Dynamic Replication and Partitioning

Costin Raiciu

University College London

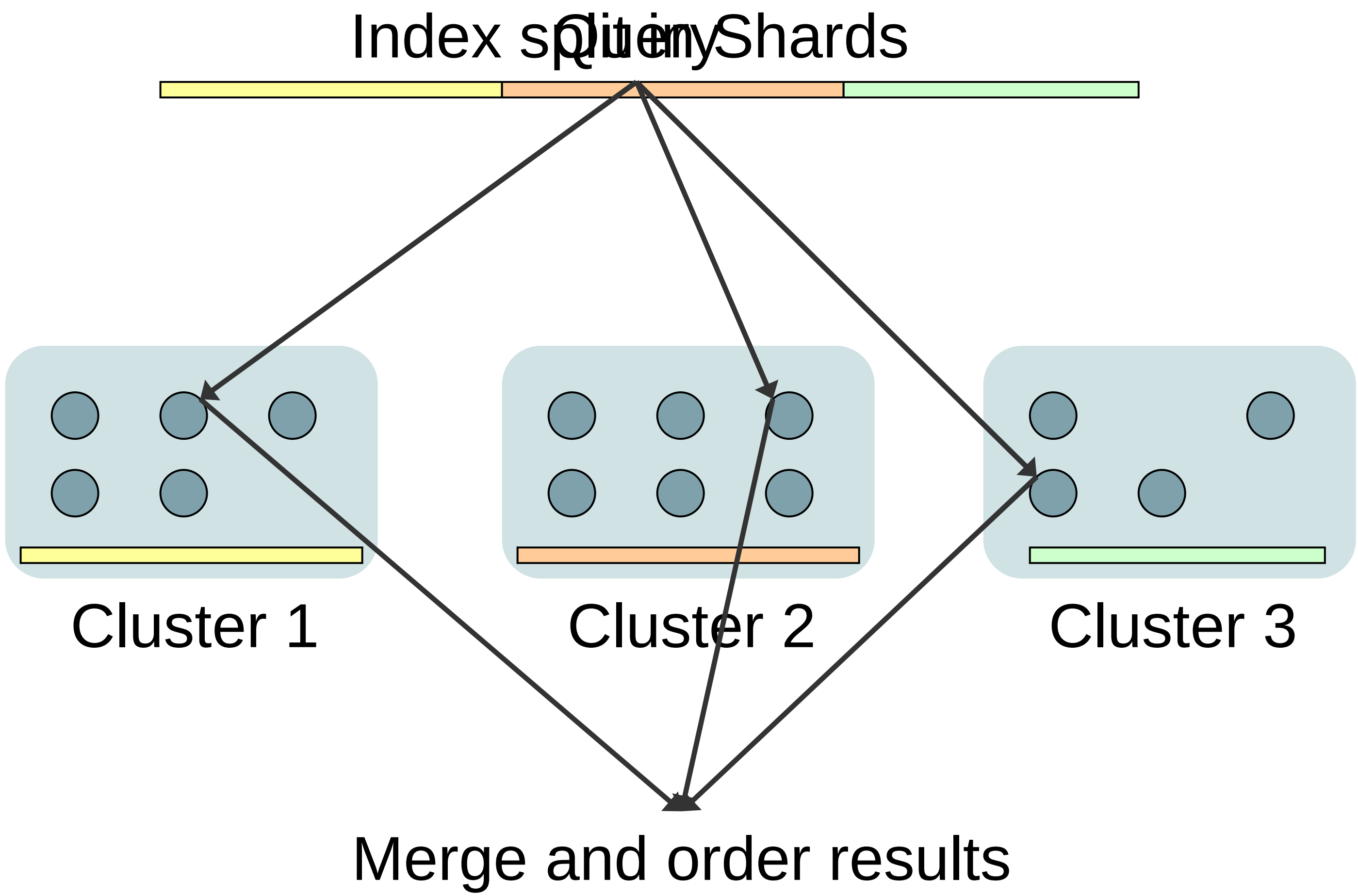
Joint work with

Mark Handley, David S. Rosenblum

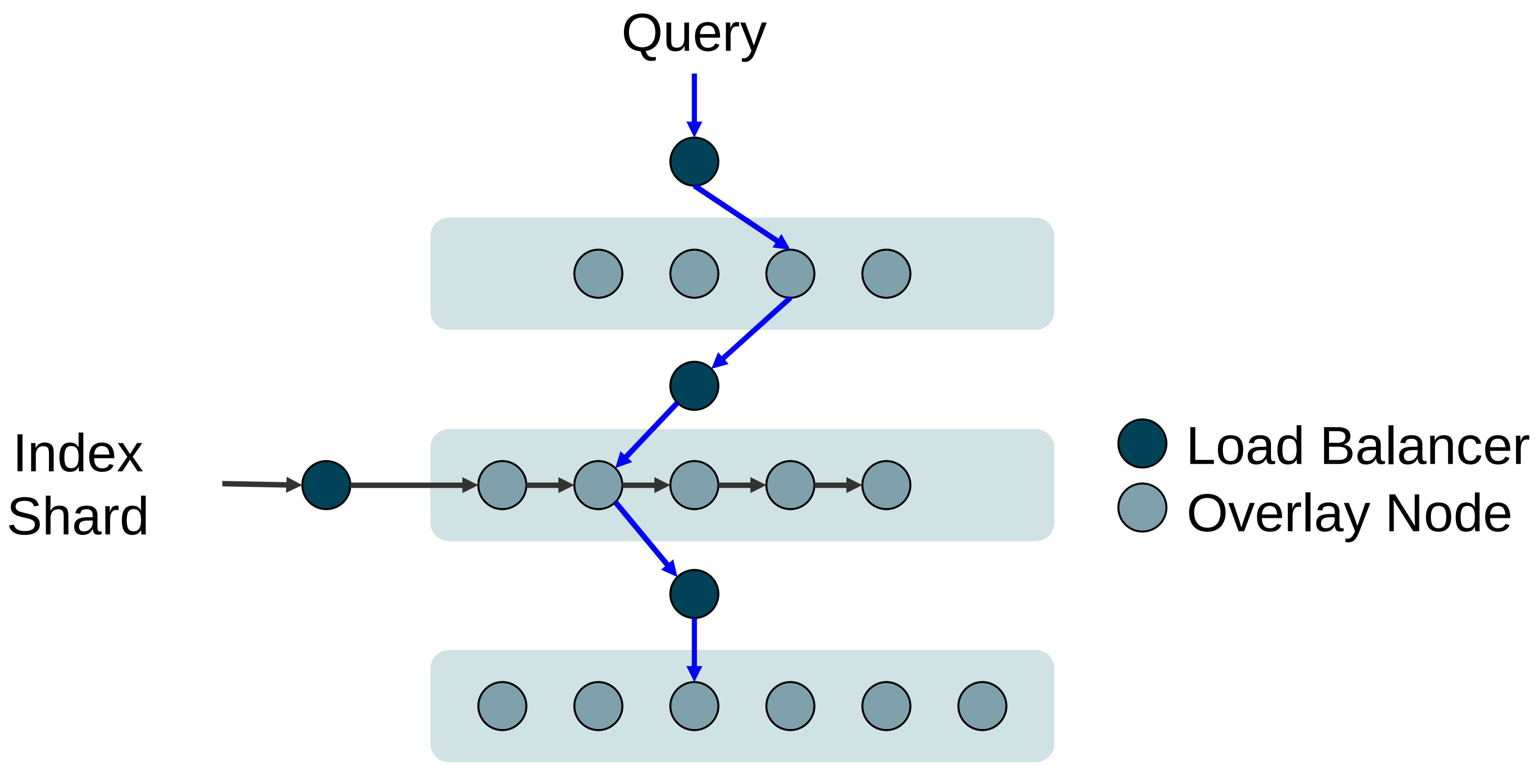
Motivation: Web Search

- Search engines
  - Create an index of the web
  - Queries consult the index to find relevant documents
  - The documents are then ordered (e.g. Page Rank)
- The index is huge: a few TB
  - Must be partitioned to fit into memory
  - Must be replicated to increase query throughput and system availability

Google Web Search (Barroso et. al)



