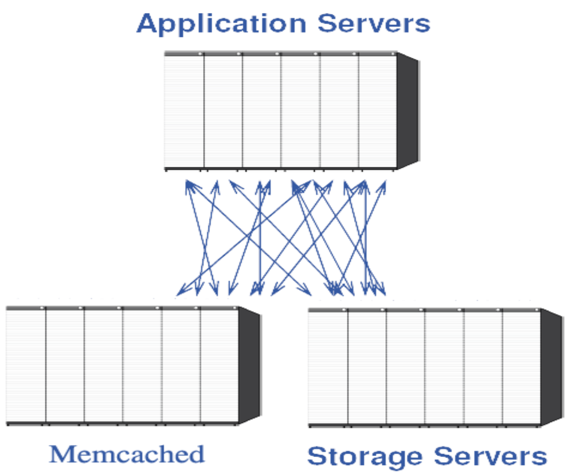
# RAMCube: Exploiting Network Proximity for RAM-Based Key-Value Store

Yiming Zhang, Rui Chu
@ NUDT
Chuanxiong Guo, Guohan Lu, ***Yongqiang Xiong***, Haitao Wu
@ MSRA

June, 2012

# Background

- Disk-based storage is problematic
  - I/O latency and bandwidth
- Current typical storage system
  - Using RAM as a cache
  - App servers + storage servers + cache servers
  - Facebook keeps more than 75% data in memcached



# Why Cache is NOT Preferred

- Consistency
  - App is responsible for consistency (e.g. memcached)
    - Error-prone
    - E.g., the failure of Facebook Automated Verifying System


PR NEWS 2011 GIFT GUIDE APPS AUTOS

Facebook Down: Service Unavailable – DNS failure

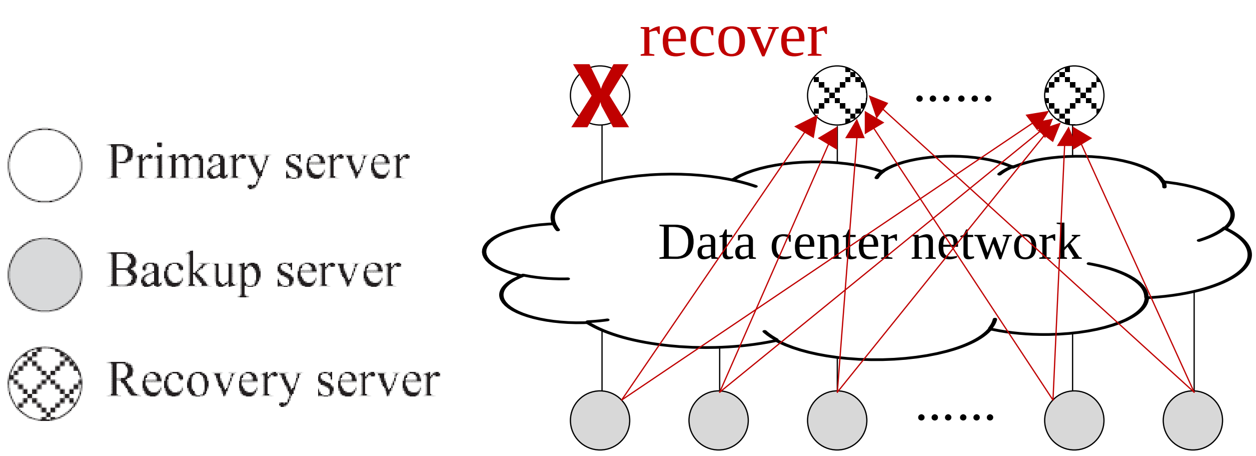
**By:** Peter Chubb | September 23, 2010 | 514 Comments

+1 Recommend this on Google

- Efficiency
  - RAM is 1000X faster than disk

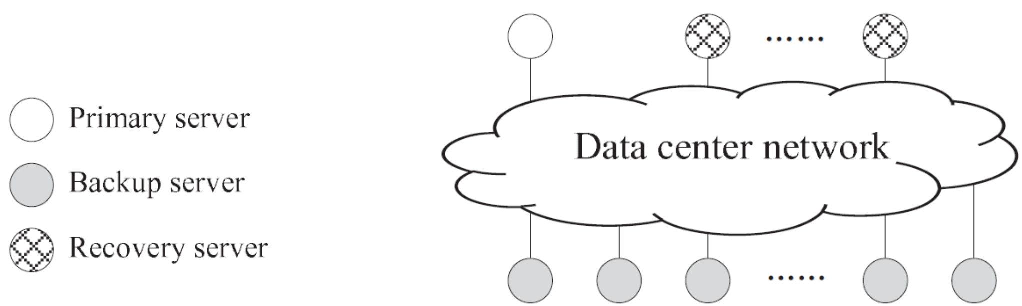
# Using RAM as a Persistent Store

- RAMCloud: RAM-based key-value store
  - Data is kept entirely in the RAM of servers
- Primary-backup for durability
  - Multiple copies for each key-value pair
  - Primary node + backup node + recovery node
    - Write/read/recover procedure



