volley: automated data placement for geo-distributed cloud services

sharad agarwal, john dunagan, navendu jain, stefan saroiu, alec wolman, harbinder bhogan

very rapid pace of datacenter rollout

- April 2007
  - Microsoft opens DC in Quincy, WA
- September 2008
  - Microsoft opens DC in San Antonio, TX
- July 2009
  - Microsoft opens DC in Dublin, Ireland
- July 2009
  - Microsoft opens DC in Chicago, IL

geo-distribution is here

- major cloud providers have tens of DCs today that are geographically dispersed
  - cloud service operators want to leverage multiple DCs to serve each user from best DC
- user wants lower latency
- cloud service operator wants to limit cost
  - two major sources of cost: inter-DC traffic and provisioned capacity in each DC
- if your service hosts dynamic data (e.g. frequently updated wall in social networking), and cost is a major concern
  - partitioning data across DCs is attractive because you don't consume inter-DC WAN traffic for replication

research contribution

- major unmet challenge: automatically placing user data or other dynamic application state
  - considering both user latency and service operator cost, at cloud scale
- we show: can do a good job of reducing both user latency and operator cost
- our research contribution
  - define this problem
  - devise algorithm and implement system that outperforms heuristics we consider in our evaluation
- exciting challenge
  - scale: O(100million) data items
  - need practical solution that also addresses costs that operators face
  - important for multiple cloud services today; trends indicate many more services with dynamic data sharing
  - all the major cloud providers are building out geo-distributed infrastructure

sharad.agarwal@microsoft.com

overview

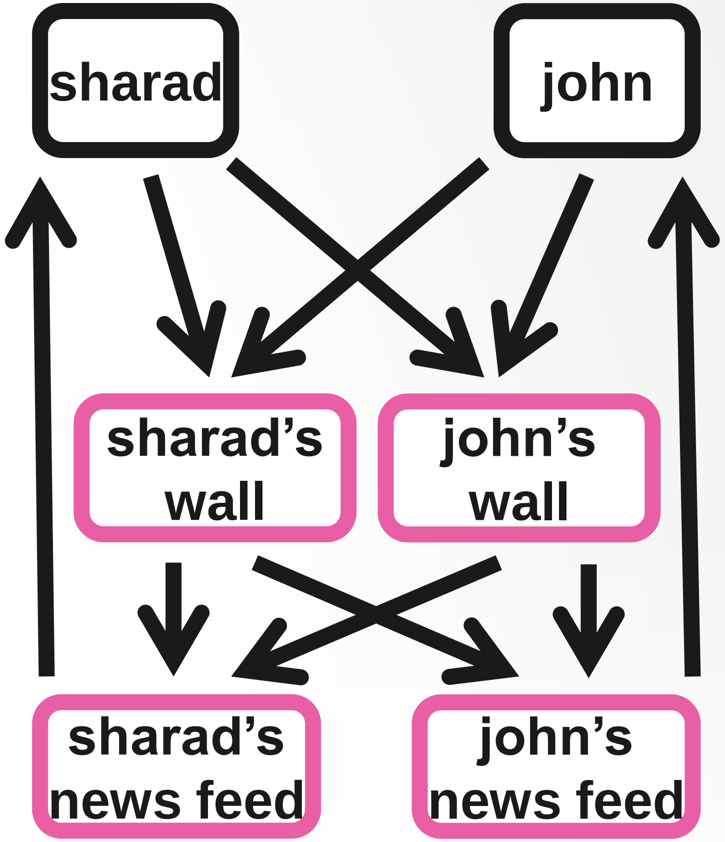
how do users share data?

