Towards a Leaner Geo-distributed Cloud Infrastructure

Iyswarya Narayanan,
Anand Sivasubramaniam,
Bhuvan Urgaonkar

PENN STATE

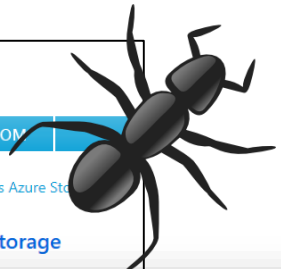
Aman Kansal,
Sriram Govindan

Microsoft

Microsoft

Geo-distribution

- Essential for
  - Low latency
  - High availability
- **What are the capacity implications?**



Windows Azure

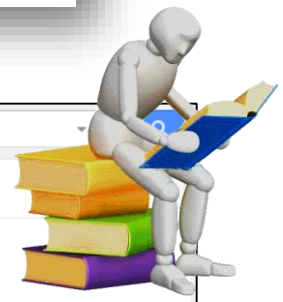
HOME | BLOGS + | VIDEOS | WINDOWSAZURE.COM

MSDN Blogs > Windows Azure > Details of the December 28th, 2012 Windows Azure St
Disruption in US South

Details of the December 28th, 2012 Windows Azure Storage

resulted in the Fabric Controller hitting a bug that

On December 28th, 2012 there was a service interruption that affected 1.8% of the Windows Azure Storage accounts. The affected storage accounts were in one storage stamp (cluster) in the U.S. South region. We apologize for the disruption and any issues it caused affected



Google | Search for topics

Groups

Google App Engine ›

Post-mortem for February 24th, 2010 outage

15 posts by 13 authors

However, some documentation detailing the procedure to support the datastore during failover incorrectly referred to the old configuration. This led to confusion during the event.

are taking to mitigate the impact of future outages like this one in the future.

On February 24th, 2010, all Googe App Engine applications were in

amazon web service

Sign Up | My Account / Console | English

AWS Products & Solutions | AWS Product Information | Developers | Support

Summary of the October 22,2012 AWS Service Event in the US-East Region

We'd like to share more about the service event that occurred on Monday, October 22nd in the US- East Region. We have now completed the

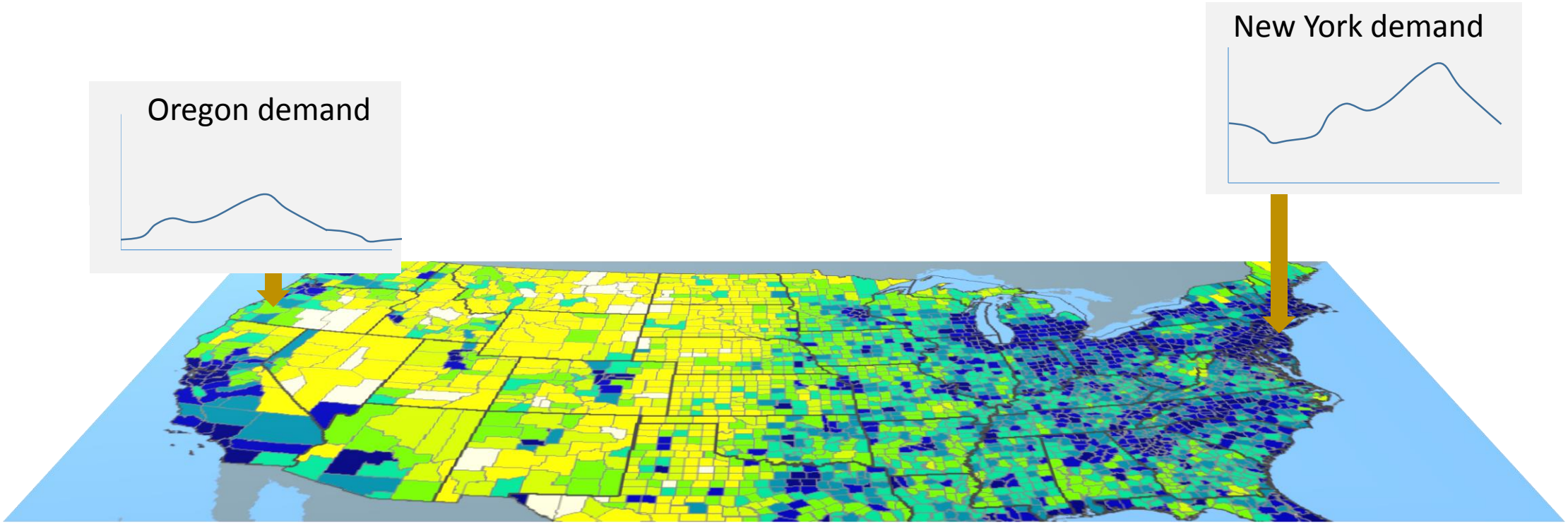
problem was a latent bug in an operational data collection agent that runs on the EBS st

