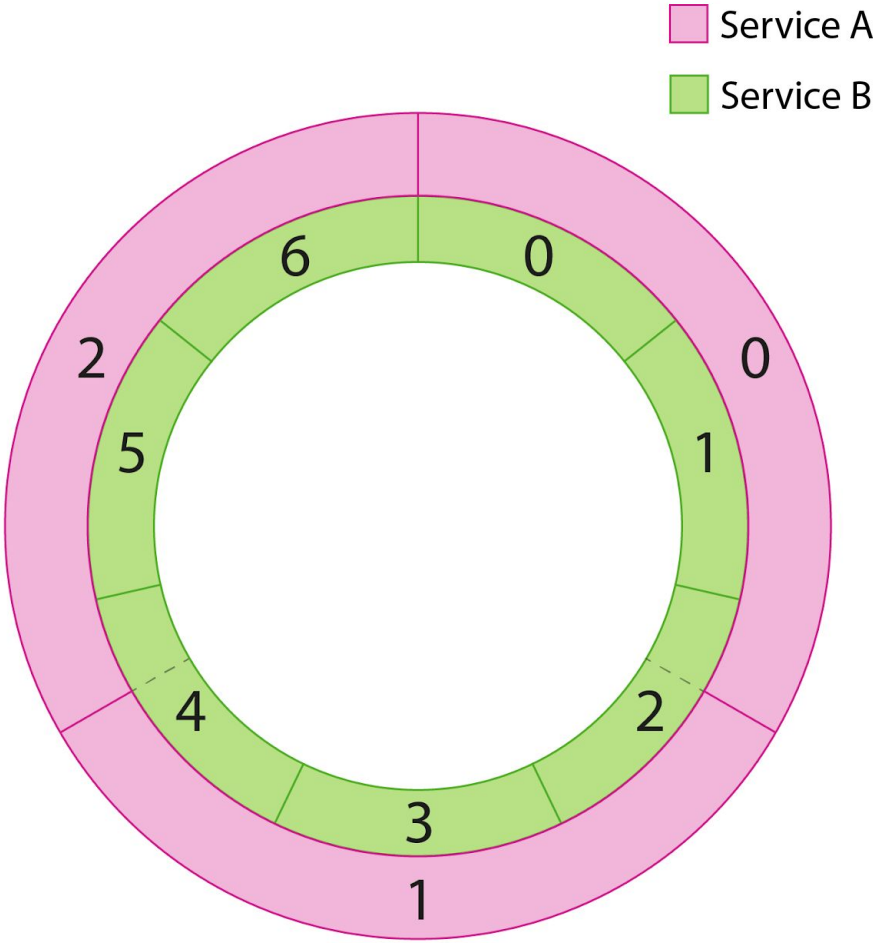
2: [3, 4, 5, 6, 0, 1, 2]



CONTINUOUS COORDINATES

Services fully occupy the same domain.

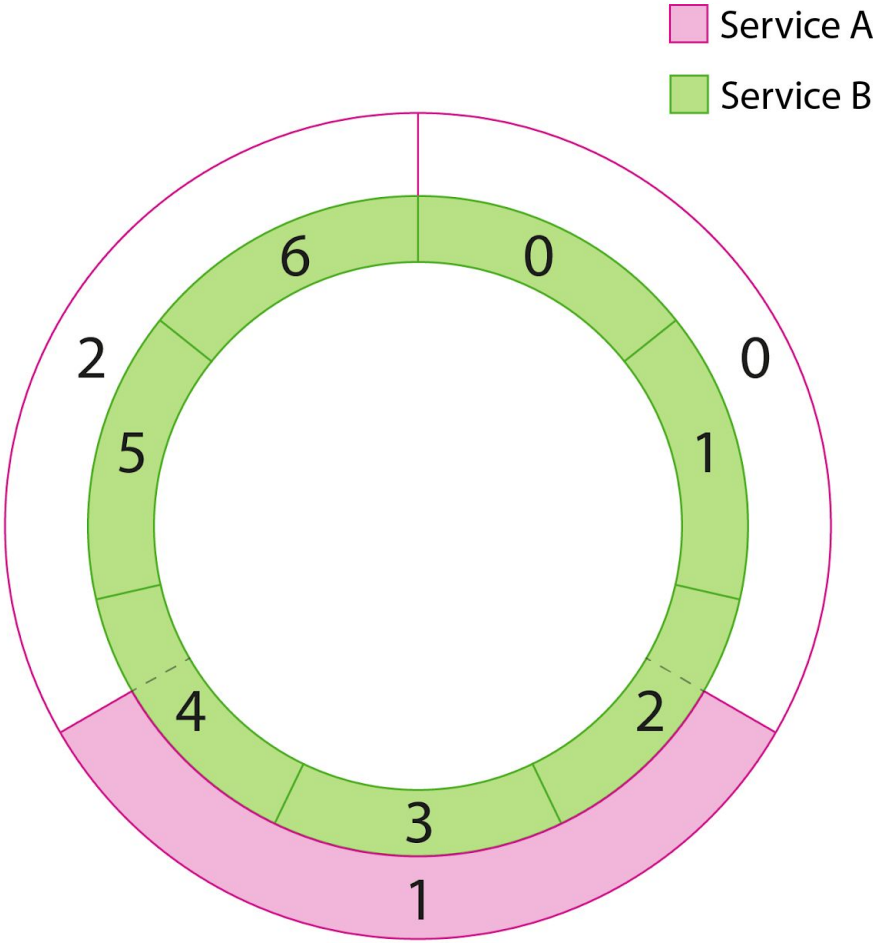
Load balancers can map from their respective range to discrete destinations.