# RAMCloud (cont.)

- Fast failure recovery
  - Availability: MTTF / (MTTF + MTTR)
- Not specifically designed for DCN
  - Temporary network problems cause false failure detection
  - Parallel unarranged recovery flows
  - ToR switch failures



# RAMCube: Exploiting DCN Proximity

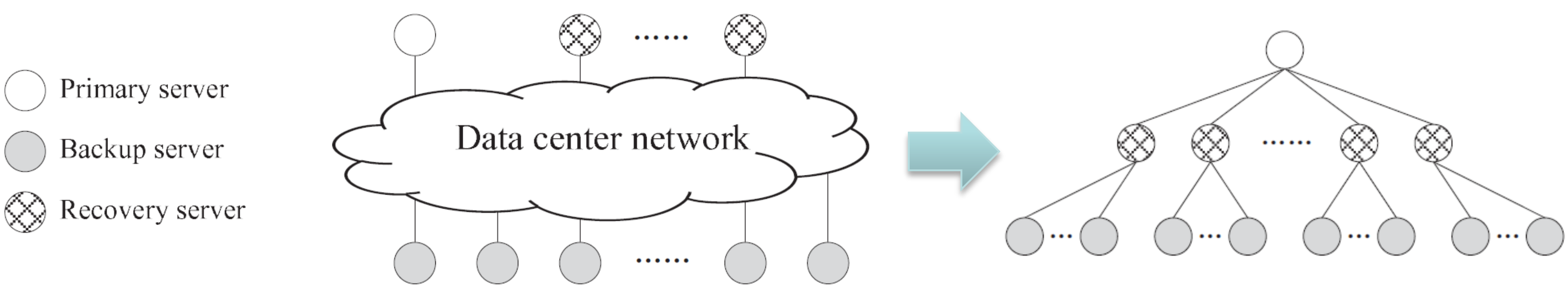
- Goal of RAM-based storage
  - ✓High performance
    - Low latency
      - – 10+ μs read/write latency
    - Large scale
      - – Hundreds of TB data
    - High throughput：
      - – Millions of read ops per server per sec
      - – Hundreds of thousands of write ops / server / sec
  - ✓Fast failure recovery for high availability
    - 1~2 sec

# RAMCube Design Choices

- Network hardware
  - InfiniBand is high-bandwidth, low-latency
    - But expensive and not common in DCN
  - ✓We use commodity Ethernet-based BCube
- Primary-backup vs. symmetric replication
  - Symmetric replication easy to achieve high availability
    - But uses more RAM
  - ✓We use primary-recovery-backup like RAMCloud

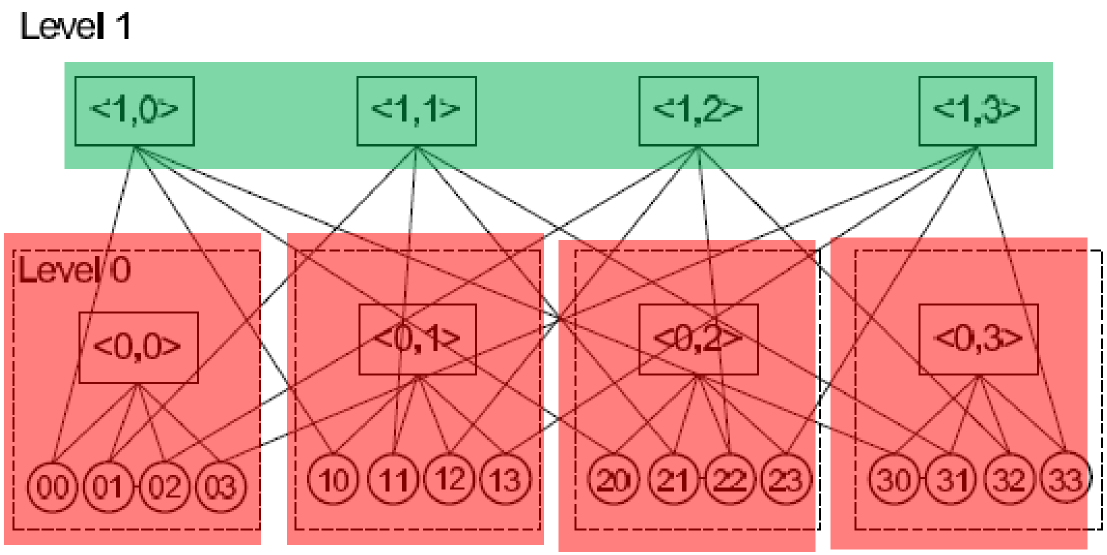
# Basic Idea of RAMCube

- DCN topology is known, not a blackbox
- Exploiting DCN proximity
  - ✓ Form a directly-connected-tree
  - ✓ Primary-recovery, recovery-backup are 1-hop neighbors
  - ✓ Recovery servers
    - watch their primary server
    - fetch data from their backup servers in recovery