Compute Cloud (EC2)

At 10:00AM PDT Monday, a small number of Amazon Elastic Block Store (EBS) volumes in one of our five Availability Zones in the US-East Region began seeing degraded performance, and in some cases, became "stuck" (i.e. unable to process further I/O requests). The root cause of the problem was a latent bug in an operational data collection agent that runs on the EBS storage servers. Each EBS storage server has an agent that contacts a set of data collection servers and reports information that is used for fleet maintenance. The data collected with this system is important, but the collection is not time- sensitive and the system is designed to be tolerant of late or missing data. Last week, one

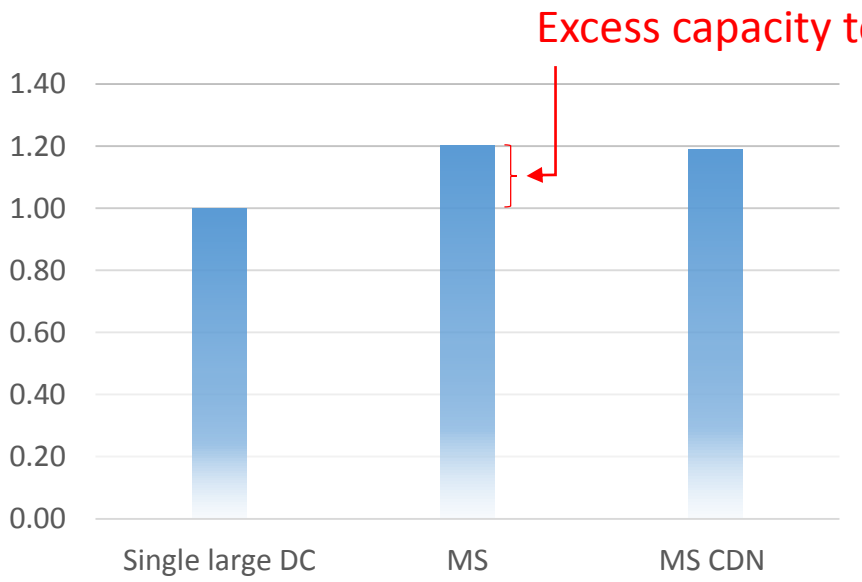
Model: Client Demand

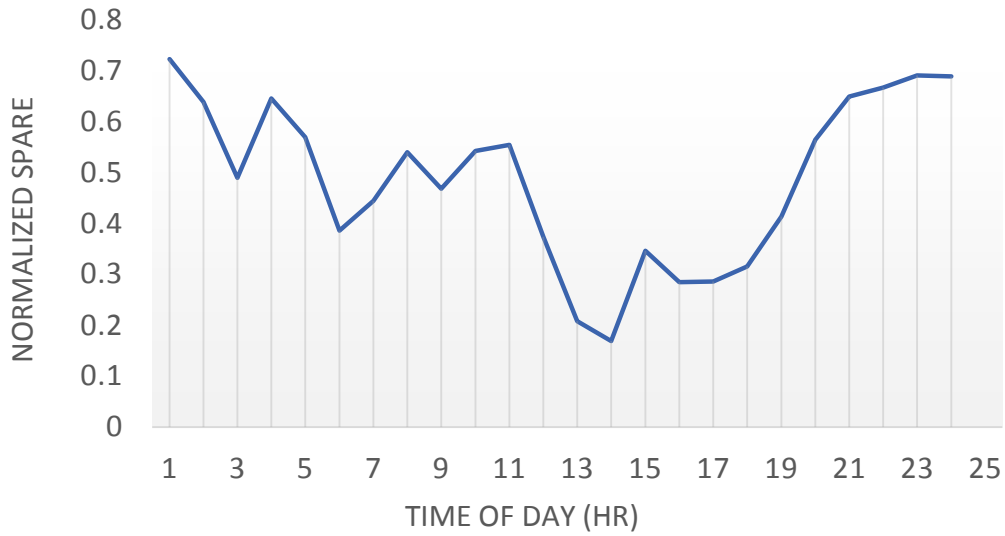
- Proportional to population of location
- Varies with time of day
  - Each location has its own time zone