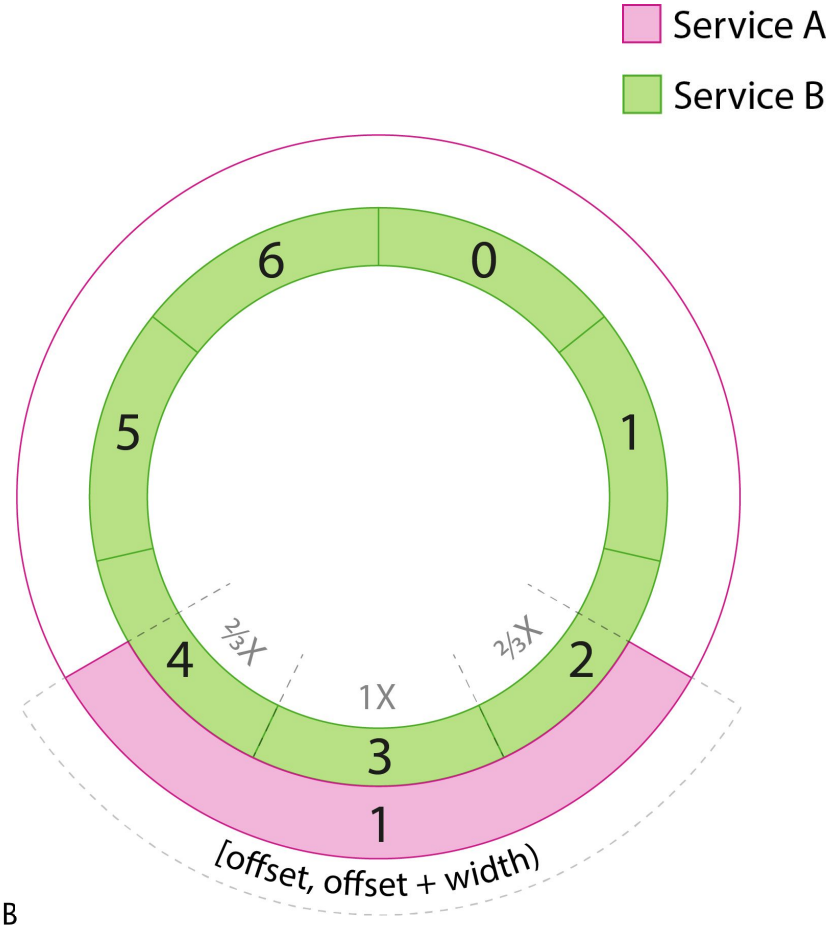
P2C + FRACTIONAL LOAD

Each load balancer picks two coordinates randomly within its range and maps them to discrete destinations.