Big Picture: Distributed Rendez-Vous



Average Replication Level R=5
Hop Count H=3

Distributed Rendez Vous is important

- Many other applications use it
  - Online Filtering
  - Distributed databases
- Combines replication and partitioning
  - Increasing replication (R) increases availability, but has high cost for storing the index
  - Increasing the forwarding hops (H) creates high bandwidth cost for transient objects
  - Tradeoff: $R \cdot H \geq$ #nodes

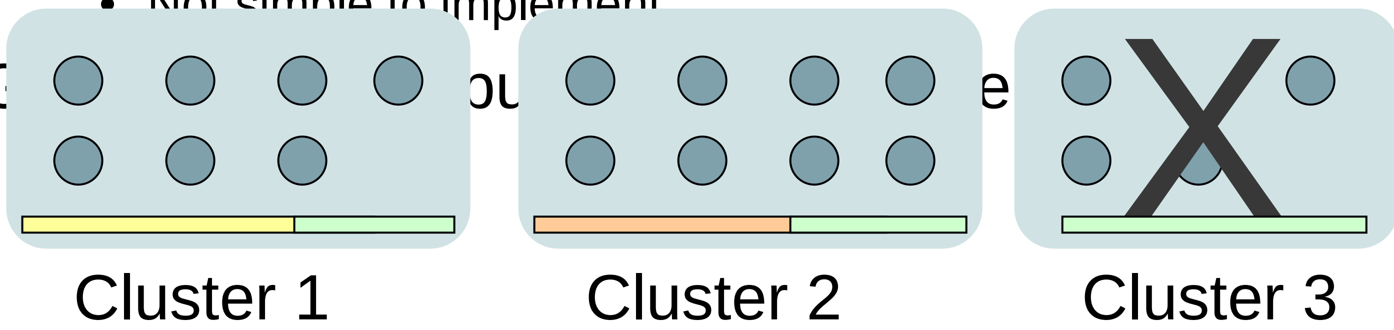
The Problem

- Who chooses the number of clusters? Depends on:
  - Frequencies and sizes of index and queries
  - Bandwidth constraints
  - Memory constraints
  - Number of nodes
- R varies with time!

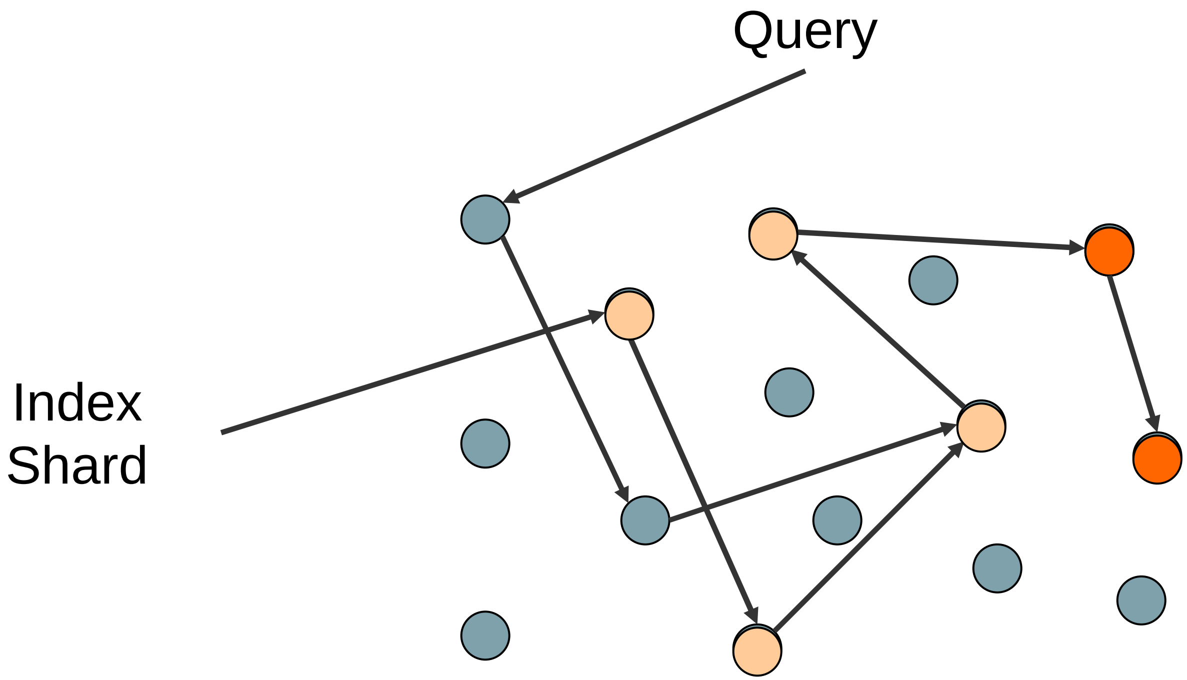
How can we adjust the *Replication Rate* in distributed rendez-vous?