volley

evaluation

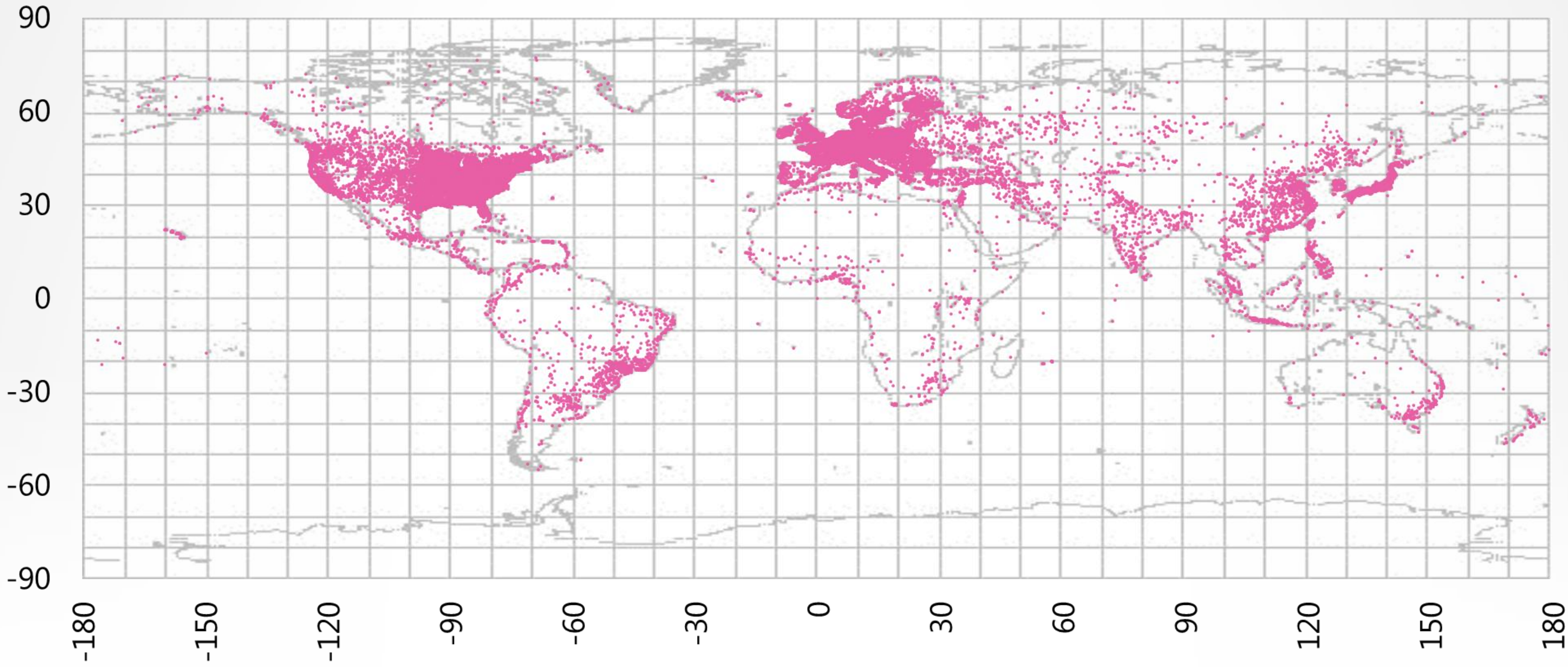
data sharing is common in cloud services

- many can be modeled as pub-sub
  - social networking
    - Facebook, LinkedIn, Twitter, Live Messenger
  - business productivity
    - MS Office Online, MS Sharepoint, Google Docs
- Live Messenger
  - instant messaging application
  - O(100 million) users
  - O(10 billion) conversations / month
- Live Mesh
  - cloud storage, file synchronization, file sharing, remote access



users scattered geographically (Live Messenger)

