CubicRing:
Enabling One-Hop Failure Detection/Recovery in Distributed In-Memory Storage Systems

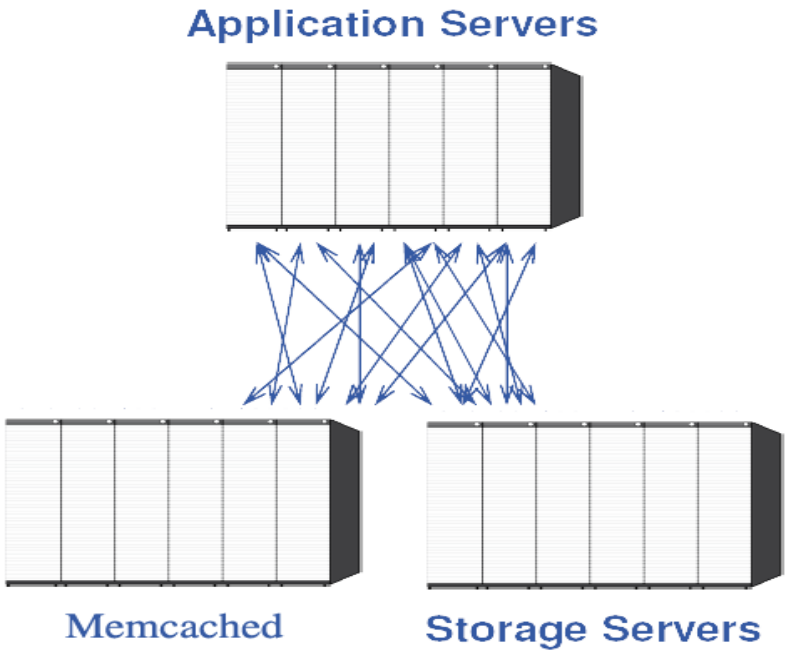
Yiming Zhang, ***Chuanxiong Guo***, Dongsheng Li,
Rui Chu, Haitao Wu, Yongqiang Xiong

NSDI 2015

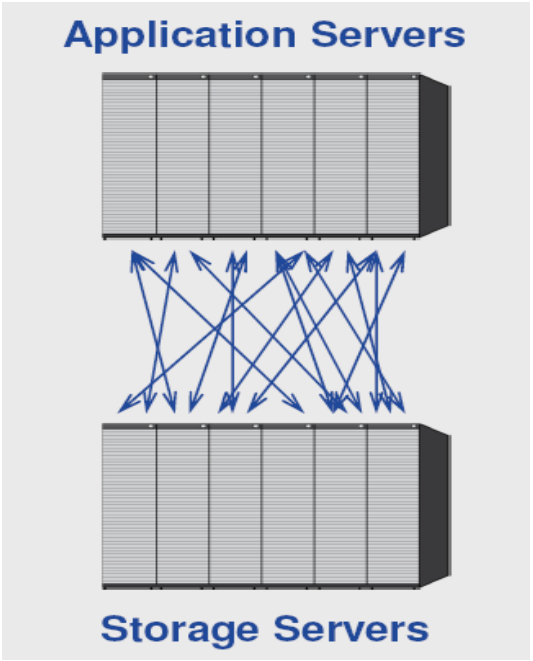
Background

- Traditional disk-based storage systems
  - Use RAM as a cache
  - App servers + storage servers + cache servers
  - Facebook keeps more than 75% of its online data in its memcached servers (2011)

- Disk-based storage issues
  - I/O latency and bandwidth
  - Cache consistency



In-Memory Storage Systems



- Use RAM as persistent storage
  - Data is kept entirely in the RAM
- Redis
  - An in-memory key-value store with rich data model
- CMEM
  - Tencent's pubic in-memory key-value store service
  - Stores several tens of TB of data of online games
- RAMCloud
  - Uses InfiniBand to achieve 10-us level I/O latency
  - Boosts the performance of online data-intensive app

Network Related Challenges for In-Memory Storage

- False failure detection
  - Transient network problems vs. real server failures
- Recovery traffic congestion
  - Thousands of recovery flows bring network congestion which results in long recovery time
- ToR switch failures
  - When a ToR switch fails, all its servers are considered dead and several TB of data may need to be recovered

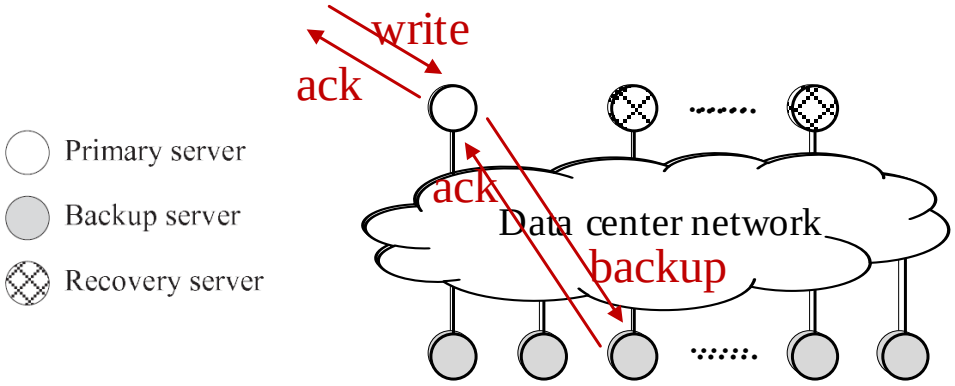
Solution: CubicRing

- One-hop failure detection
  - Shorten the paths that heartbeats have to traverse
- Avoid traffic congestion
  - Restrict the recovery traffic within the smallest possible range
- Avoid single failure for ToR switch
  - Build the in-memory storage system on a multi-homed cubic topology