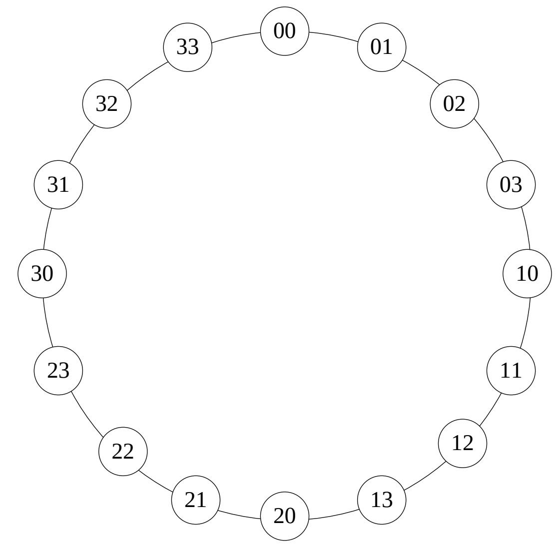
# Using BCube

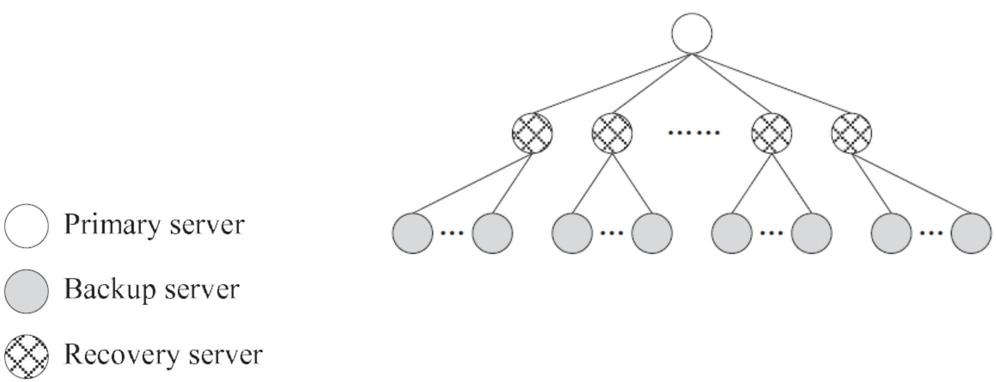
- BCube is a server-centric network
  - ✓Switches act as layer-2 crossbars
  - ✓Recursively defined
    - E.g., A BCube(4,1) is constructed from 4 BCube(4,0) and 4 4-port switches

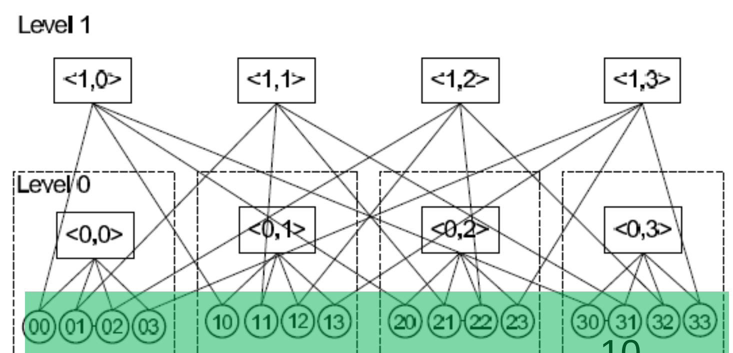


# Mapping Key Space onto BCube

- Multi-layer logical rings
- Primary ring
  - All servers in BCube are primary servers.
  - The whole key space is mapped onto primary ring
    - Each primary server for a subset of the key space