Capacity Implications of Latency

- Geo-distributed => closer to clients
- Each data center must meet its local peak

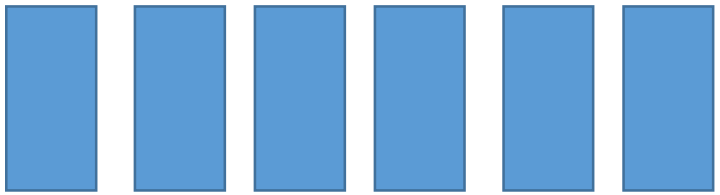




Capacity Implications of Availability

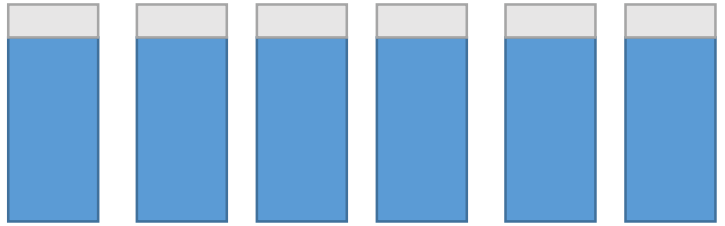
- Assuming uniform capacity, with 6 data centers:

Each DC is 16.6% of total.



Capacity without spare for availability

Total spare is 20% of initial total
(since spare at failed DC unavailable)



Capacity with spare for availability

Question

What is the total capacity required and how much of it is placed at each data center

- To meet latency constraints, AND
- To allow for one DC to fail (at any time, including at peak)

(Server capacity considered, state replication assumed taken care already for geo-distributed operation even without failure.)

Optimization Formulation

- Pre-failure latency (L): same as that achieved by any client if routing to nearest DC
  - 100-th percentile of nearest DC mapping, could use other percentiles
- Post-failure latency (L'): L' >=L
- Assume: service-time same at each DC (nearest or otherwise)
  - Operating below queue build-up point. Capacity vs. queuing delay trade-off not considered
- Capacity at DC-j is c_j
- Fraction of load from client location "*i*" directed to DC "*j*" when DC "k" has failed is denoted f_{ijk}
  - K=0 corresponds to no DC failed

$$\min \sum_{j \in \{1,\ldots,n\}} c_j$$

Minimize the sum of DC capacities

Subject to:

$$\sum_{j: l_{ij} \le l} f_{ij0}(t) \ge d_i(t), \forall i, t.$$

$$\sum_{j: j \ne k, l_{ij} \le l'} f_{ijk}(t) \ge d_i(t), k > 0, \forall i, t.$$

Demand at *i*-th client served by DCs within latency limit (for all *i, t*). Possibly different latency before and after failure.

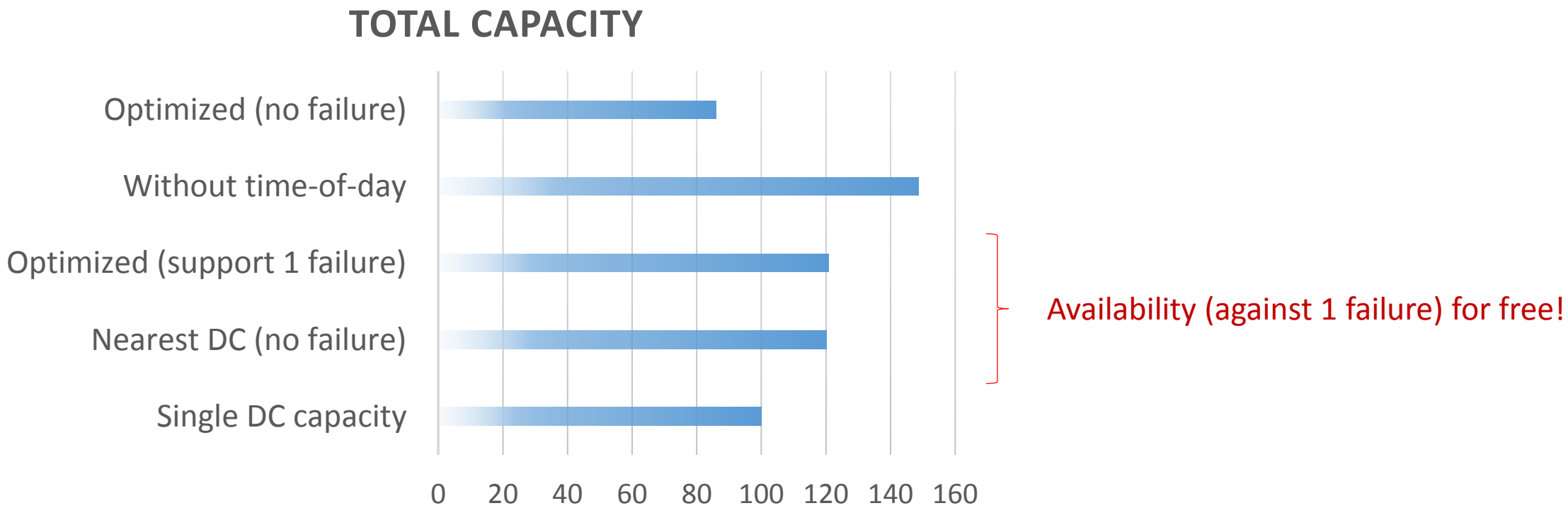
$$c_j \ge \sum_i f_{ij0}(t) \quad \forall j, t,$$