- **Structure**
- Failure Recovery
- Evaluation

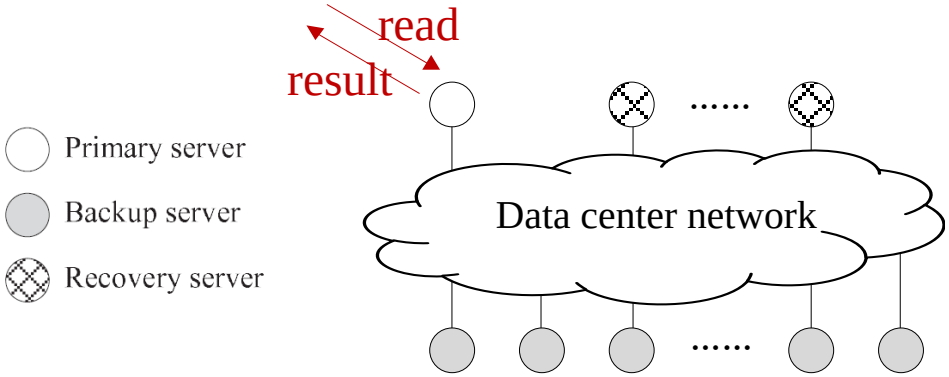
Primary-Recovery-Backup

- Primary-recovery-backup
  - Only one primary copy is stored in the RAM
  - Redundant backup copies are stored on disks
    - Fast recovery requires 10+ GB aggregate recovery throughput
  - Also adopted by RAMCloud
  - But cannot handle the network-related challenges



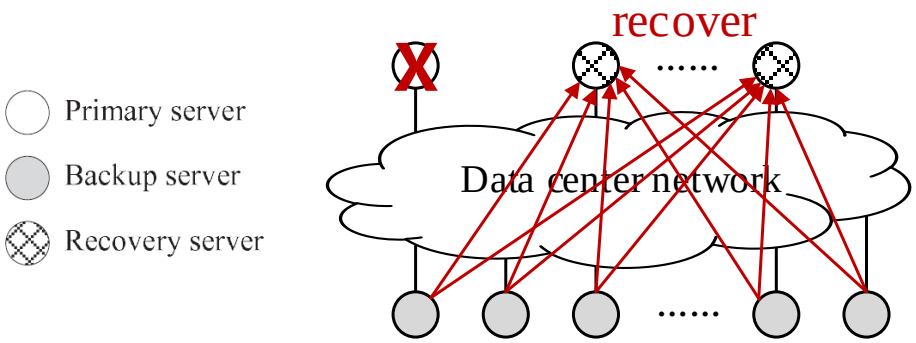
Primary-Recovery-Backup

- Primary-recovery-backup
  - Only one primary copy is stored in the RAM
  - Redundant backup copies are stored on disks
    - Fast recovery requires 10+ GB aggregate recovery throughput
  - Also adopted by RAMCloud
  - But cannot handle the network-related challenges



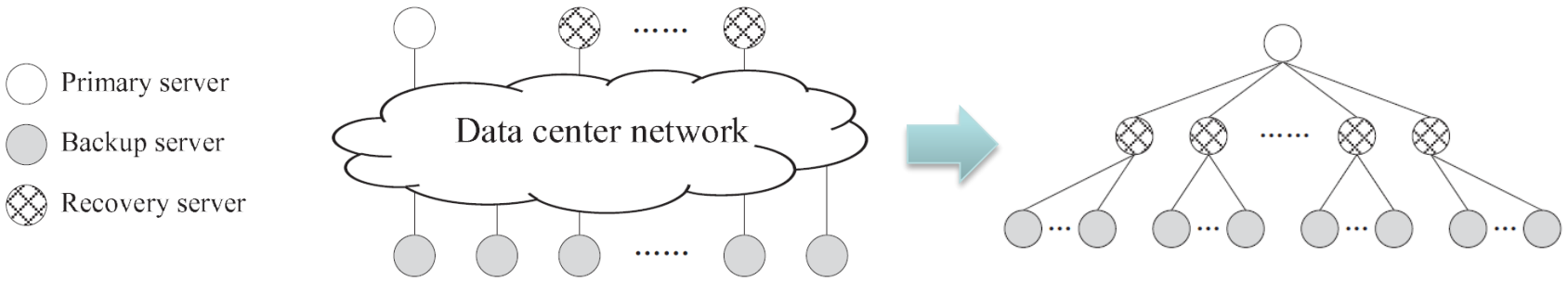
Primary-Recovery-Backup

- Primary-recovery-backup
  - Only one primary copy is stored in the RAM
  - Redundant backup copies are stored on disks
    - Fast recovery requires 10+ GB aggregate recovery throughput
  - Also adopted by RAMCloud
  - But cannot handle the network-related challenges