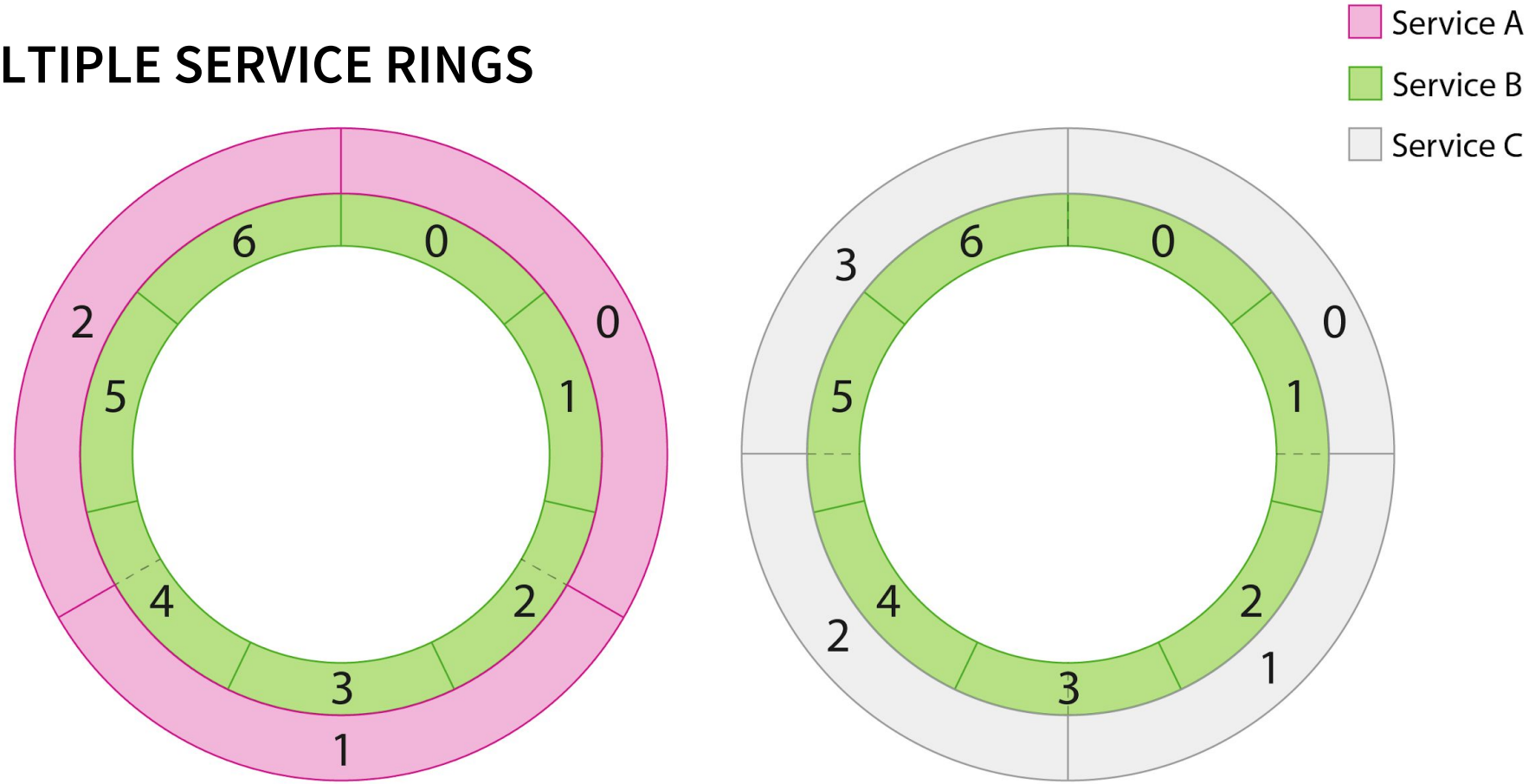
This inherently respects the fractional boundary conditions.



```
// compute the offset and width of this balancer.
val offset = coord.offset
val width = apertureWidth

// select two coordinates, randomly and uniformly,
// within our range [offset, offset + width) and map
// them to the destination ring.
val (a, b) = destRing.pick2(offset, width)

val nodeA = vector(a)
val nodeB = vector(b)

val aStatus = nodeA.status
val bStatus = nodeB.status
if (aStatus == bStatus) {
  // what proportion of a and b, respectively,
  // fall within [offset, offset + width)?
  val aw = destRing.weight(a, offset, width)
  val bw = destRing.weight(b, offset, width)
  // weight the load w.r.t to the ring proportions
  // to avoid biasing towards the node picked less often.
  if (nodeA.load / aw <= nodeB.load / bw) nodeA else nodeB
} else {
  if (Status.best(aStatus, bStatus) == aStatus) nodeA else nodeB
}
```