$$c_j \ge \sum_i f_{ijk}(t) \quad k > 0, k \ne j, \forall j, t.$$

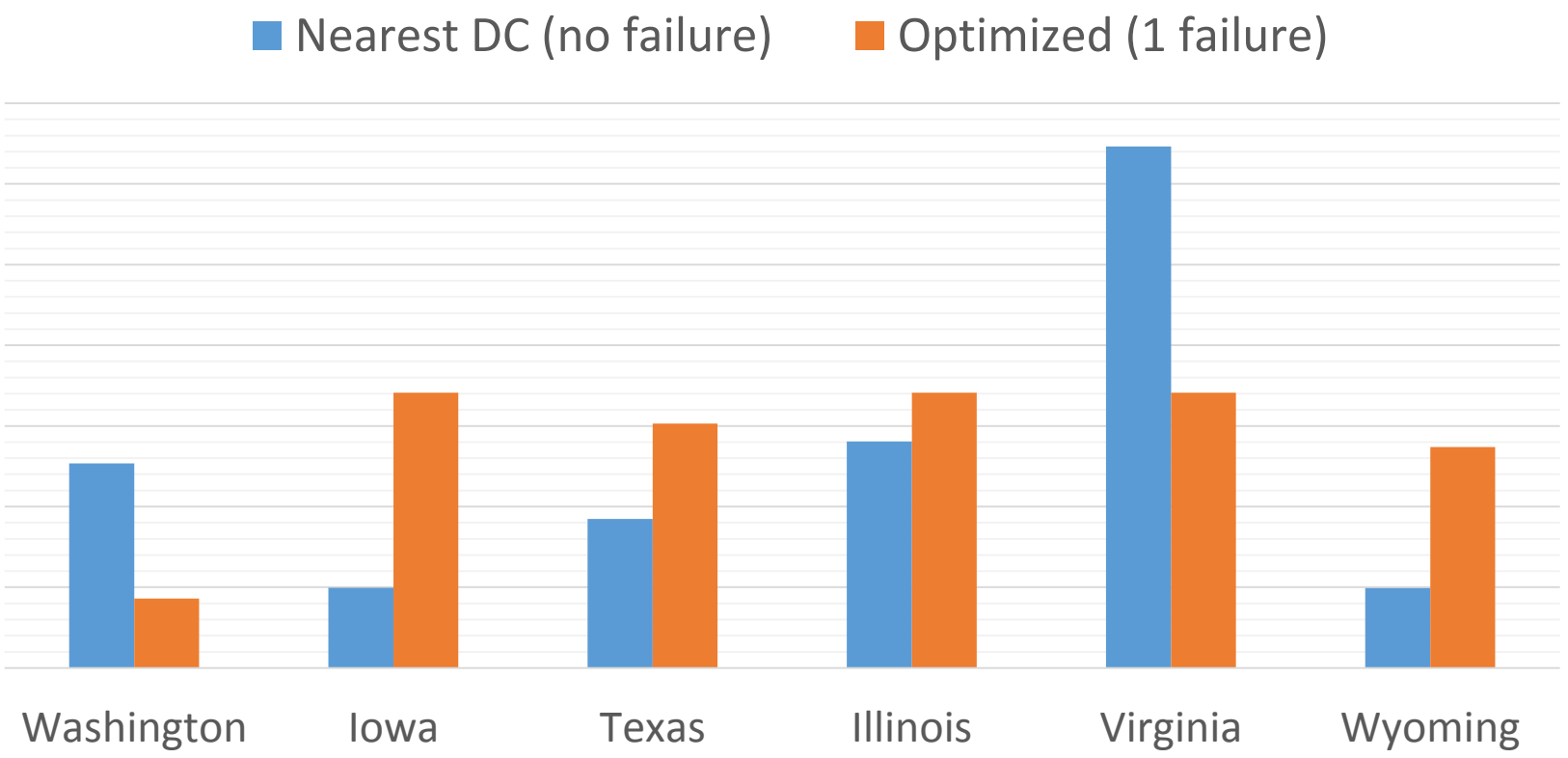
Capacity at *j*-th DC is the maximum of capacities needed at different time slots, including after failure.