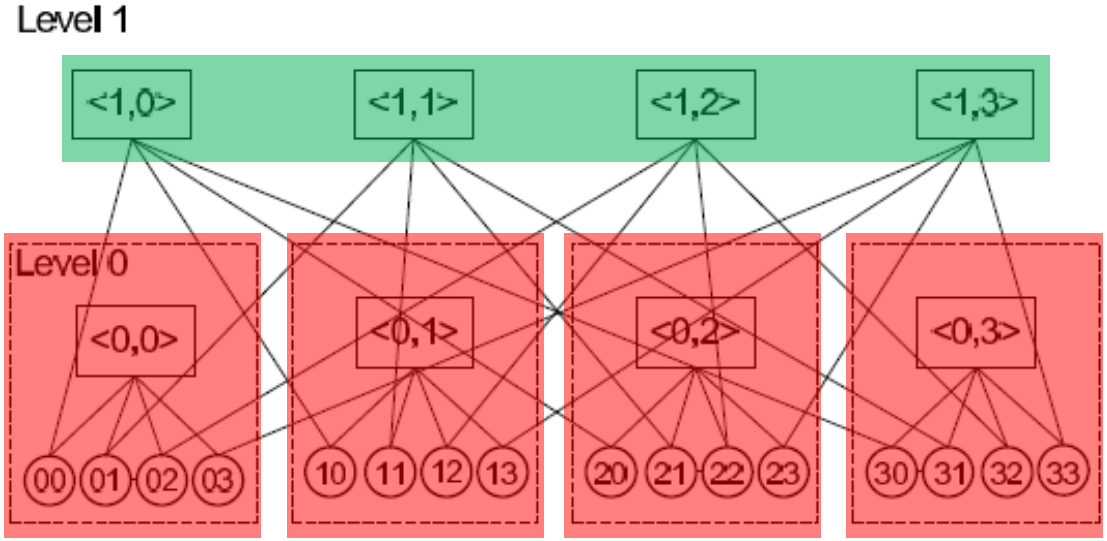
Directly-Connected-Tree

- Basic idea
  - ✓Restrict failure detection and recovery traffic within the *smallest* possible range (i.e. **1-hop**)
- Exploiting DCN proximity
  - ✓Form a directly-connected-tree
  - ✓Primary-recovery: 1-hop detection
  - ✓Recovery-backup: 1-hop recovery



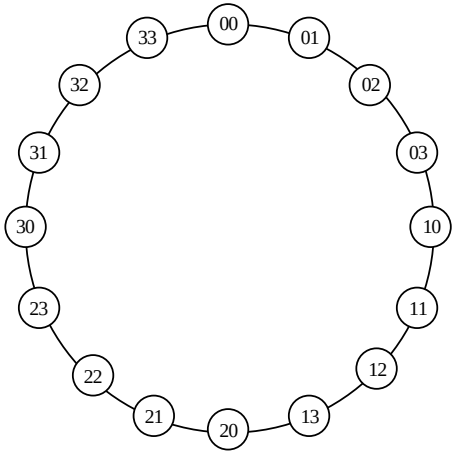
From Tree to Cube

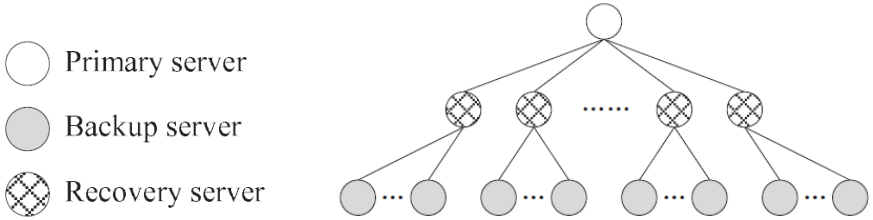
- Embed the trees into cubic DCN
  - ✓Each server equally plays all the three roles
- Generalized hypercube
  - ✓Each vertex can be viewed as the root of a tree
- BCube is a recursively defined network
  - ✓A BCube(4,1) is constructed from 4 BCube(4,0) and 4 4-port switches

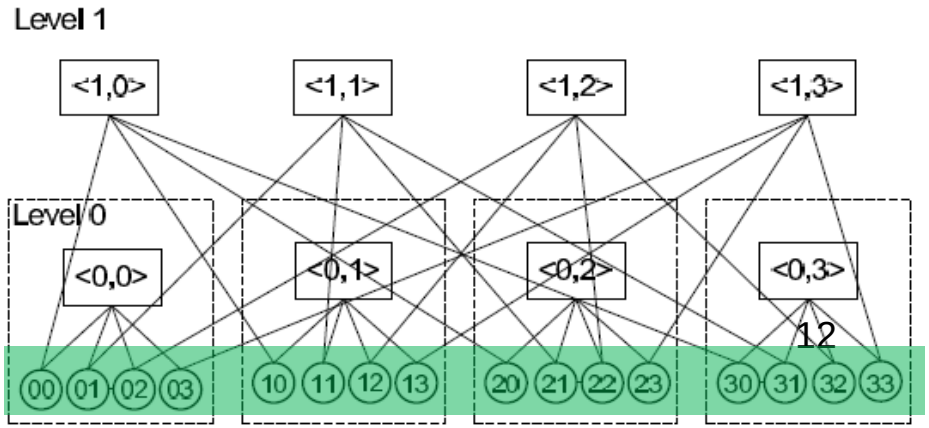


CubicRing on BCube

- CubicRing
  - Three layer of rings: primary ring, recovery ring, backup ring
- **Primary ring**
  - All servers in BCube are *primary servers*.
  - The whole key space is mapped into the primary ring