Obvious approach

- Google architecture
  - Replication tied to network structure
  - Increase replication level
    - Destroy cluster, add the nodes to the other clusters
  - Issues
    - Temporarily reduces the capacity of the network
    - Not simple to implement



A randomized implementation



N=15
R=5
H=3

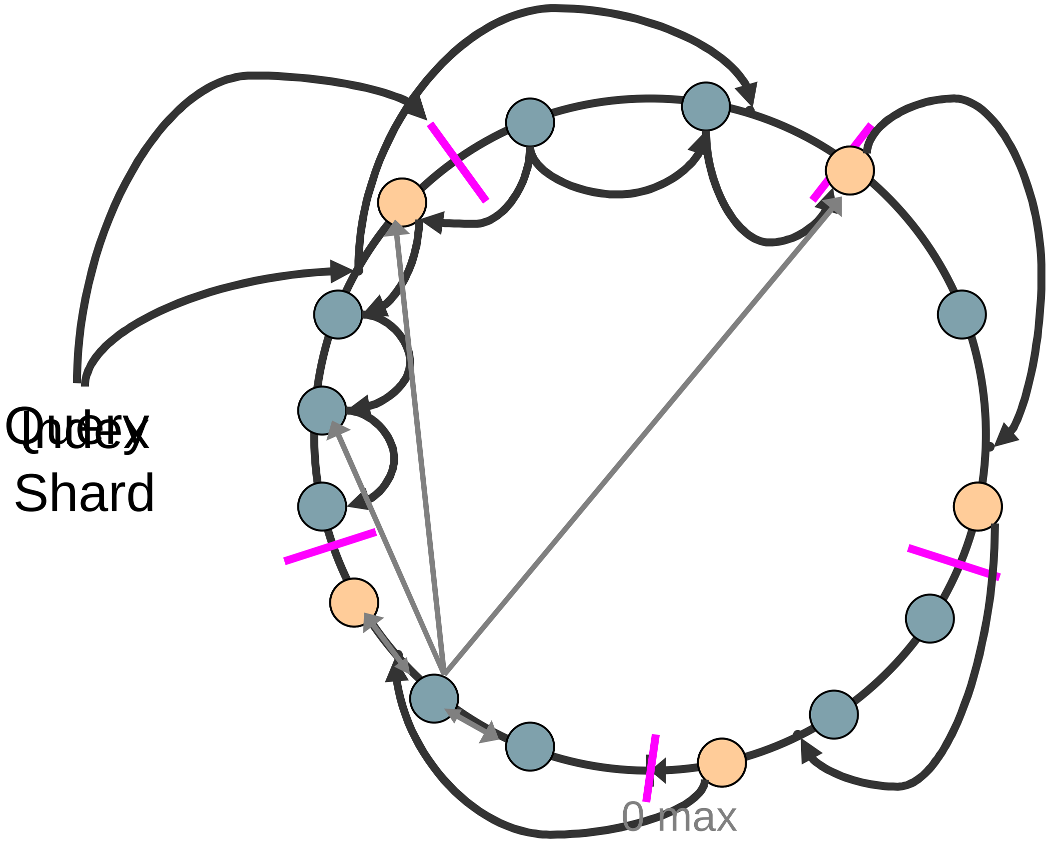
To increase the replication level, each node creates 1 new replica for active queries

On average, each query meets each index shard once

Our solution: ROAR

- **R**endez-Vous **O**n **A** **R**ing
  - Similar in spirit to Random
  - But with deterministic properties
  - Does not tie network structure to replication level