Results

Experiments using demand measured for one Microsoft cluster, and 6 of MS DC locations within US.
L'= L

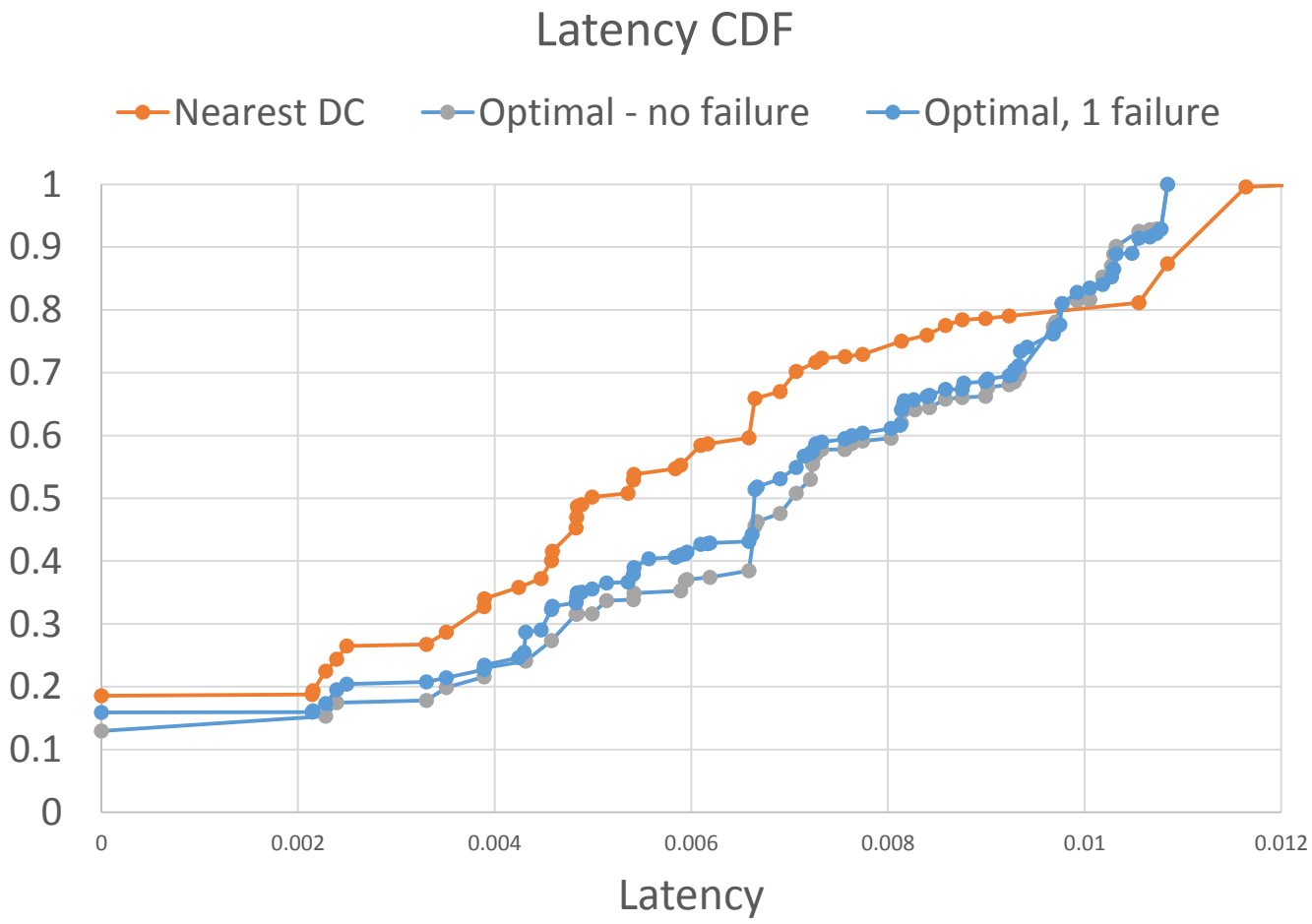


The largest DC capacity is split differently after optimization.