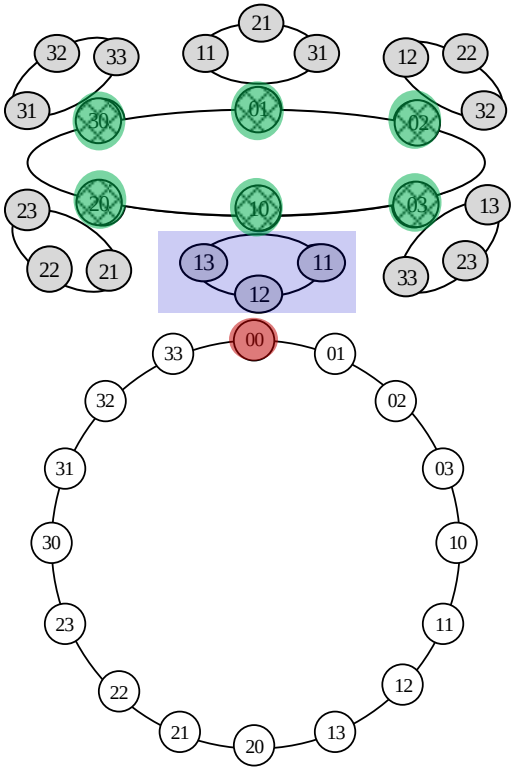


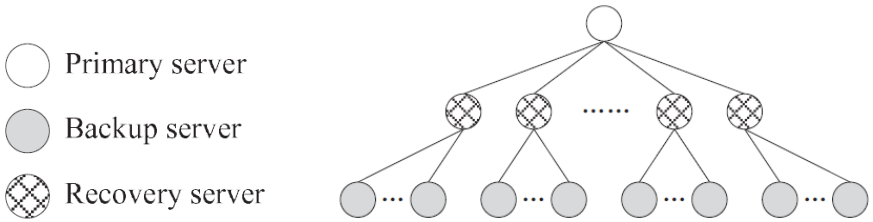


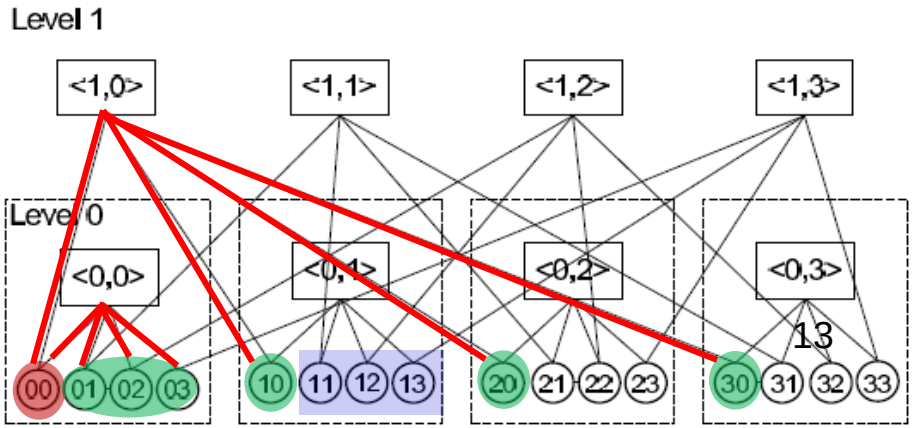


CubicRing on BCube(cont.)

- **Recovery ring**
  - *Recovery servers* of P are 1-hop to *P*
- **Backup ring**
  - *Backup servers* 1-hop to *R* and 2-hop to *P*