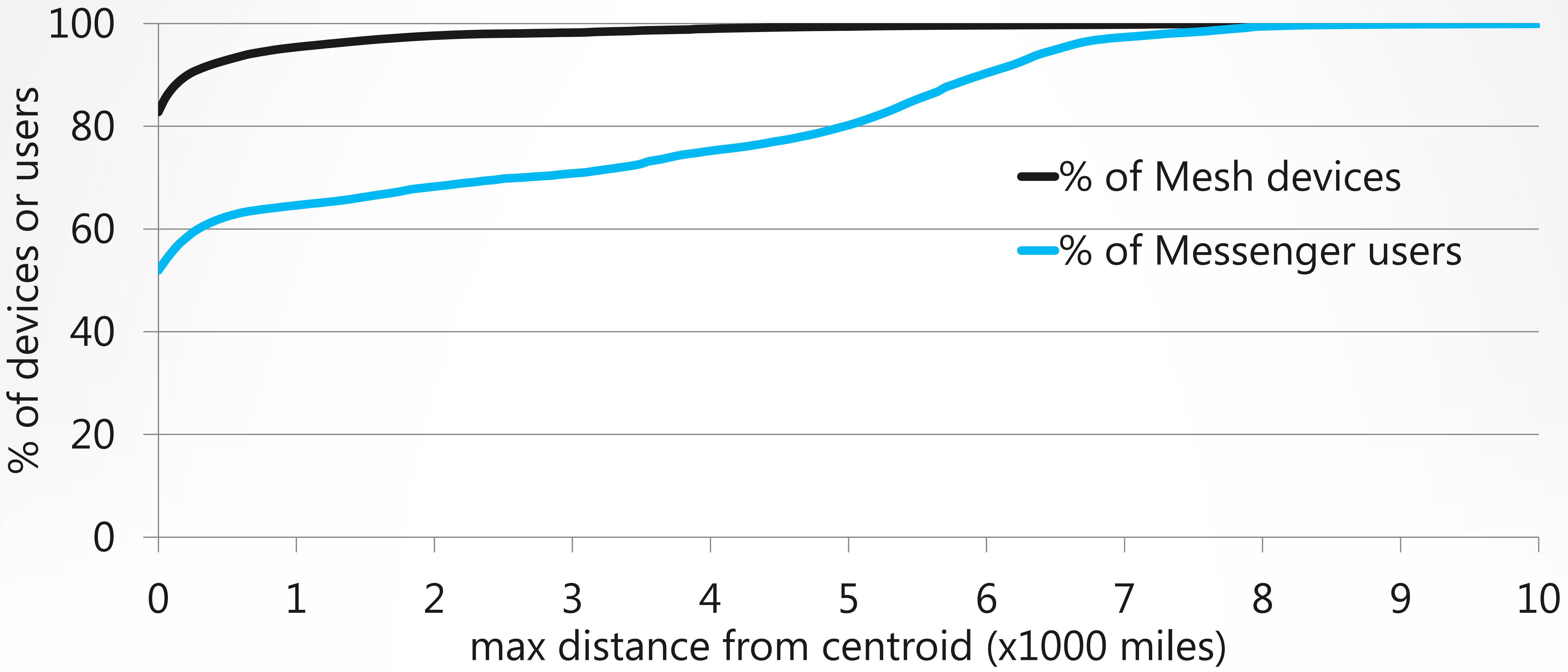
PLACING ALL DATA ITEMS IN ONE PLACE IS REALLY BAD FOR LATENCY



sharad.agarwal@microsoft.com

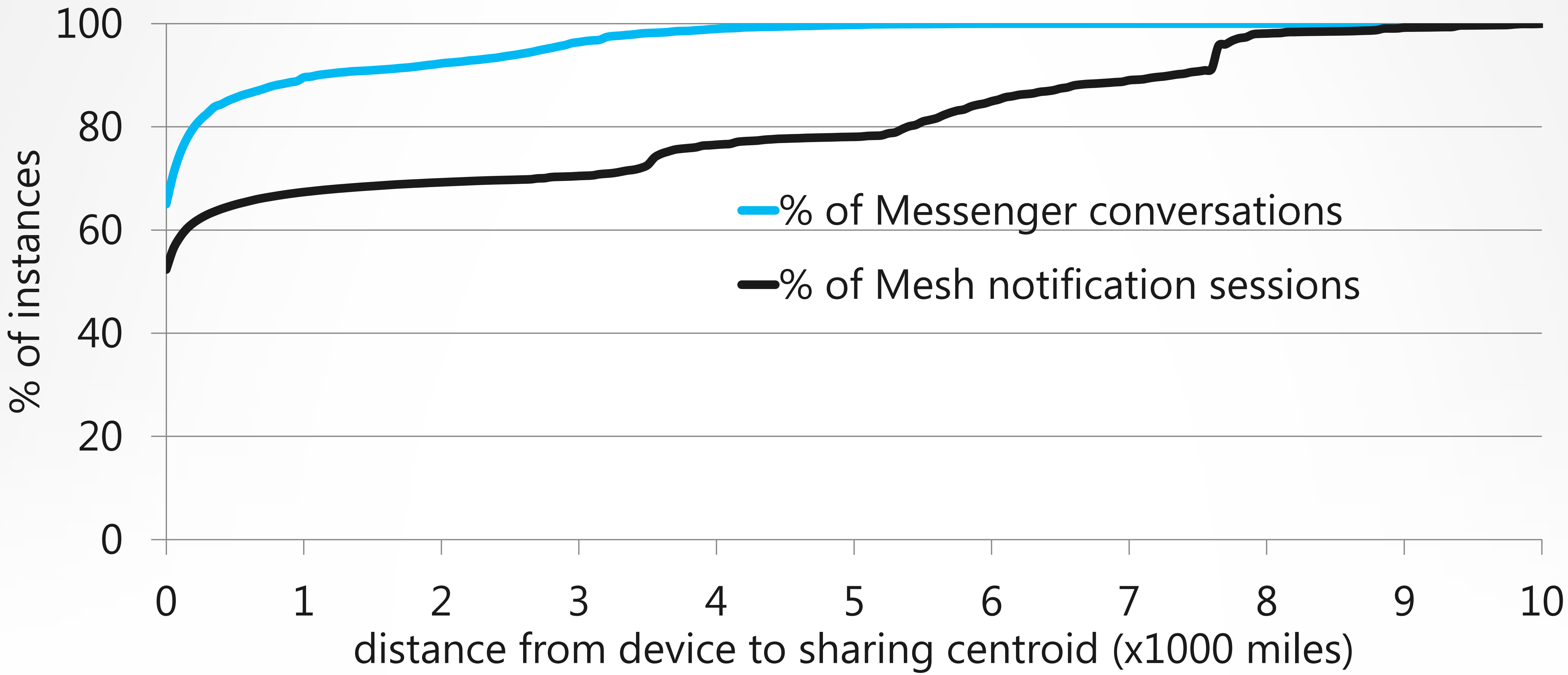
users travel

ALGORITHM NEEDS TO HANDLE USER LOCATIONS THAT CAN VARY



users share data across geographic distances

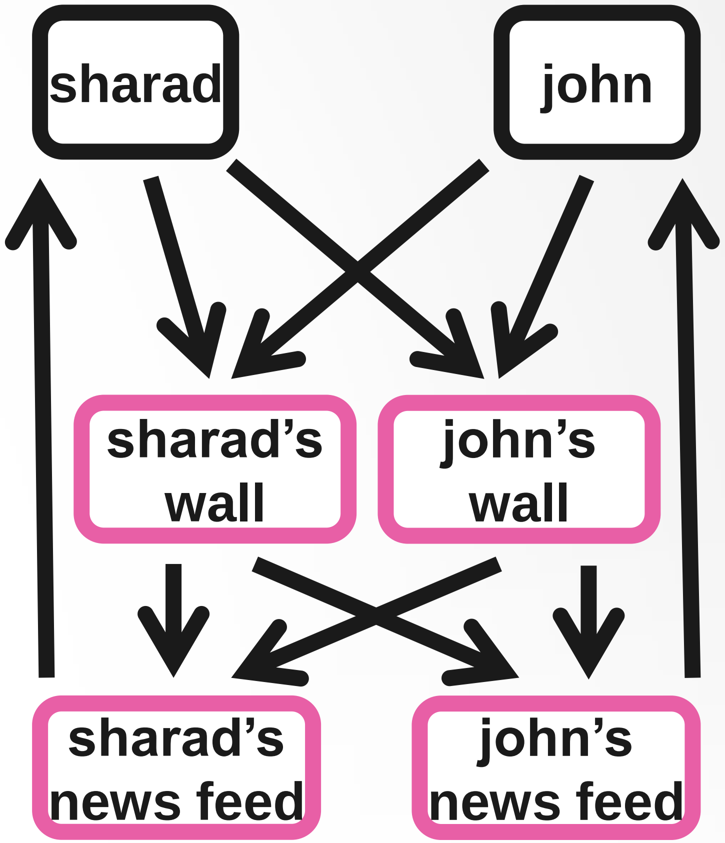
ALGORITHM NEEDS TO HANDLE DATA ITEMS THAT ARE ACCESSED AT SAME TIME BY USERS IN DIFFERENT LOCATIONS