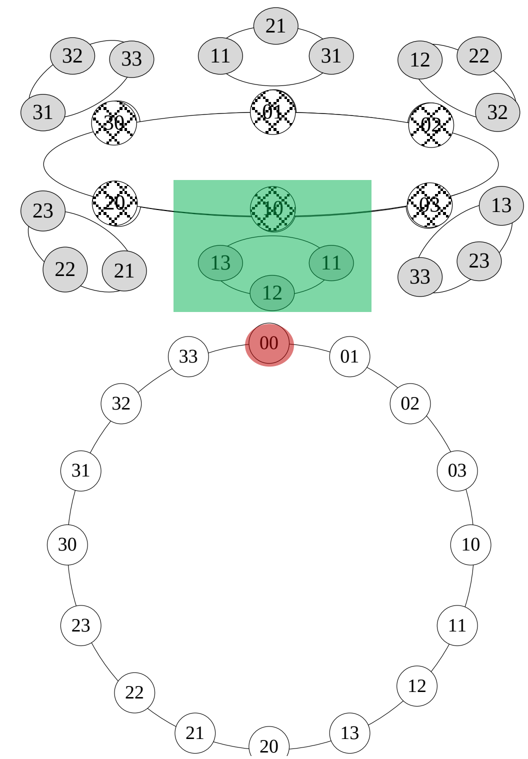


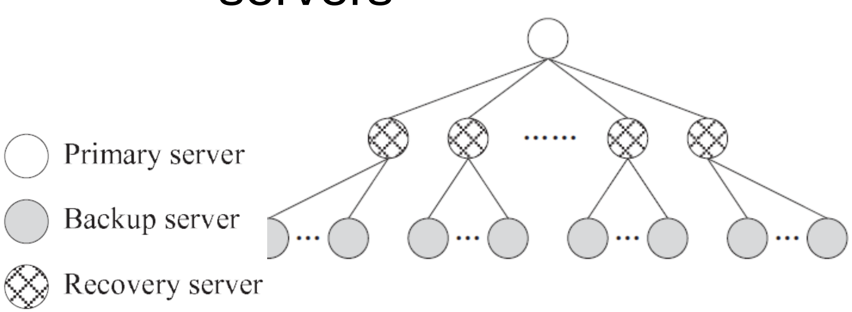



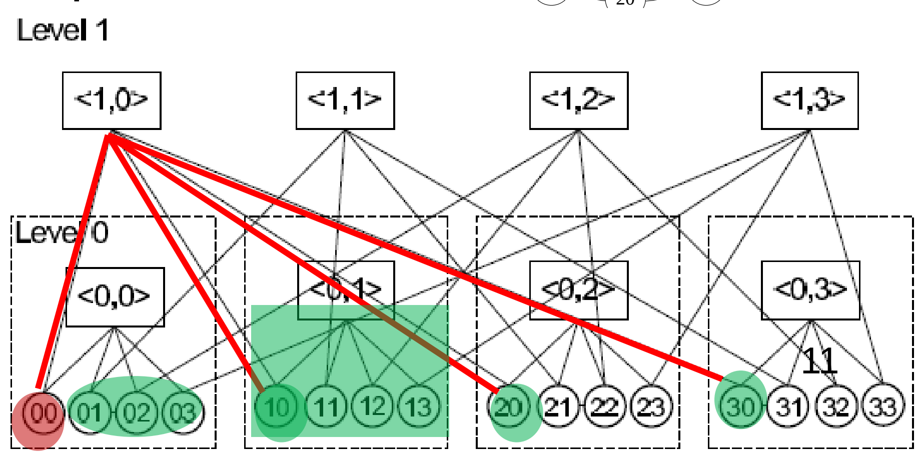



# Multiple Rings in RAMCube (cont.)

- Each primary server *P* has a **recovery ring**
  - Composed of 1-hop neighbors to *P*
  - Map the ***p***'s key space to recovery servers
- Each recovery server *R* has a **backup ring**
  - Composed of servers 1-hop to *R* and 2-hop to *P*
  - Map the ***r***'s key space to backup servers