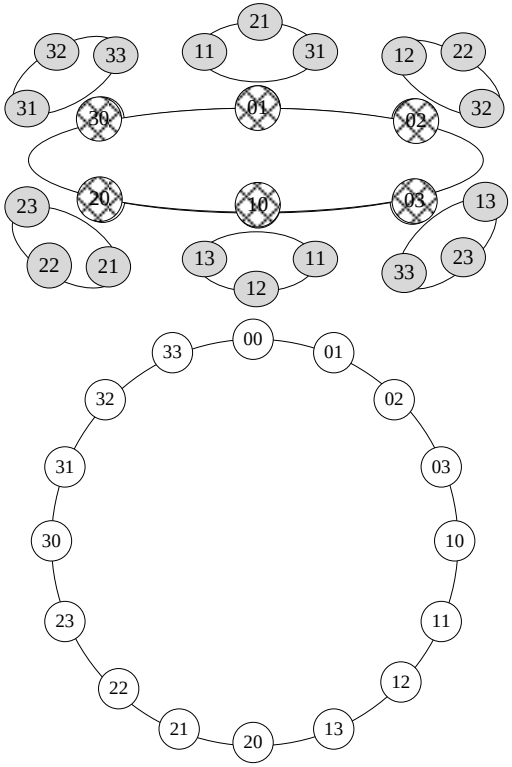


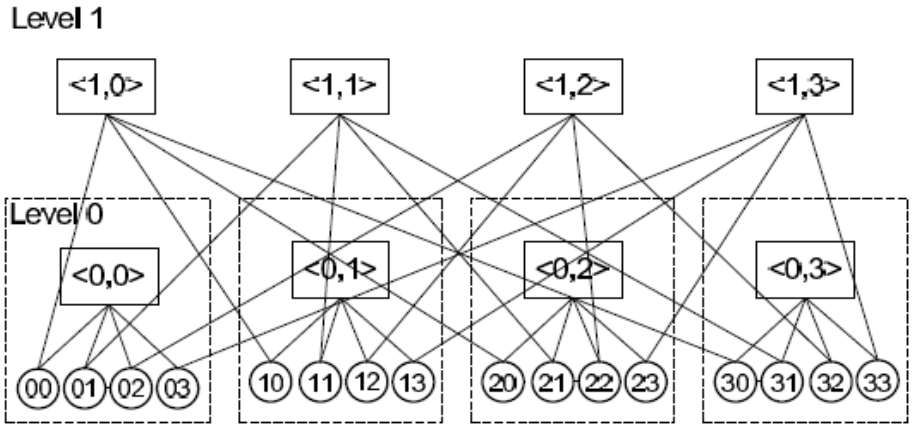




CubicRing Property

- CubicRing for BCube(n, k)
  - # of primary servers
    - $P = n^{k+1}$
  - # of recovery servers for each primary server
    - $R = (n-1)(k+1)$
  - # of backup servers for each primary server
    - $B = (n-1)^2k(k+1)/2$
- CubicRing has plenty of primary, recovery and backup servers
  - BCube(16,2),
    - P = 4096, R = 45, B = 675

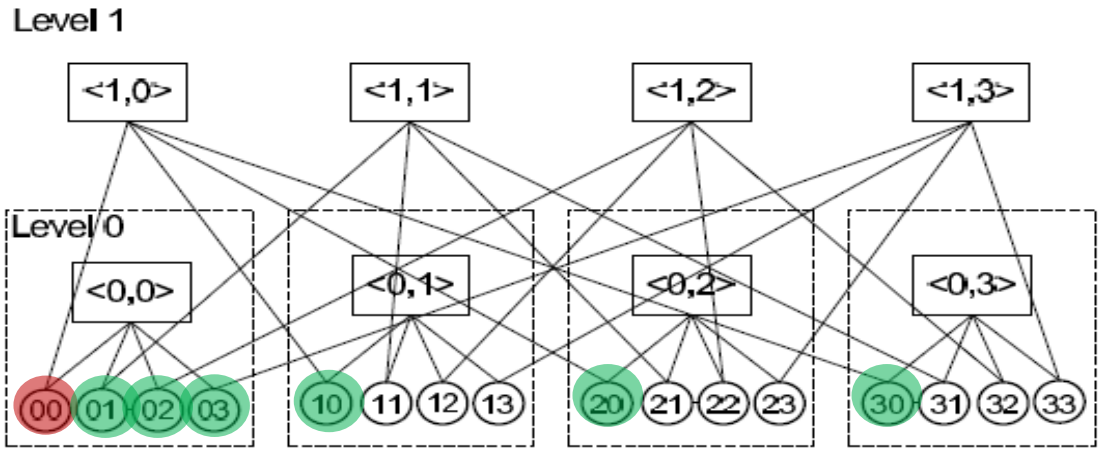




- Structure
- **Failure Recovery**
- Evaluation

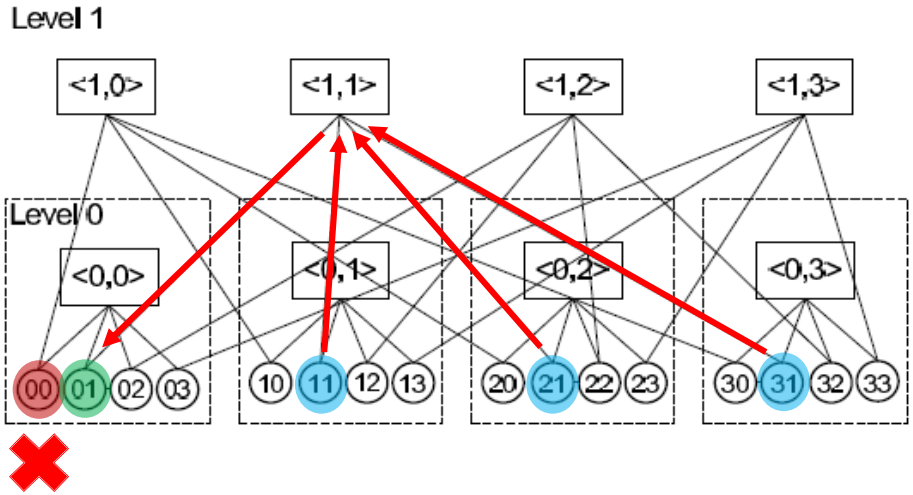
Failure Detection

- Heartbeats
  - Primary servers periodically send heartbeats to their recovery servers
- Confirmation of server failure
  - If a recovery server does not receive heartbeats, it will report this server failure to a coordinator
  - The coordinator verifies the server failure