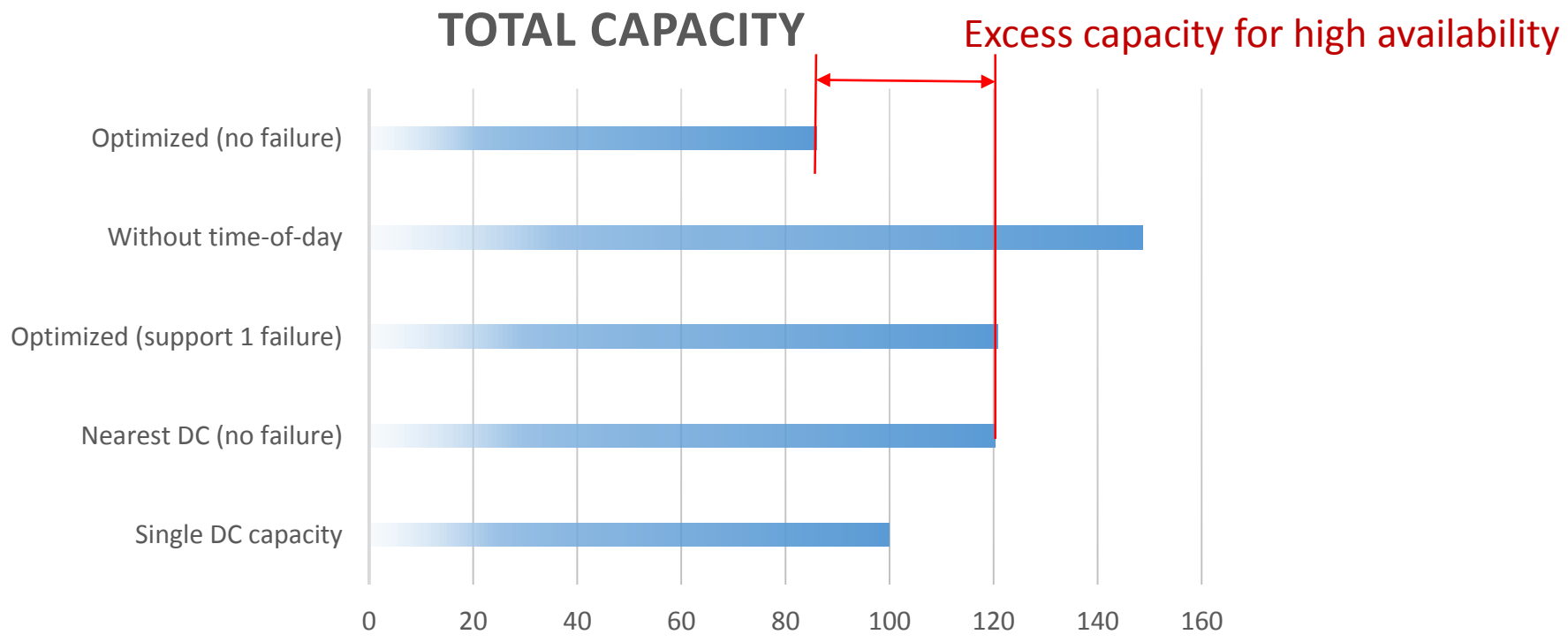
Latency Caveat

- Even though we enforced a latency bound, the latency distribution changes
  - Since clients can be allocated anywhere within latency limit, they may not be allocated to “nearest” DC