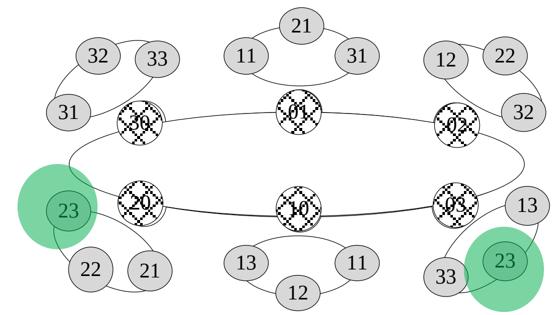


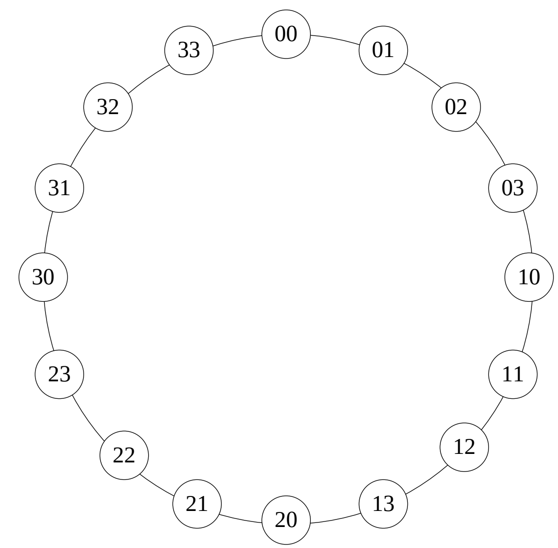



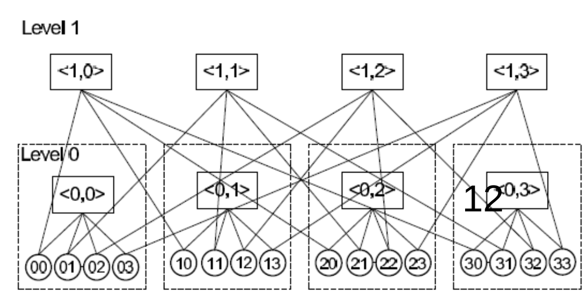



# RAMCube Property

- MultiRing of RAMCube for BCube(n, k)
  - # of primary servers
    - $P = n^{k+1}$
  - # of recovery servers for each primary p
    - $R = (n-1)(k+1)$
  - # of backup servers for each primary server
    - $B = (n-1)^2 k(k+1)/2$
- Each primary node has plenty of recovery and backup servers
  - BCube(16,2),
    - P = 4096, R = 45, B = 675
  - BCube(4,1),
    - P = 16, R = 6, B = ***9***

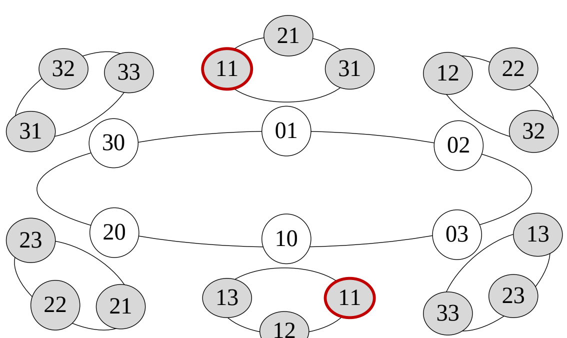


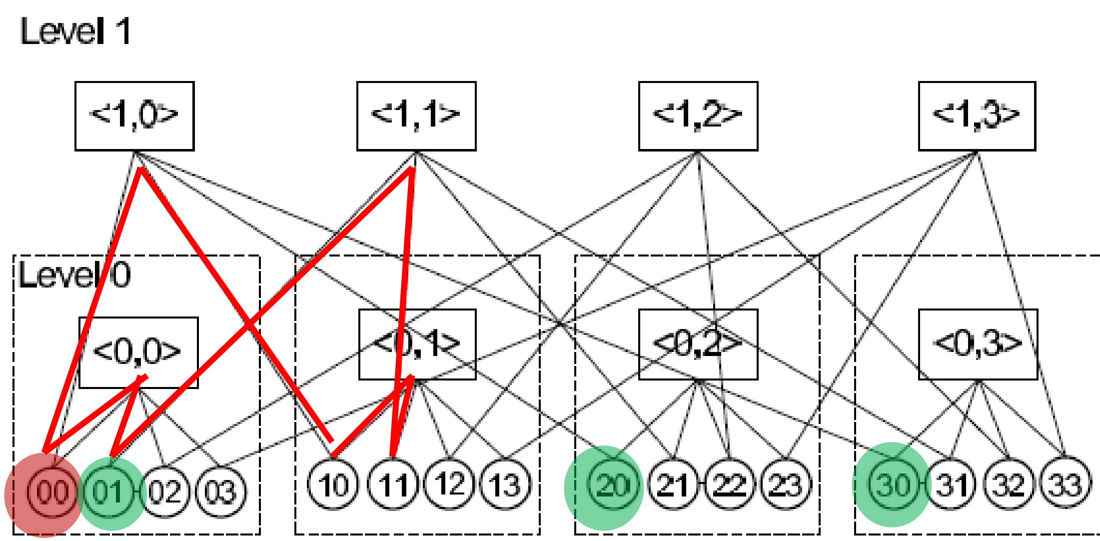



# Failure detection

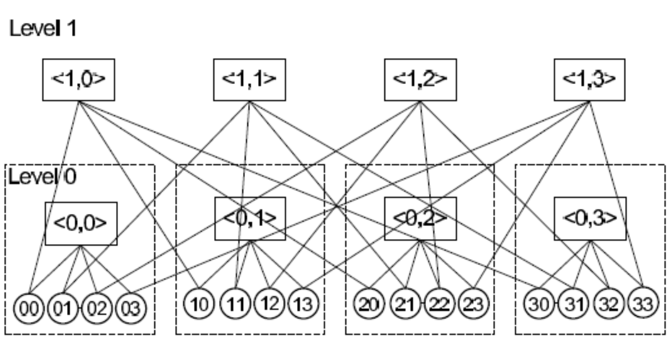
- Heartbeats
  - Primary node periodically sends heartbeats to each of its recovery nodes
- Confirmation of failure
  - Recovery node uses source routing to issue additional pings to suspicious primary node