sharad.agarwal@microsoft.com

sharing of data makes partitioning difficult

- data placement is challenging because
  - complex graph of data inter-dependencies
  - users scattered geographically
  - data sharing across large geographic distances
  - user behavior changes, travels or migrates
  - application evolves over time



overview

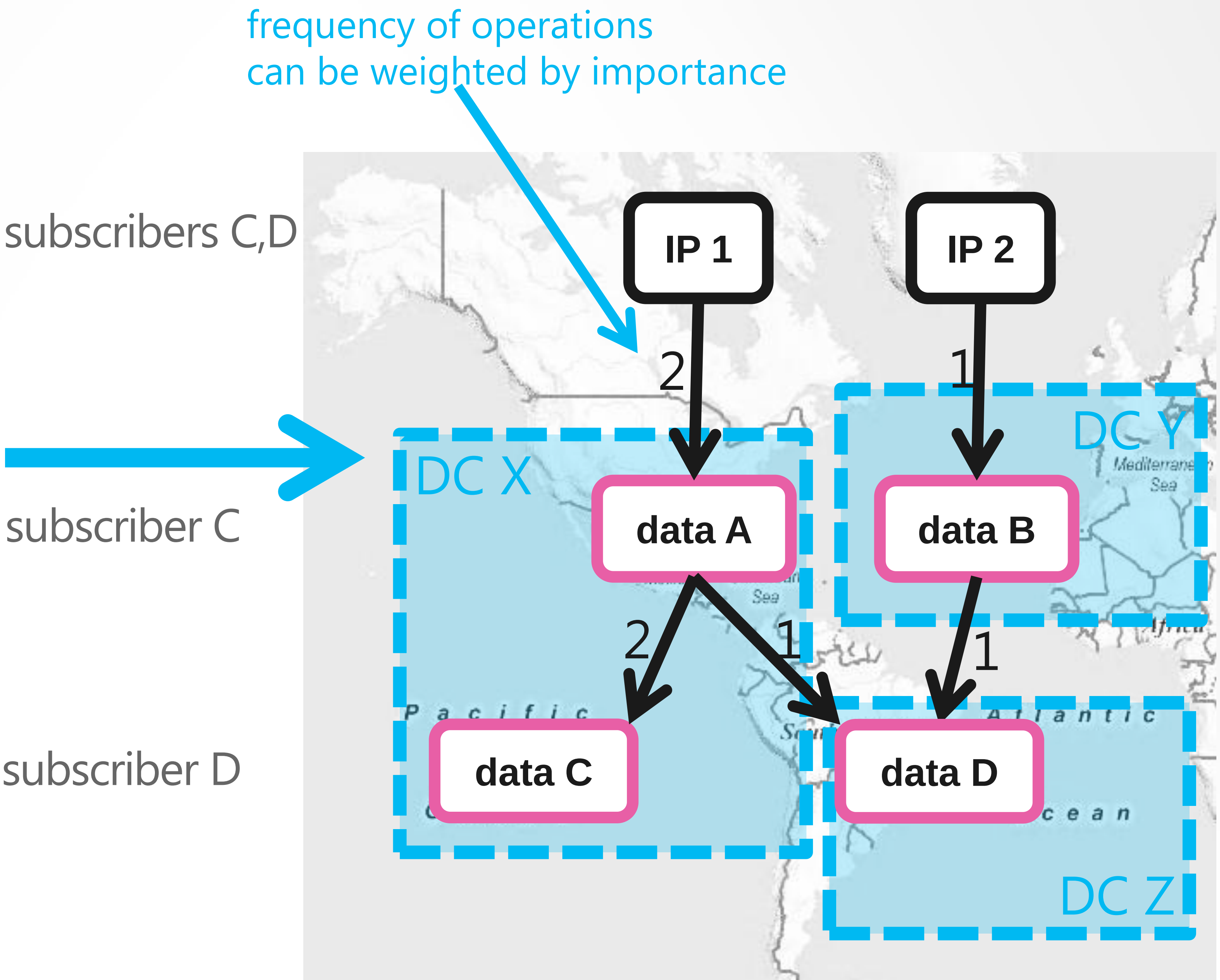
how do users share data?