Failure Recovery

- Primary server failure



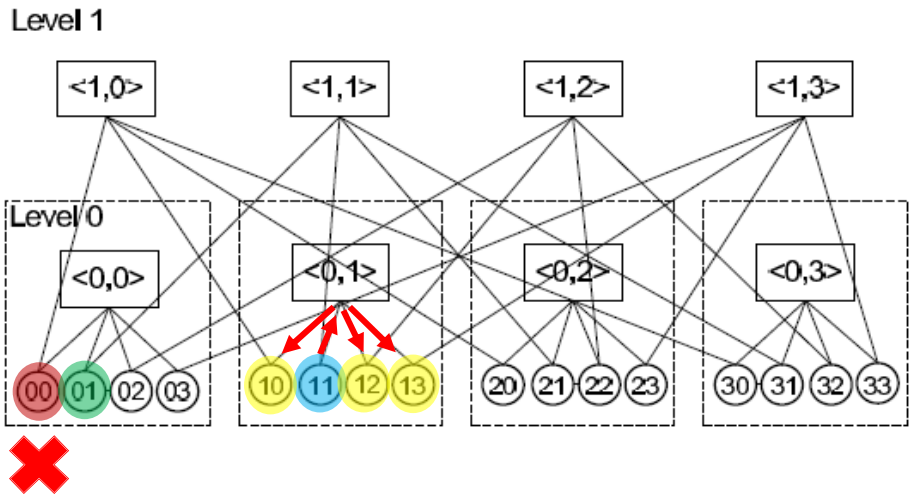
Pseudocode 1: Single server failure recovery

```
1: procedure RECOVERFAILURES(FailedServer F)
2:     Pause relevant services
3:     Reconstruct key space mapping of F
4:     Recover primary data for primary server failure*
5:     Recover backup data for primary server failure*
6:     Resume relevant services
7:     Recover from recovery server failure*
8:     Recover from backup server failure
9: end procedure
```

All the recoveries with * are performed concurrently.

Failure Recovery

- Primary server failure



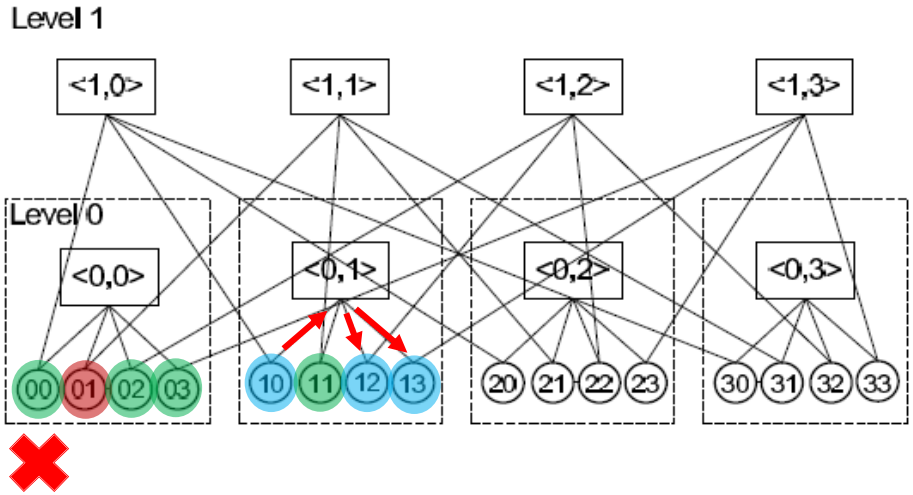
Pseudocode 1: Single server failure recovery

```
1: procedure RECOVERFAILURES(FailedServer F)
2:     Pause relevant services
3:     Reconstruct key space mapping of F
4:     Recover primary data for primary server failure*
5:     Recover backup data for primary server failure*
6:     Resume relevant services
7:     Recover from recovery server failure*
8:     Recover from backup server failure
9: end procedure
```

All the recoveries with * are performed concurrently.

Failure Recovery

- Recovery server failure



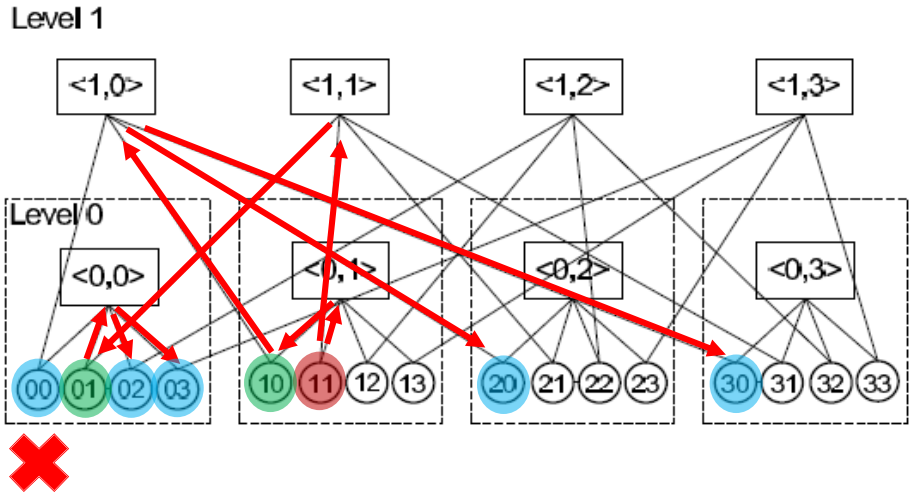
Pseudocode 1: Single server failure recovery

```
1: procedure RECOVERFAILURES(FailedServer F)
2:     Pause relevant services
3:     Reconstruct key space mapping of F
4:     Recover primary data for primary server failure*
5:     Recover backup data for primary server failure*
6:     Resume relevant services
7:     Recover from recovery server failure*
8:     Recover from backup server failure
9: end procedure
```

All the recoveries with * are performed concurrently.