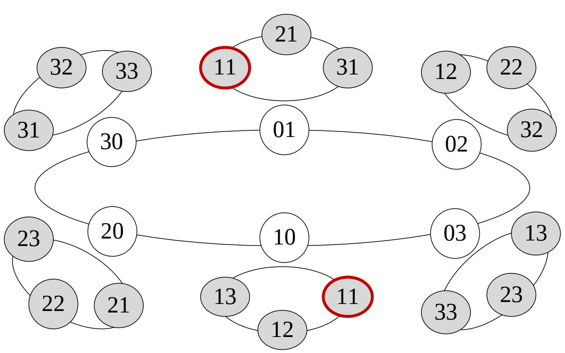


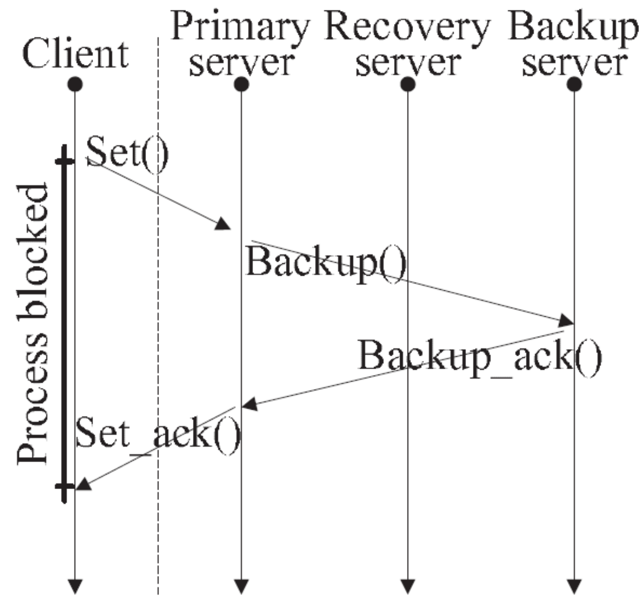
# Failure Recovery

- Primary node failure
  - Recovery nodes fetch dominant copies to their RAM from directly-connected backup nodes
  - Given 10Gbps network bandwidth and 100MB/s disk bandwidth
    - BCube (16,2) can easily recover 64GB data in 1~2 sec
- Recovery/backup node failure
  - Not as critical as primary node failure



# Prototype Implementation



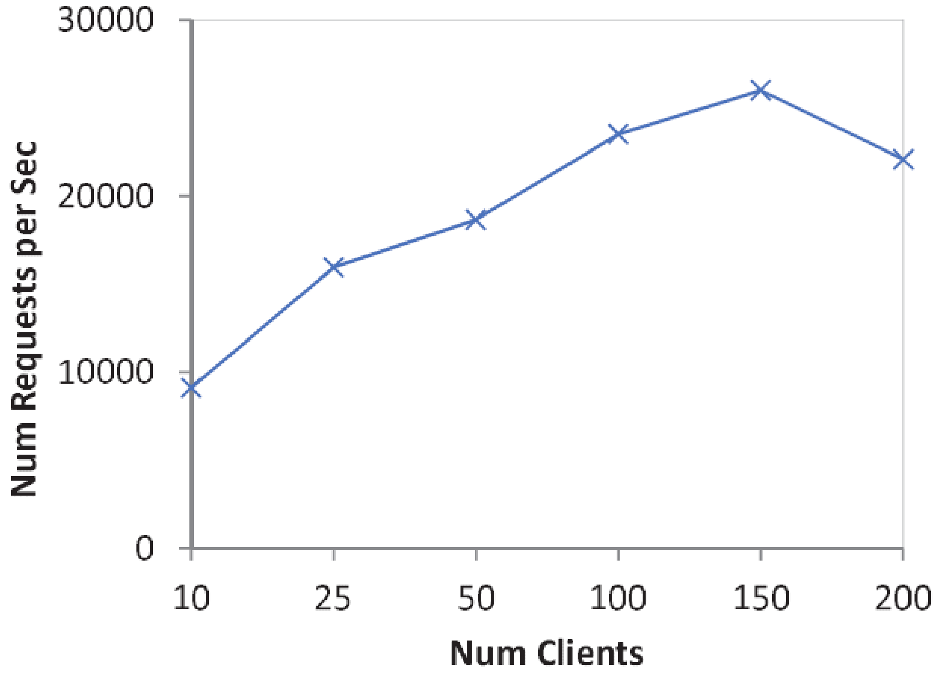

- Platform & language
  - Windows 2008R2,
  - C++, libevent
  - Code size: 3000+ lines
- RAMCube components
  - Connection manager
    - maintains the status of directly connected neighbors and handles network events
  - memory manager
    - uses a slab-based mechanism and handles set/get/delete requests