ROAR Overview



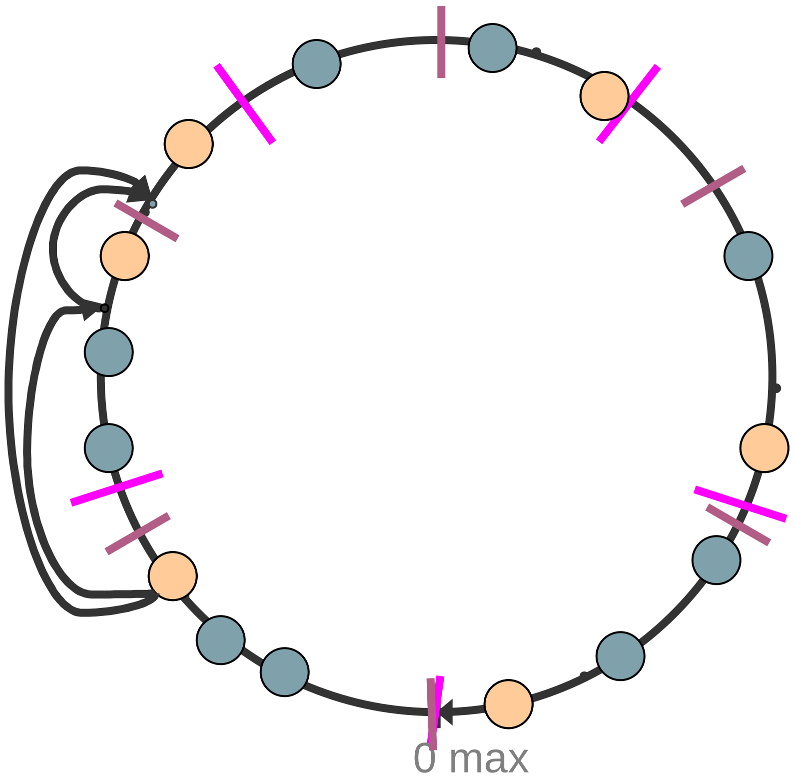
Replication Level: 5

- Nodes on a Chord ring
- ID space virtually split in R intervals
- Replicate
  - Hash and store
  - Forward to **equivalent** node in next interval
- Route
  - Uniformly choose interval and direction
  - Route to all nodes in that interval

ROAR Analysis

- Equal spacing is important
  - When R increases, it ensures that no 2 replicas are in the same interval
  - **Stable state:** if R is constant enough time, equivalent nodes have equivalent content
    - Useful for fault tolerance
  - When R changes:
    - Stability is maintained if R is doubled of halved
    - Otherwise, not stable: wait for objects to expire