MULTIPLE SERVICE RINGS



CONTINUOUS COORDINATE MODEL

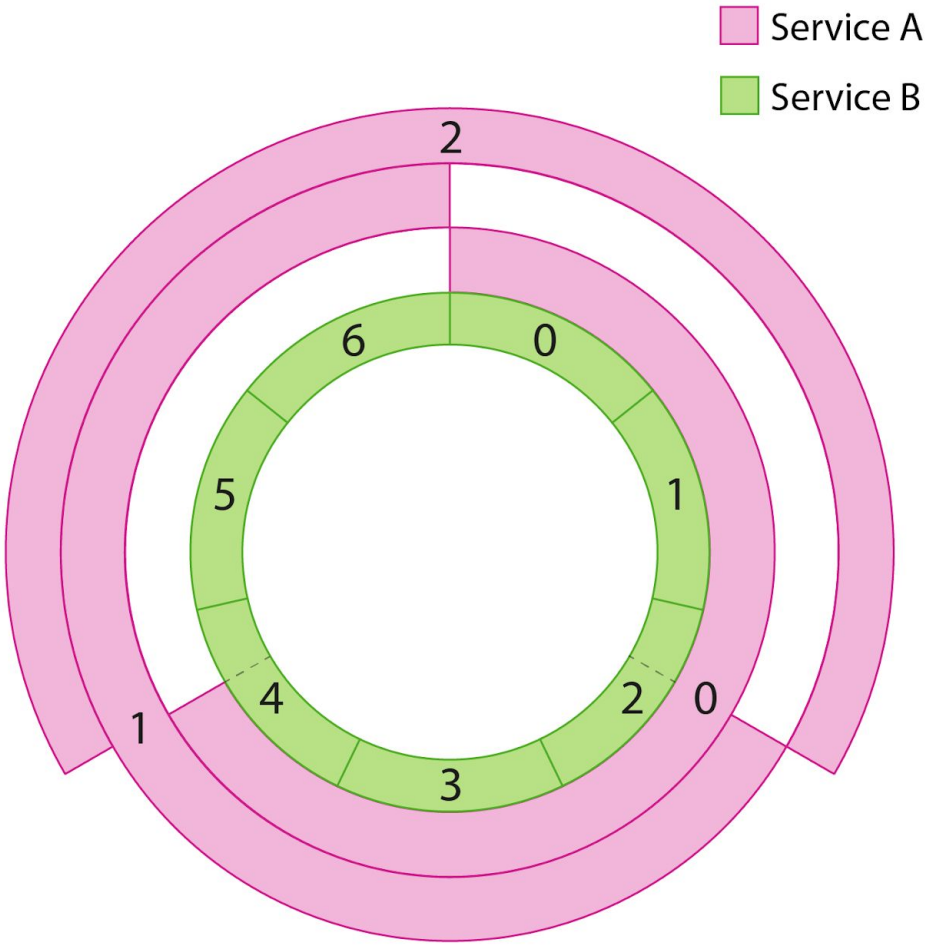
fair request distribution
with distinct services talking to the same destination ring

distributed
light coordination around metadata to construct rings

fewer sessions
aperture size naturally falls out of representation

DYNAMIC APERTURE SIZE

The aperture can grow/shrink so long as the peer ring completes whole rotations around destination ring.