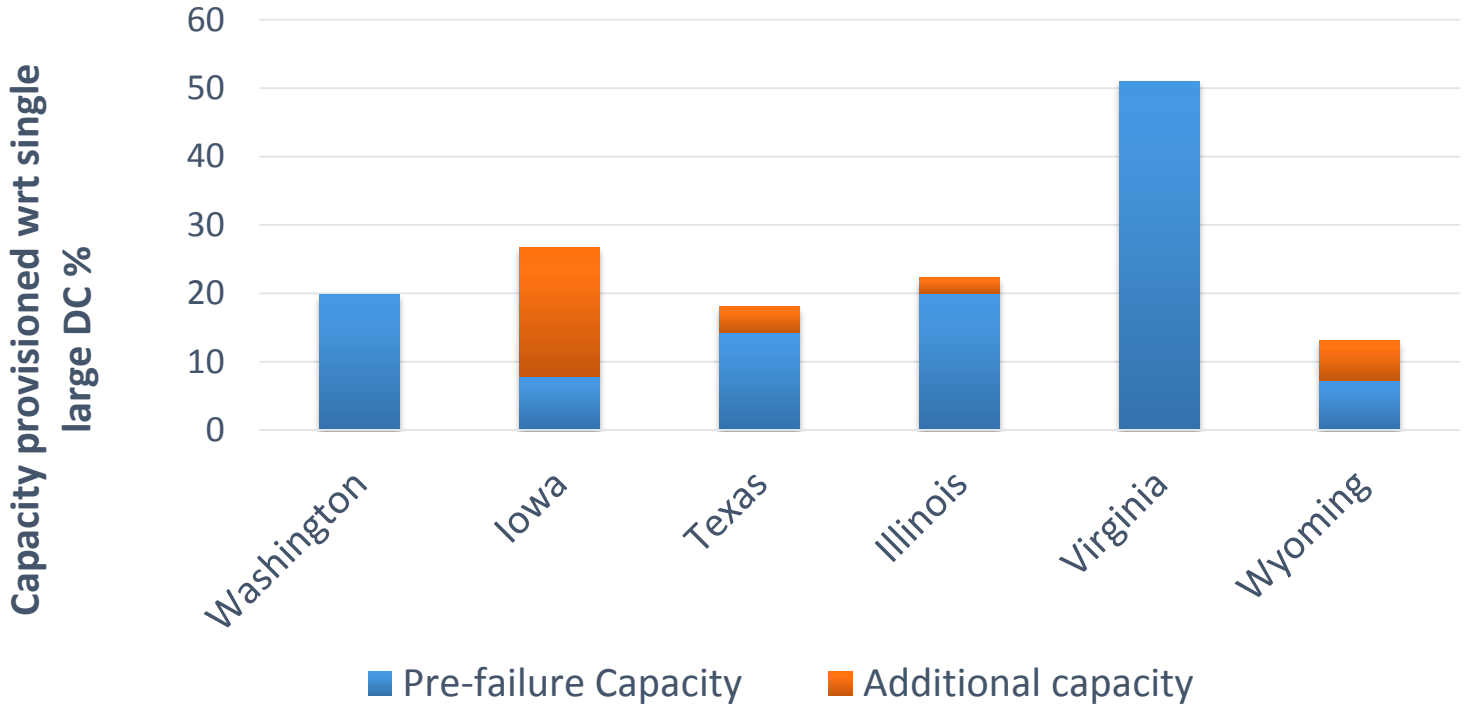
Results

Experiments using demand measured for one Microsoft cluster, and 6 of MS DC locations within US.
L'= L



Use nearest DC before failure

- Before failure, nearest DC
  - After failure, select backup within a latency bound.
  - Best possible latency before failure