Increasing Replication

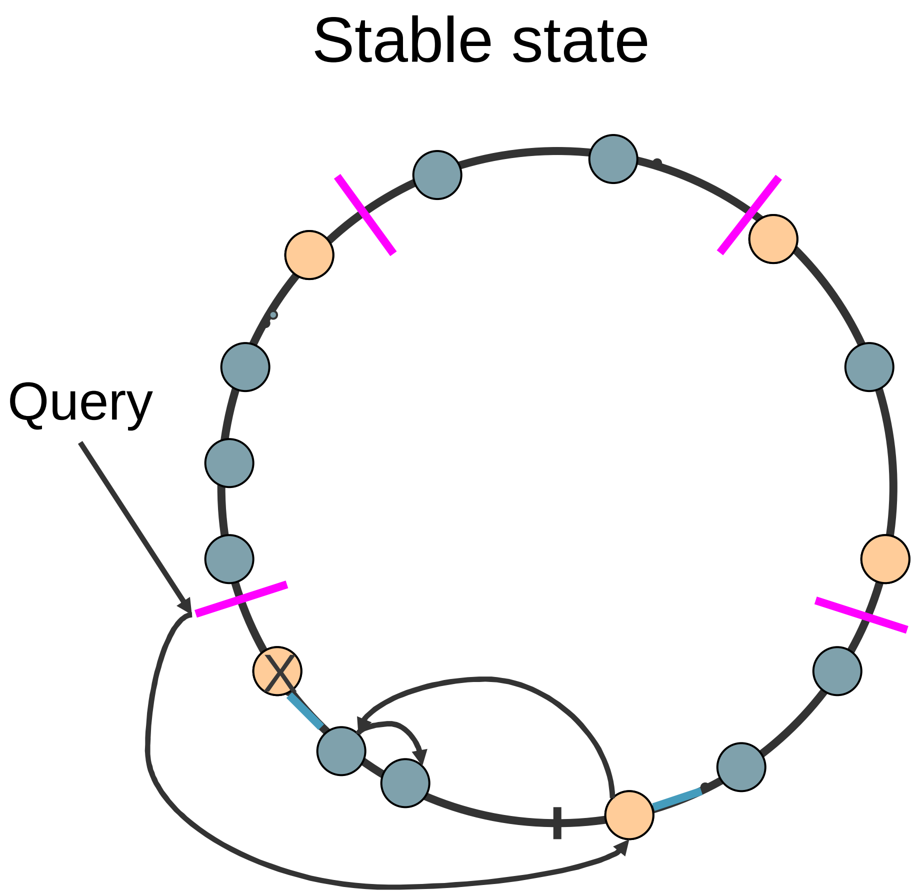


Replication Level: 5 -> 6

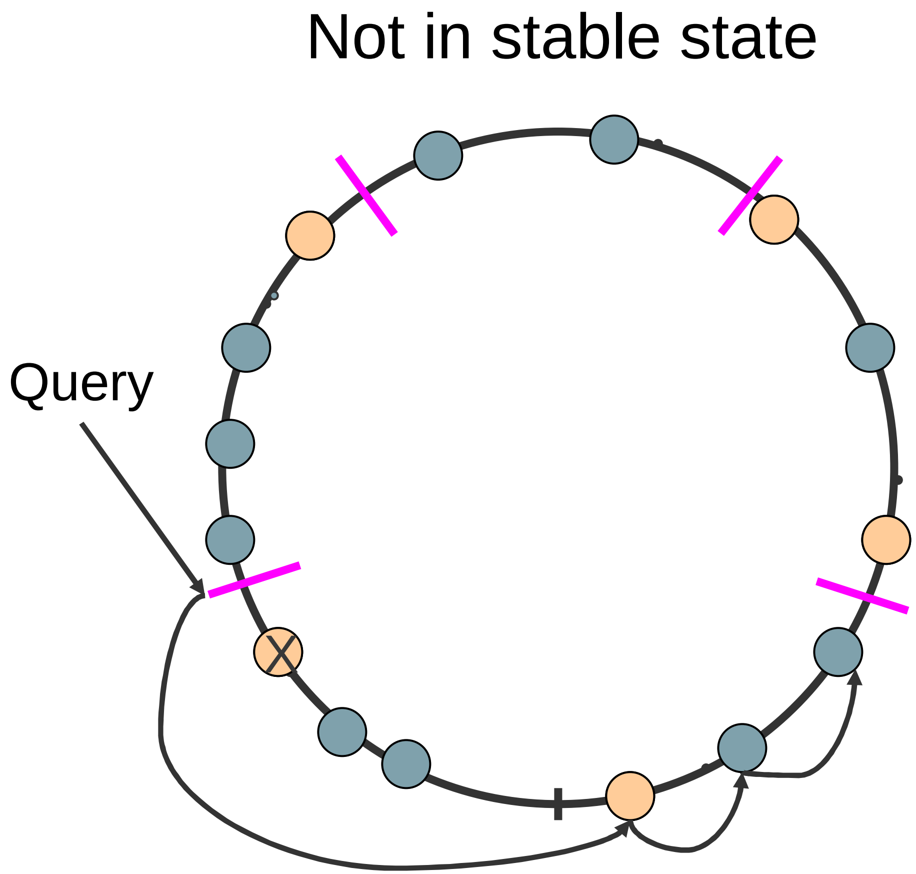
Increasing Replication (2)

- **Observation**. When replication level is R, we can route at any level R’≤R.
- ROAR can route while changing replication levels
  - Wait until all nodes in interval reach new replication
  - Begin routing at new replication level
- When is the new replication level reached?
  - Compute persistent object count at replication level R and R+1
    - When approximately equal, safe to switch to new routing.
  - Count is piggybacked on queries - very small cost