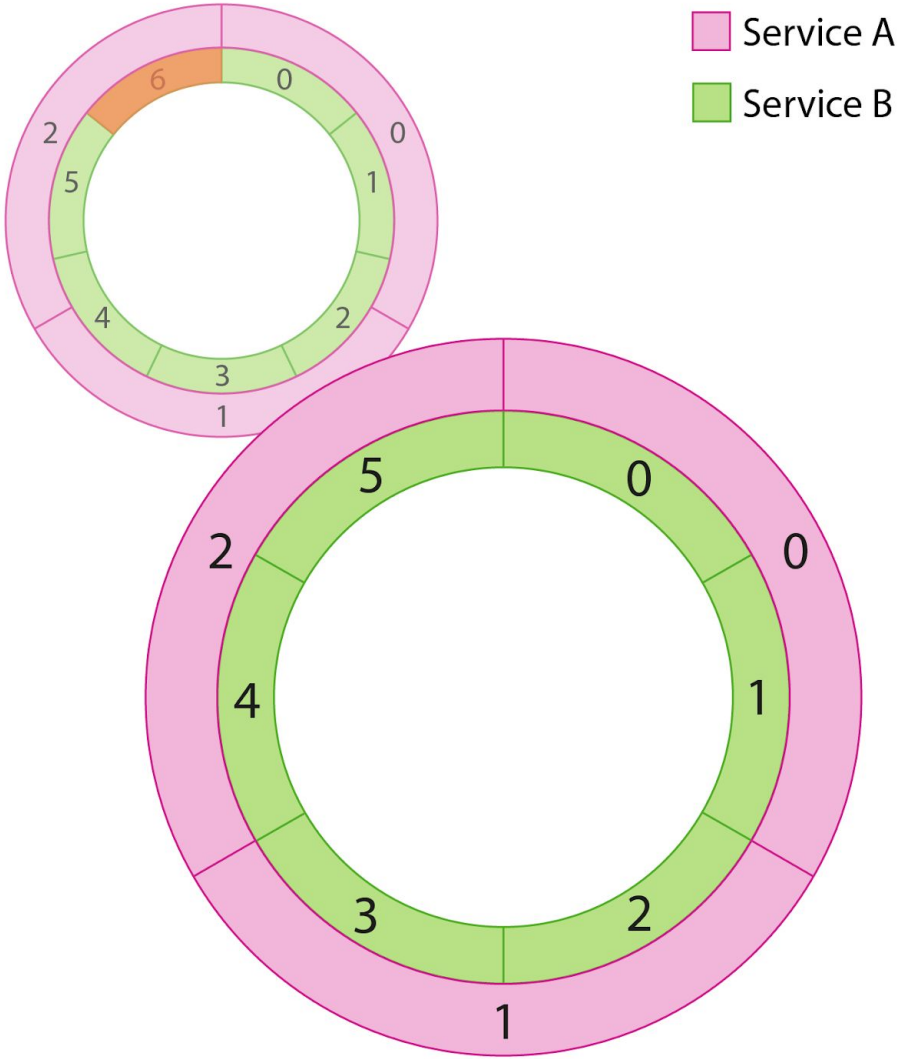
RESILIENCY

minimal disruption

entropy

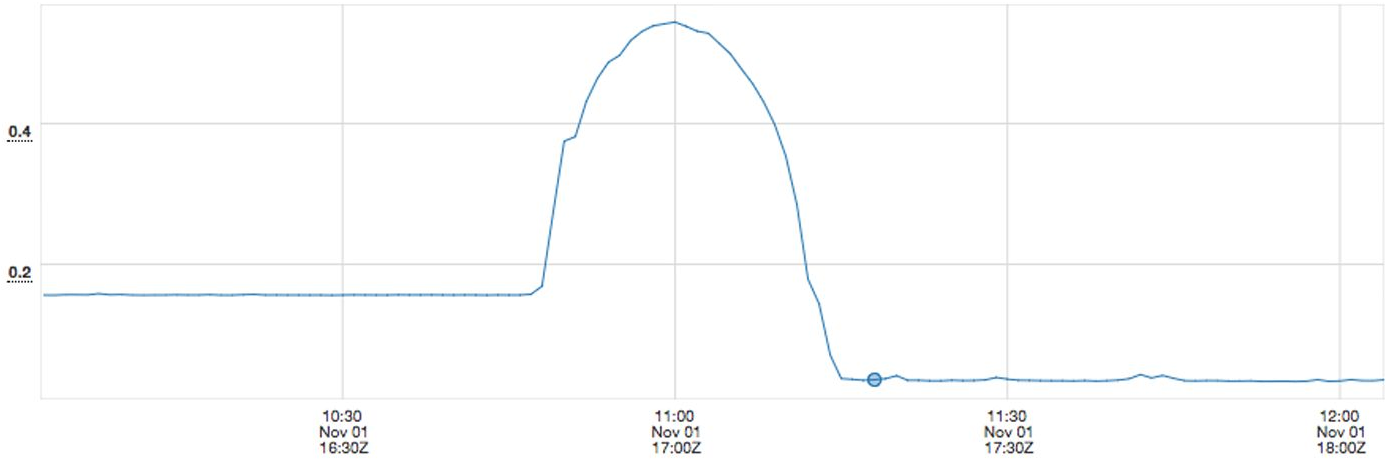
coalesced updates

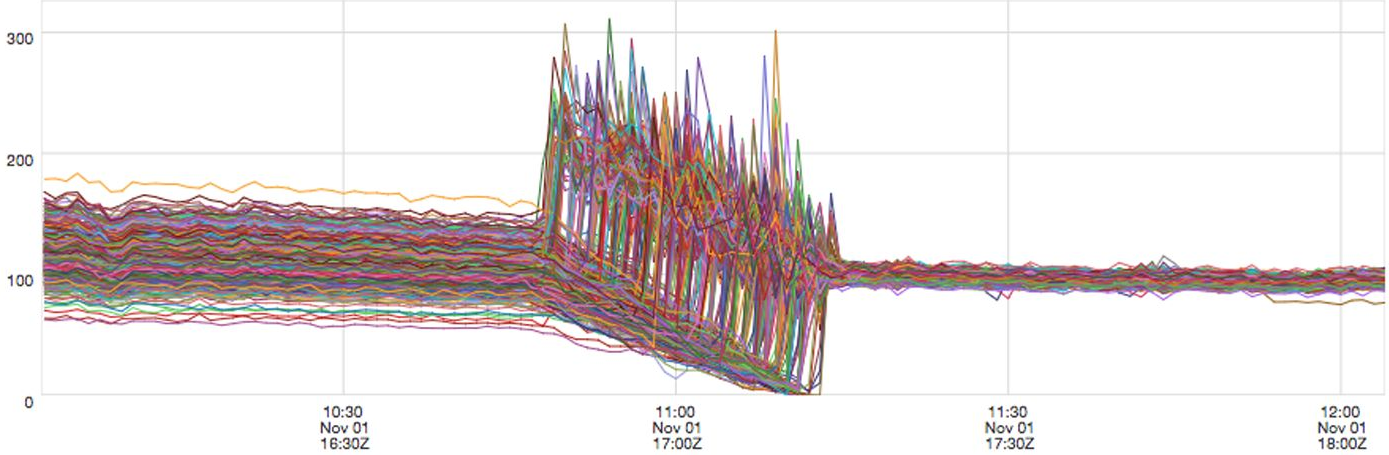
peer size heuristics



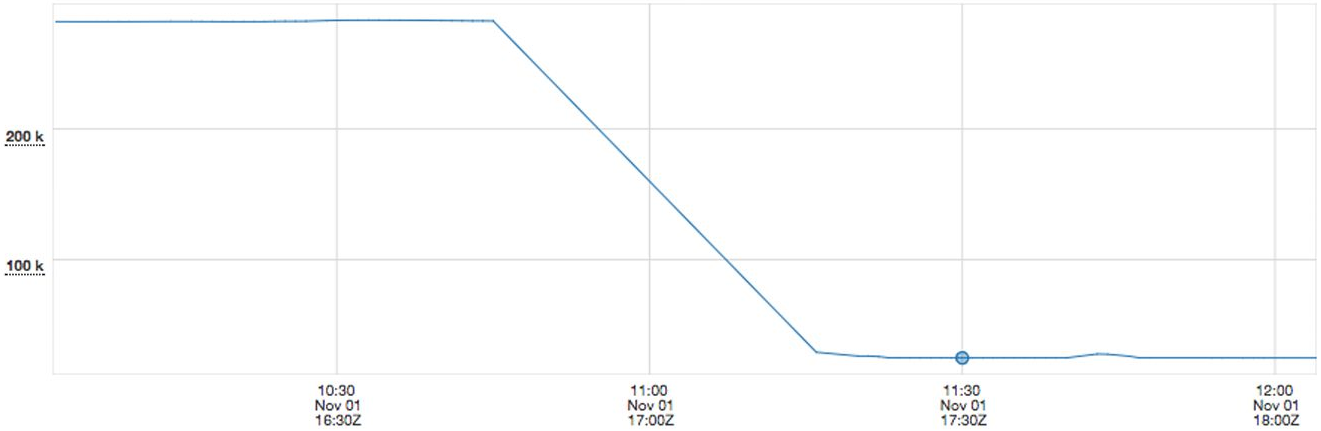
Production Results

78% reduction in standard deviation for requests per second.





91% drop in aggregate connections (~280k to ~25k)



SECOND-ORDER RESULTS

75% fewer failures
~20% reduction in latency at 99.9th percentile
20-25% less CPU used
Total garbage collection (GC) time cut in half

Attribution

Bryce Anderson, Billy Becker, Marius Eriksen, Daniel Furse, Steve Gury, Eugene Ma, Nick Matheson, Moses Nakamura, Kevin Oliver, Brian Rutkin, Daniel Schobel

Code

github.com/twitter/finagle

Questions?