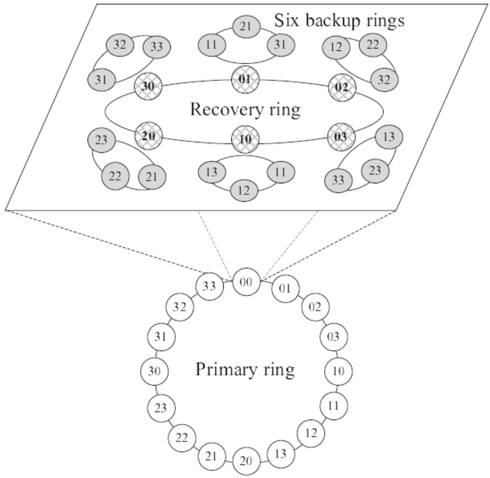
# Evaluation

- Evaluation configuration
  - BCube(4,1) with 16 servers
  - Eight 2.4GHz cores & 16G RAM per node
  - Single-threaded server and client
    - Both with a busy loop
  - 15B key + 100B value
- Experiments
  - Throughput
  - Server failure recovery
  - Switch failures

# Experimental Results -1

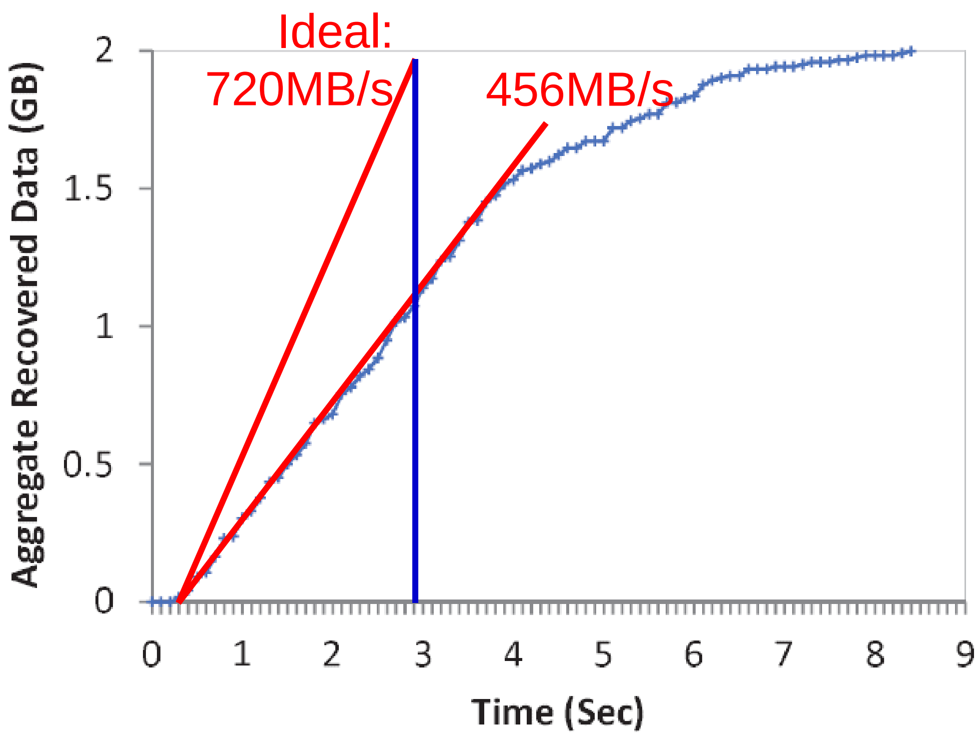
- Throughput
  - ✓No. of writes / core / sec
  - ✓Similar to memcached/RAMCloud



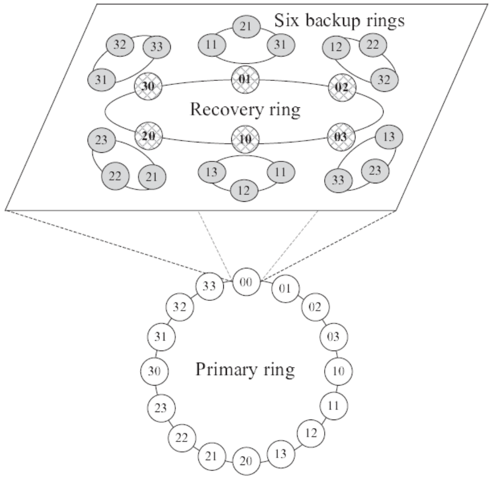



# Experimental Results -2

- Server failure recovery
  - ✓Aggregate recovery result (2 GB data)
  - ✓Still have room to improve (more ports/cores)