Failure Recovery

- Backup server failure



Pseudocode 1: Single server failure recovery

```
1: procedure RECOVERFAILURES(FailedServer F)
2:     Pause relevant services
3:     Reconstruct key space mapping of F
4:     Recover primary data for primary server failure*
5:     Recover backup data for primary server failure*
6:     Resume relevant services
7:     Recover from recovery server failure*
8:     Recover from backup server failure
9: end procedure
```

All the recoveries with * are performed concurrently.

Single Server Failure Recovery

- Summarization
  - Most of recoveries are 1-hop
  - Concurrent recoveries have little contention

Recovery type	Size[1]	From/to[2]	Length	# flows[3]
Primary data of primary server	α	B→R	1-hop	br
Backup data of primary server	α	B→B B→R	1-hop	b^2r
Recovery server	$< \alpha$	R→B	1-hop	$(b-1)br$
Backup server	$f\alpha$	B→R	2-hop	$f(b-1)br$

[1] Total recovered size (assume a primary server stores α primary data).

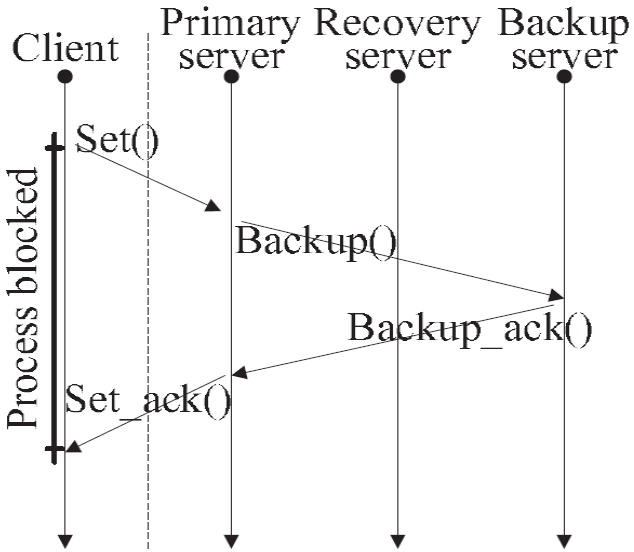
[2] From the perspective of a failed *primary server*. R: recovery server. B: backup server. Bottleneck is R's inbound network bandwidth.

[3] # flows after the 1st failure. b: # backup servers on the backup ring. r: # recovery servers on the recovery ring. f: disk replication factor.

- Structure
- Failure Recovery
- **Evaluation**

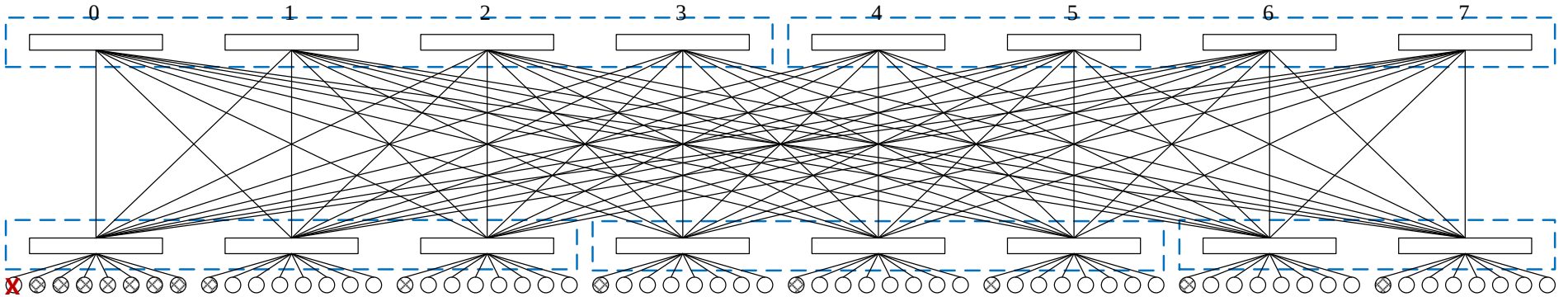
Implementation

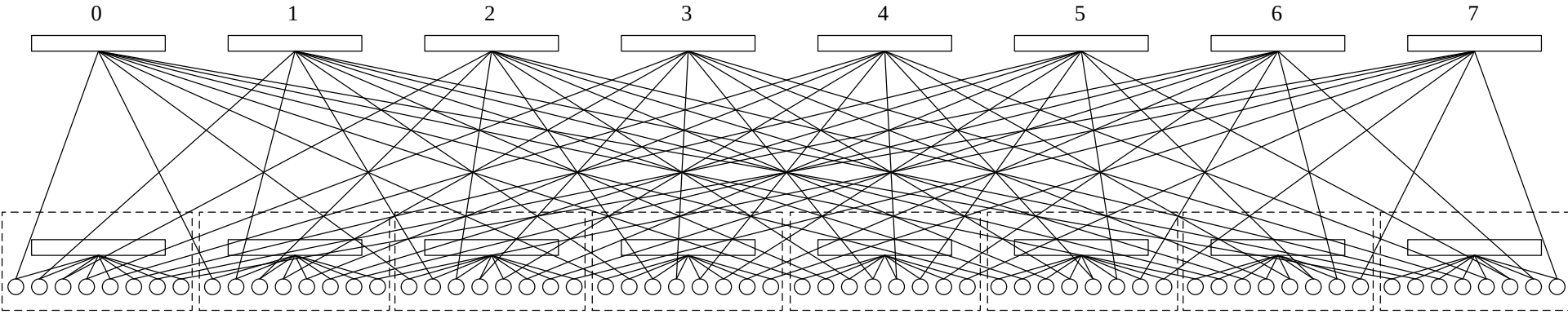
- MemCube
  - Memcached-1.4.15
  - Linux (CentOS 6.4)