volley

evaluation

simple example

- transaction$_1$:
  user updates wall A with two subscribers C,D
  - IP_1 → A
  - A → C
  - A → D
- transaction$_2$:
  user updates wall A with one subscriber C
  - IP_1 → A
  - A → C
- transaction$_3$:
  user updates wall B with one subscriber D
  - IP_2, → B
  - B → D



sharad.agarwal@microsoft.com

proven algorithms do not apply to this problem

- how to partition this graph among DCs while considering
  - latency of transactions (impacted by distance between users and dependent data)
  - WAN bandwidth (edges cut between dependent data)
  - DC capacity (size of subgraphs)
- sparse cut algorithms
  - models data-data edges
  - but not clear how to incorporate users, location / distance
- facility location
  - better fit than sparse cut and models users-data edges
  - but not clear how to incorporate edges and edge costs between data items
- standard commercial optimization packages
  - can formulate as an optimization
  - but don't know how to scale to O(100 million) objects

instead, we design a heuristic

- want heuristic that allows a highly parallelizable implementation
  - to handle huge scales of modern cloud services
  - many cloud services centralize logs into large compute clusters, e.g. Hadoop, Map-Reduce, Cosmos
- use logs to build a fully populated graph
  - fixed nodes are IP addresses from which client transactions originated
  - data items are nodes that can move anywhere on the planet (Earth)
- pull together or mutually attract nodes that frequently interact
  - reduces latency, and if co-located, will also reduce inter-DC traffic
  - fixed nodes prevent all nodes from collapsing onto one point
- not knowing optimal algorithm, we rely on iterative improvement
  - but iterative algorithms can take a long time to converge
  - starting at a reasonable location can reduce search space, number of iterations, job completion time
  - constants in update at each iteration will determine convergence

volley algorithm

- phase1: calculate geographic centroid for each data
  - considering client locations, ignoring data inter-dependencies
  - highly parallel
- phase2: refine centroid for each data iteratively
  - considering client locations, and data inter-dependencies
  - using weighted spring model that attracts data items
  - but on a spherical coordinate system
- phase3: confine centroids to individual DCs
  - iteratively roll over least-used data in over-subscribed DCs
  - (as many iterations as number of DCs is enough in practice)

Recursive Step:

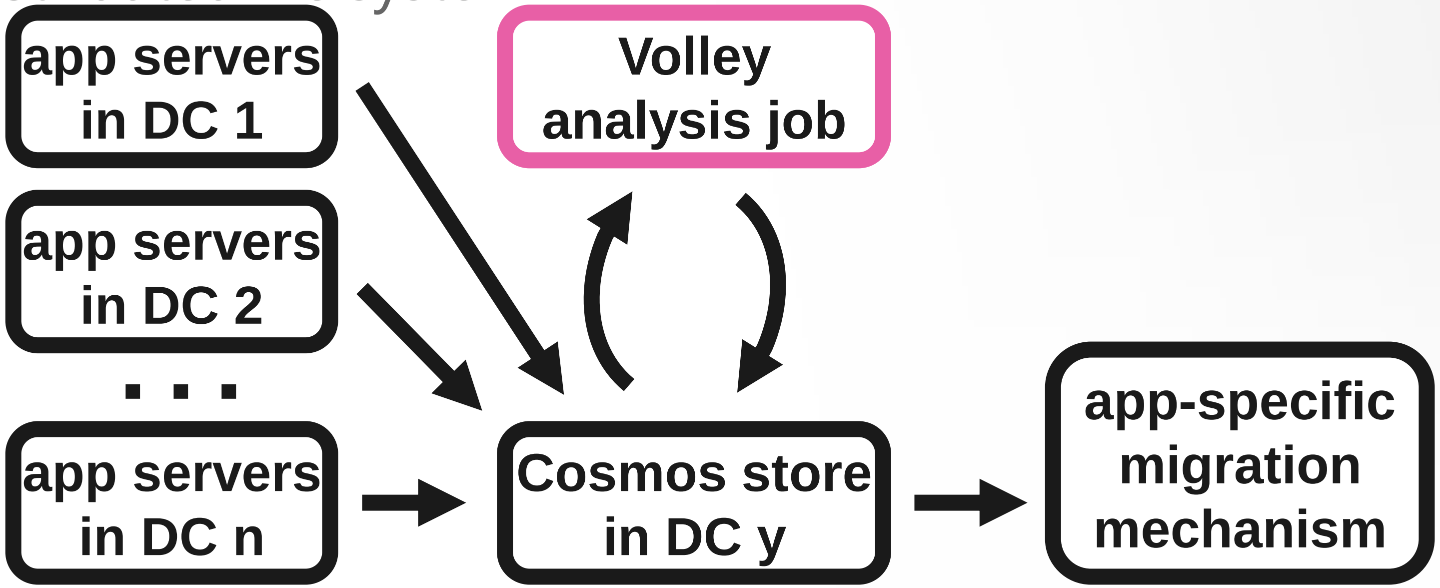
$$wsm\left(\{w_i, \vec{x}_i\}_{i=1}^{N}\right) = interp\left(\frac{w_N}{\sum w_i}, \vec{x}_N, wsm(\{w_i, \vec{x}_i\}_{i=1}^{N-1})\right)$$