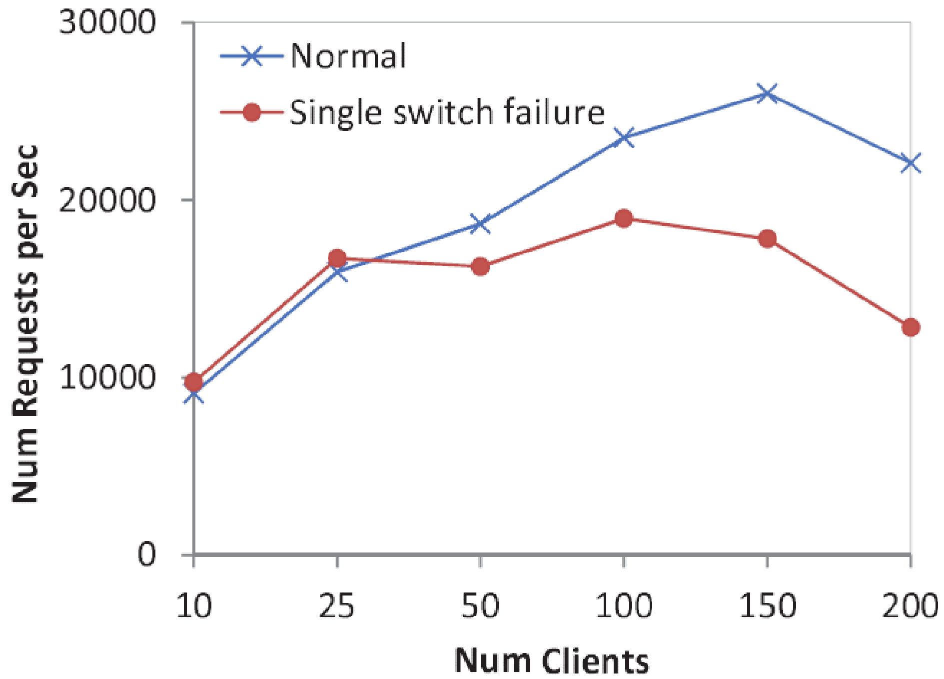


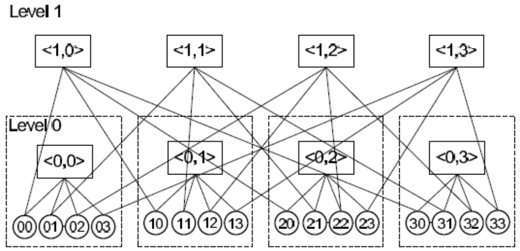

- Total Disk I/O (backup) 9*100MB/s = 900MB/s
- Total net I/O (recovery) 6*120MB/s = 720MB/s



# Experimental Results -3

- Switch failure
  - ✓Throughput after a switch failure
  - ✓Graceful performance degradation



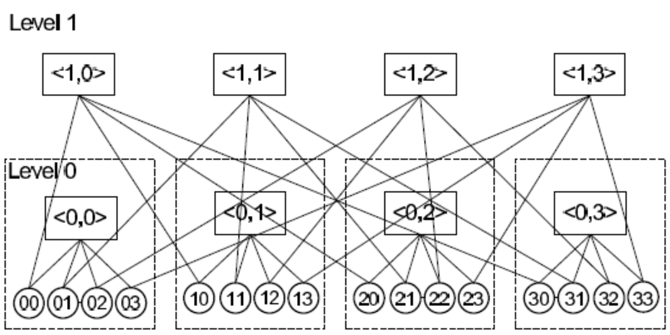



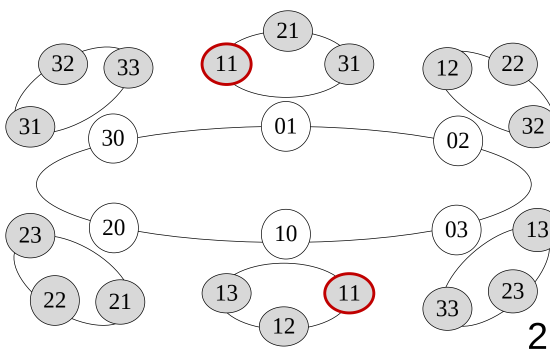
# Future Work

- SSD & Low-latency Ethernet
- Rich data model
  - Table/tablet support, column family
- Utilizing multi-core
  - Improve read/write ops throughput

# Summary

- RAMCube: RAM-based persistent k-v store
- Exploiting network proximity of BCube
  - Failure detection – 1-hop
    - Recovery nodes directly watch their primary nodes
  - Recovery traffic – 1-hop
    - No unexpected traffic overlap
  - Switch failures – multi-path
    - Graceful performance degradation



*Thank You!*