Fault Tolerance



Fault Tolerance



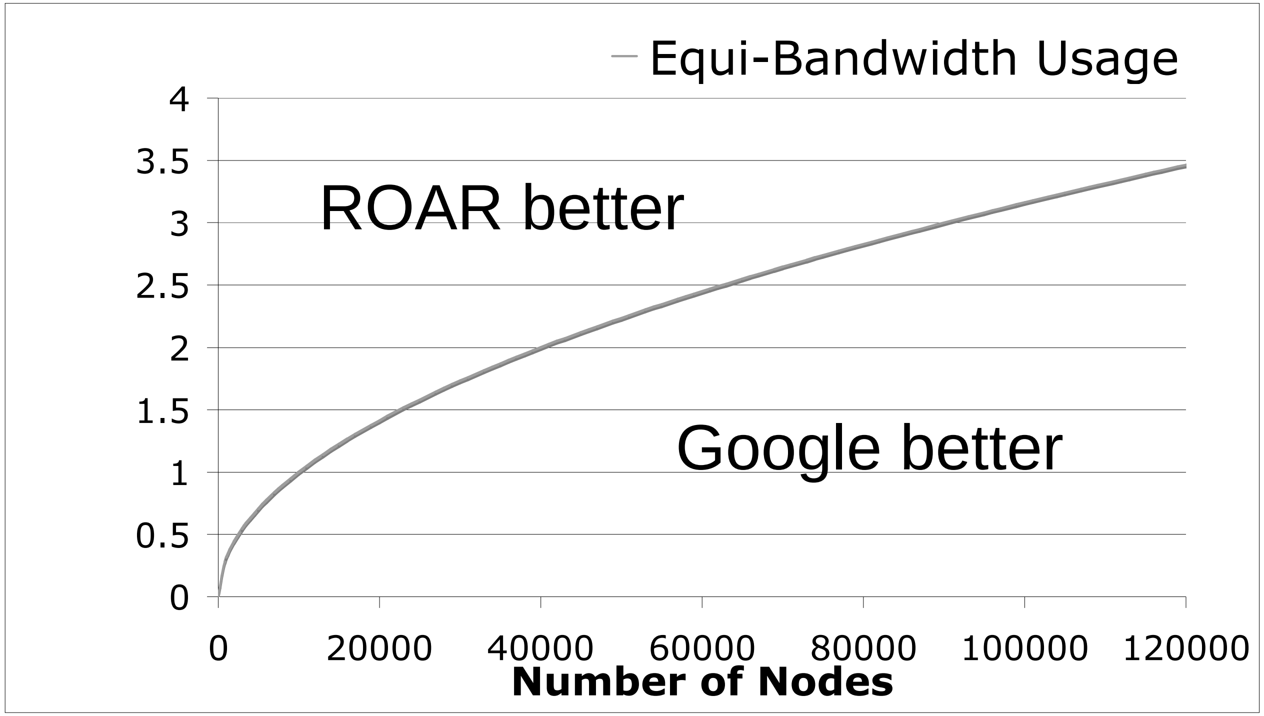
Comparison

- Bandwidth scarce system
  - R = O(√N)
  - I = # total size of index

	Google	Random	ROAR
RV Guaranteed?	Yes	35% miss probability	Yes
RV Redundant?	No	25% redundant RV probability	No
Bw for R = R+1	~2·I	I	I
Bw Cost on Node Failure	1	O(I·R/N)	O(I·R/N) or 1

Comparison (2)

- 1% permanent failures per year
  - Commercial data: 5% failures in 1st year
  - Transient failures tolerated with stable state



Summary

- Distributed rendez-vous is an important problem in distributed computing
  - Changing R is a requirement for optimal solutions
- ROAR - simple algorithm
  - Distributed in spirit
    - No need for external load balancing
    - Can run on deployed structured overlays
  - Achieves reconfiguration without changing network structure
  - In stable state as good as Google
  - When reconfigurations are often, does better

References

- Web Search for a Planet: the Google Cluster Architecture - Barroso et. al