$$\begin{aligned}
w &= \frac{1}{1 + \kappa \cdot d \cdot l_{AB}} \\
\vec{x}_A^{new} &= interp(w, \vec{x}_A^{current}, \vec{x}_B^{current}) \\
d &= \cos^{-1}[\cos(\phi_A)\cos(\phi_B) + \\
&\qquad \sin(\phi_A)\sin(\phi_B)\cos(\lambda_B - \lambda_A)] \\
\gamma &= \tan^{-1}\left[\frac{\sin(\phi_B)\sin(\phi_A)\sin(\lambda_B - \lambda_A)}{\cos(\phi_A) - \cos(d)\cos(\phi_B)}\right] \\
\beta &= \tan^{-1}\left[\frac{\sin(\phi_B)\sin(wd)\sin(\gamma)}{\cos(wd) - \cos(\phi_A)\cos(\phi_B)}\right] \\
\phi_C &= \cos^{-1}[\cos(wd)\cos(\phi_B) + \\
&\qquad \sin(wd)\sin(\phi_B)\cos(\gamma)] \\
\lambda_C &= \lambda_B - \beta
\end{aligned}$$

sharad.agarwal@microsoft.com

volley system overview

- consumes network cost model, DC capacity and locations, and request logs
  - most apps store this, but require custom translations
  - request log record
    - timestamp, source entity, destination entity, request size (B), transaction ID
  - entity can be client IP address or another data item's GUID
- runs on large compute cluster with distributed file system
- hands placement to app-specific migration mechanism
  - allows Volley to be used by many apps
- computing placement on 1 week
  - 16 wall-clock hours
  - 10 phase-2 iterations
  - 400 machine-hours of work



overview
how do users share data?
volley
evaluation

methodology

- inputs
  - Live Mesh traces from June 2009
    - compute placement on week 1, evaluate placement on weeks 2,3,4
  - 12 geographically diverse DC locations (where we had servers)
- evaluation
  - analytic evaluation using latency model (Agarwal SIGCOMM'09)
    - based on 49.9 million measurements across 3.5 million end-hosts
  - live experiments using Planetlab clients
- metrics
  - latency of user transactions
  - inter-DC traffic: how many messages between data in different DCs
  - DC utilization: e.g. no more than 10% of data in each of 12 DCs
  - staleness: how long is the placement good for?
  - frequency of migration: how much data migrated and how often?