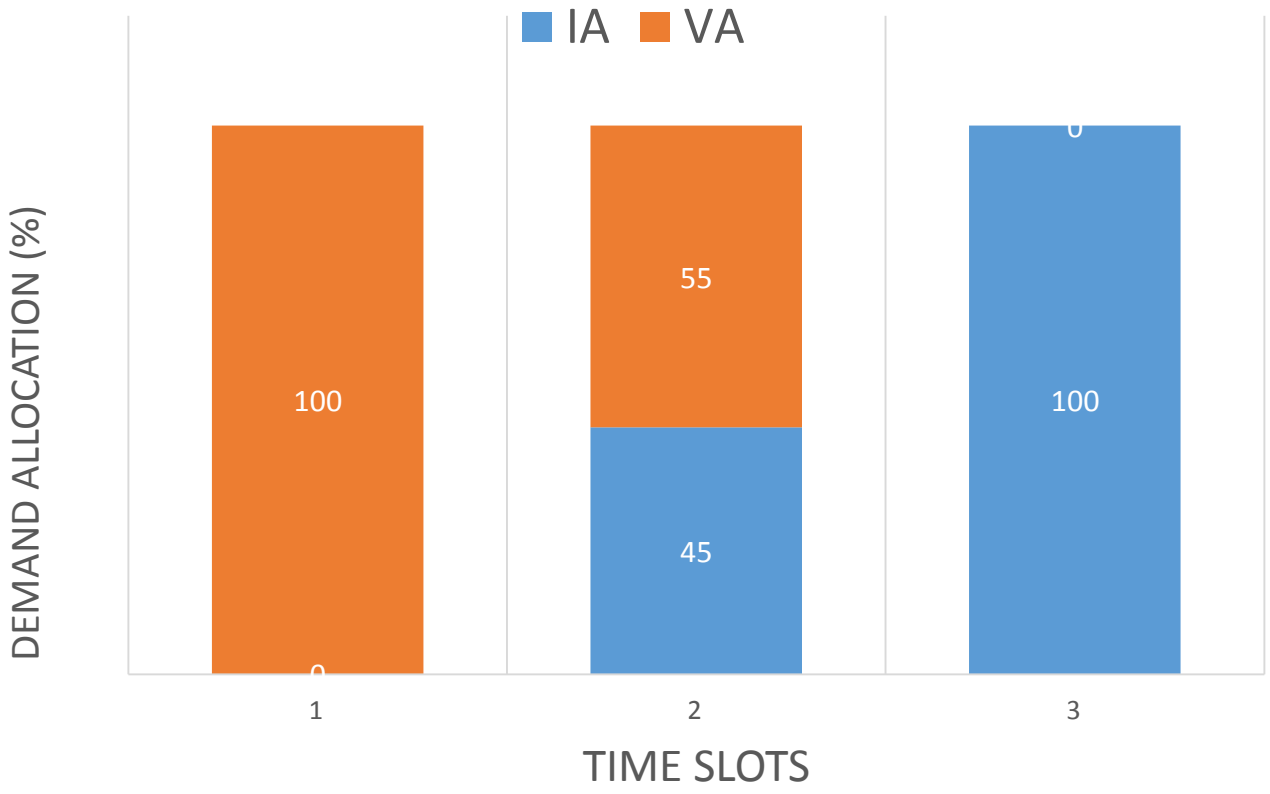


25.125% extra capacity needed (compared to nearest DC without failure)

Design Implications: Routing

- Geo-load balancing
  - No longer routing to nearest DC
  - After failure, not all traffic from failed DC routed to same backup
  - Client regions served by non-failed DCs also affected
- Load balancer
  - Be aware of app layer capacity allocation or app layer response time?
    - Unlike anycast, cannot route by network layer latency

E.g. Penn. clients allocation to DC changes with time of day among Iowa and Virginia DCs



Design Implications: State Replication

- Replicas be placed where clients will be routed to after failure
- Option 1: Replicate at nearest DC
  - Low latency overhead for strong consistency
  - May have to migrate data after failure, nearest-DC may not have network bandwidth when taking on failed DC's traffic
- Option 2: Replicate at the DC where client will be routed to
  - Higher latency penalty during non-failure operation for stronger consistency
  - Decision depends on client location, and may differ even for same client location
  - May change with time of day: multiple replicas?

Design Implications: Cloud API

- Existing clouds
  - Tenant apps can request capacity in any DC
  - Clouds do not guarantee capacity will be available
- For high availability
  - Arbitrary capacity at next-nearest DC after a DC failure is not going to be available
  - Availability should be part of cloud API:
    - infrastructure provisions spare capacity per app requests (lower price than active capacity, monetized on the spot market)
    - app expects capacity at right locations

Related Works

- Capacity planning, facility location [Goiri et al ICDCS'11, others]
  - Showed that more DCs, where each is lower availability (lower cost) but extra geo-spares, better
  - Computed optimal capacity placements
  - Did not account for time of day variations, compare to nearest DC routing
- Server power management [Mathew et al, Infocom'12]
  - Turns servers off to adapt capacity across multiple DCs within latency bound to current demand
  - Do not compute actual capacity to be built
- Prior works require hard numbers for availability (e.g. 5-9s for power)
  - Not available for software issues, router misconfiguration outside of cloud operator's network, operator errors, non-power equipment failures

Conclusions

Geo-distribution affects capacity planning

Jointly optimize **availability** and **latency** to reduce capacity

- Actual spare depends on latency objectives before and after failures
- High-availability for “free” compared to naively routing to nearest DC

Software design may change due to geo-capacity optimization

- Request routing, state replication, cloud API, auto-scaling impacted