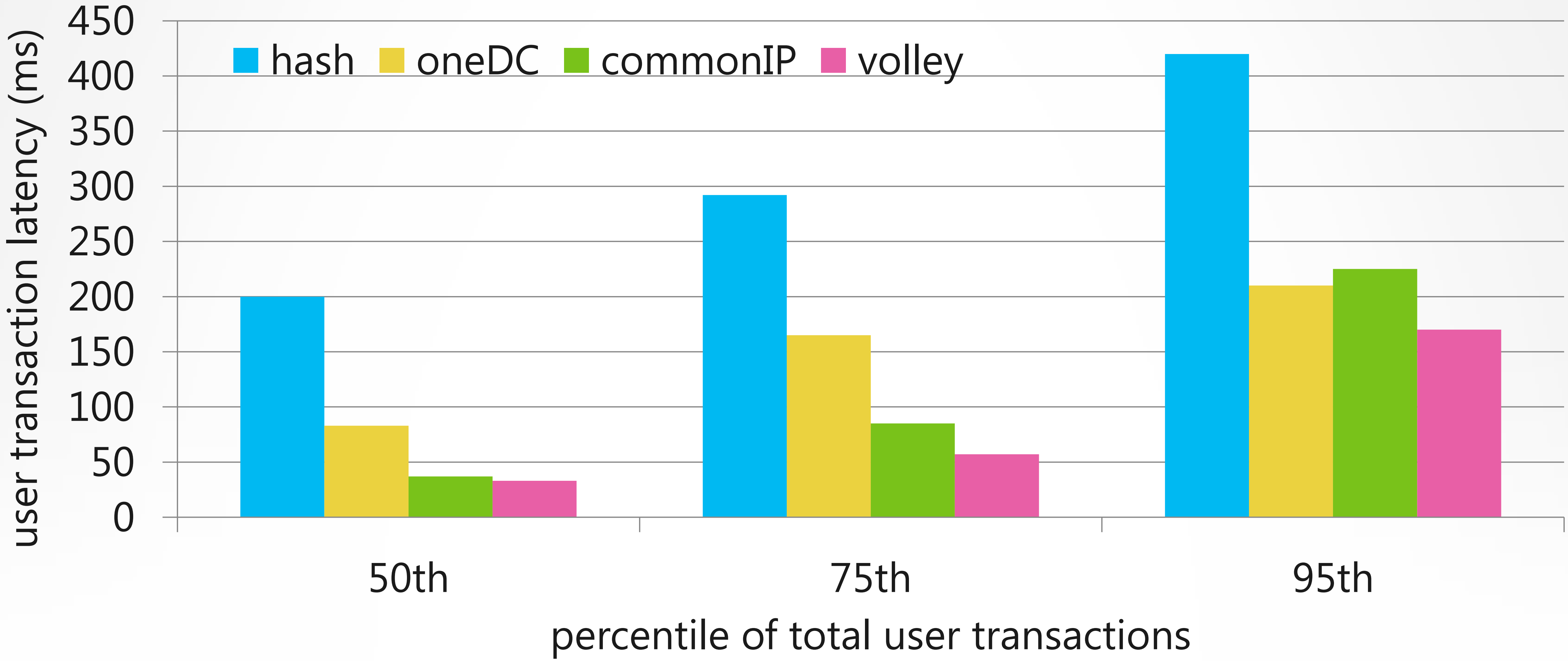
other heuristics for comparison

- hash
  - static, random mapping of data to DCs
  - optimizes for meeting any capacity constraint for each DC
- oneDC
  - place all data in one DC
  - optimizes for minimizing (zero) traffic between DCs
- commonIP
  - pick DC closest to IP that most frequently uses data
  - optimizes for latency by keeping data items close to user
- ~~firstIP~~
  - ~~(didn't work as well as commonIP)~~

sharad.agarwal@microsoft.com

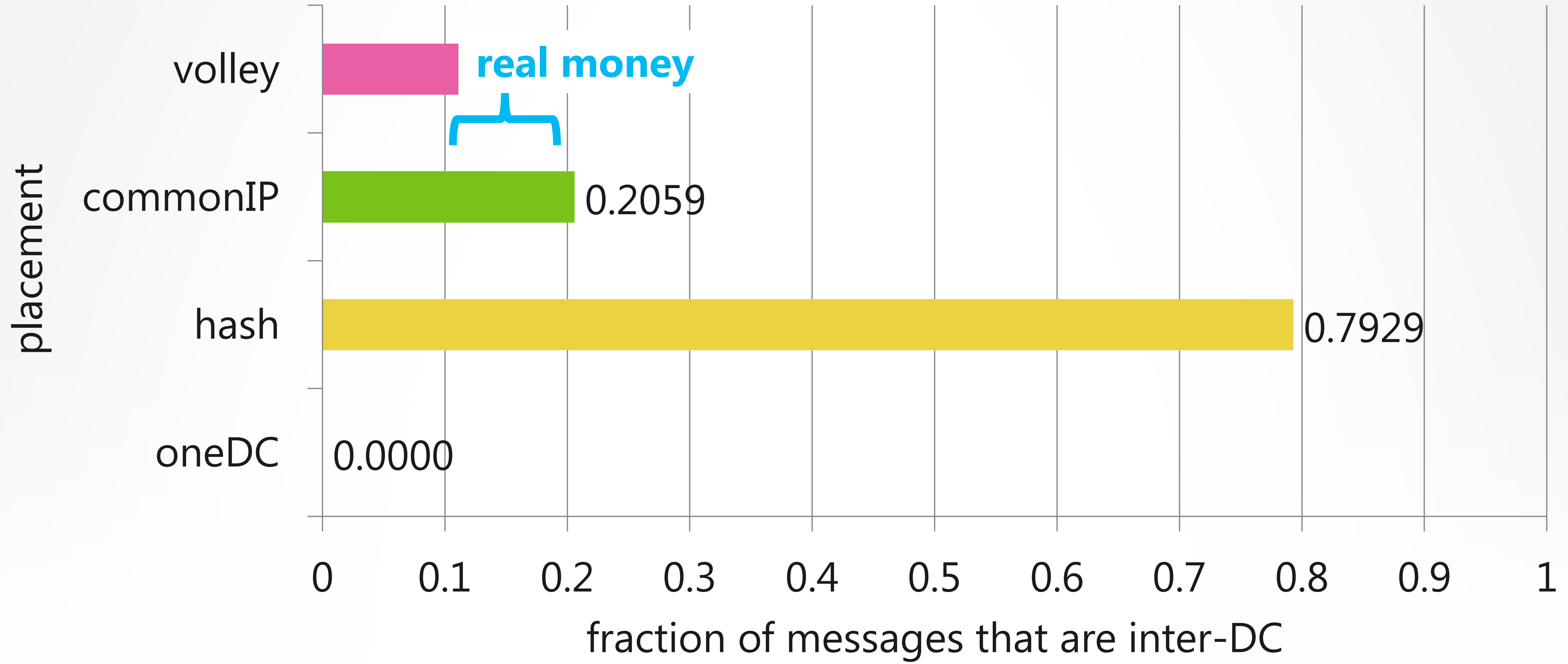
user transaction latency (analytic evaluation)

INCLUDES SERVER-SERVER (SAME DC OR CROSS-DC) AND SERVER-USER



inter-DC traffic (analytic evaluation)

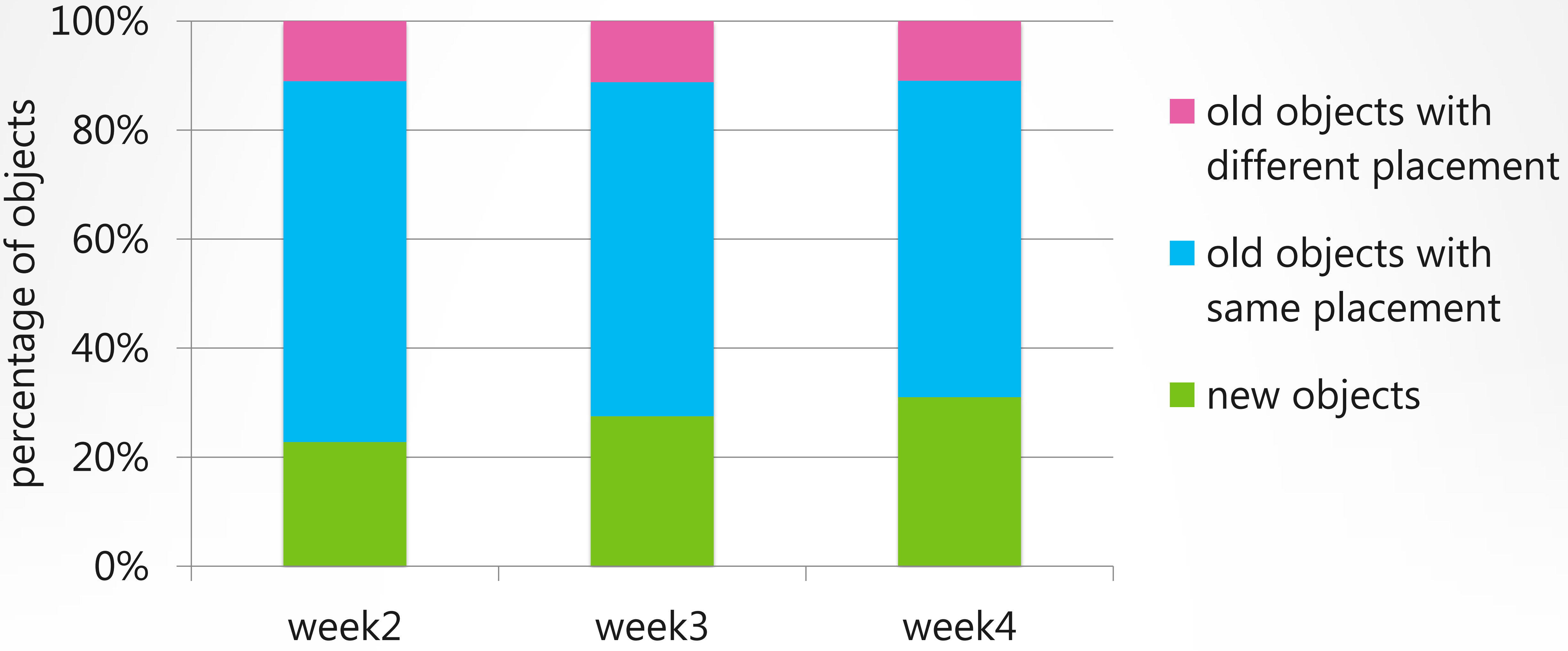
WAN TRAFFIC IS A MAJOR SOURCE OF COST FOR OPERATORS



sharad.agarwal@microsoft.com

how many objects are migrated every week

COMPARED TO FIRST WEEK



summary

- Volley's data partitioning
  - simultaneously reduces user latency and operator cost
  - reduces datacenter capacity skew by over 2X
  - reduces inter-DC traffic by over 1.8X
  - reduces user latency by 30% at 75th percentile
  - runs in under 16 clock-hours for 400 machine-hours computation across 1 week of traces
- Volley solves a real, increasingly important need
  - partitioning user data or other application state across DCs
  - simultaneously reducing operator cost and user latency
- more cloud services built around sharing data between users (both friends & employees)
- cloud providers continue to deploy more DCs

thanks!

sharad agarwal

john dunagan

navendu jain

stefan saroiu

alec wolman

harbinder bhogan