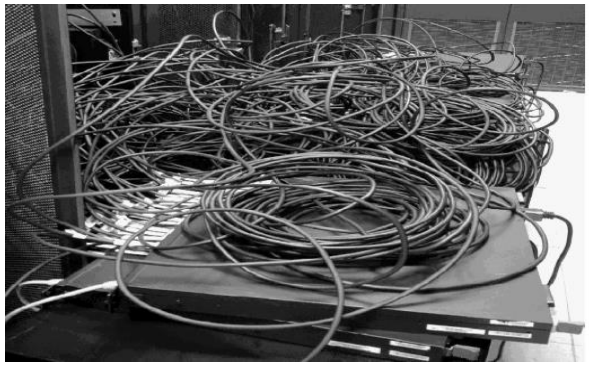


- MemCube components
  - Connection manager: Maintains the status of neighbors and interacts with other servers
  - Storage manager: Handles RAM I/O requests and asynchronously writes backup data to disks
  - Recovery manager: Reconstructs primary/backup data on the new primary/backup servers

Testbed

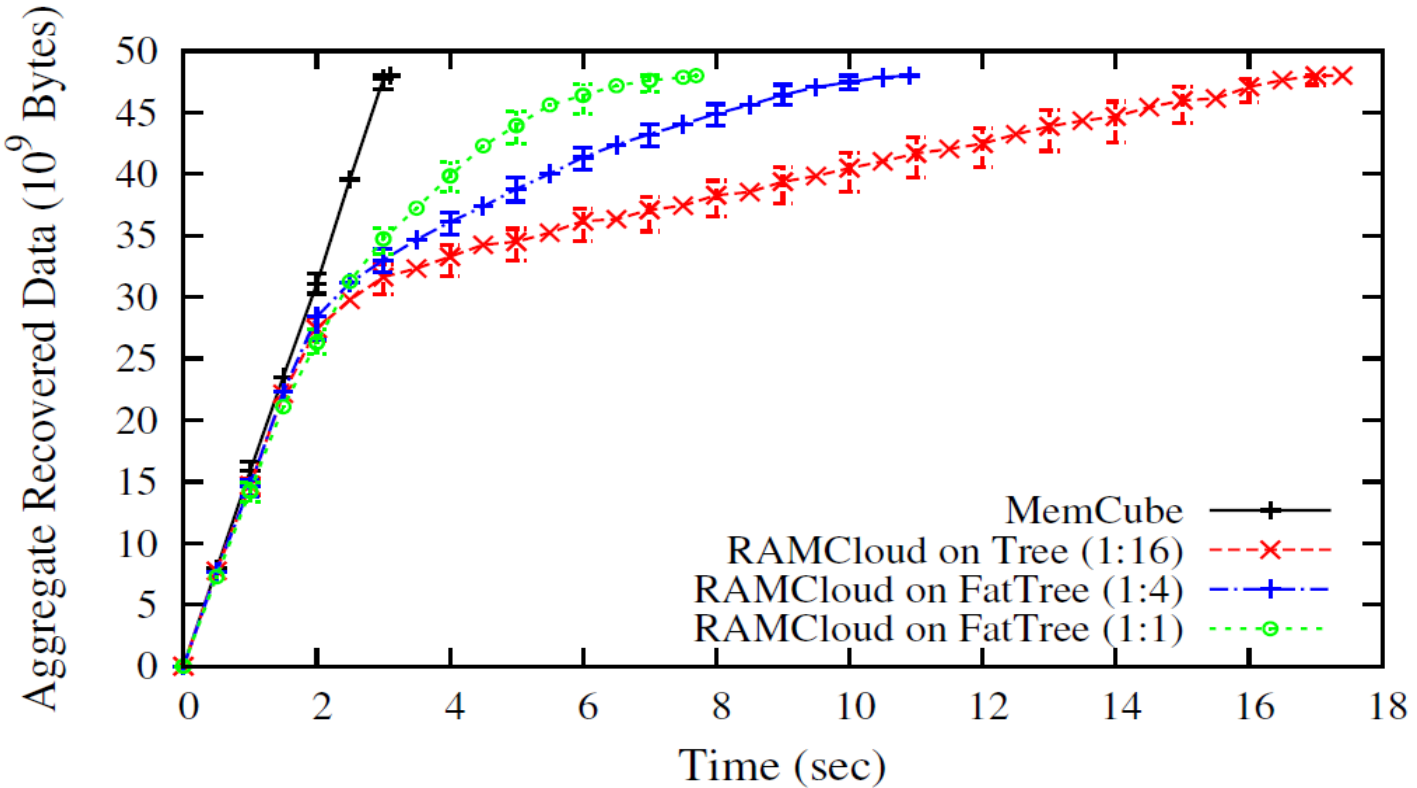

0
1
2
3
4
5
6
7
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63


0
1
2
3
4
5
6
7



Server Failure Recovery

- Recovers 48 GB of data in 3.1 seconds
- Aggregate recovery throughput: 123.9 Gb/sec
- 88.5% of the ideal aggregate bandwidth



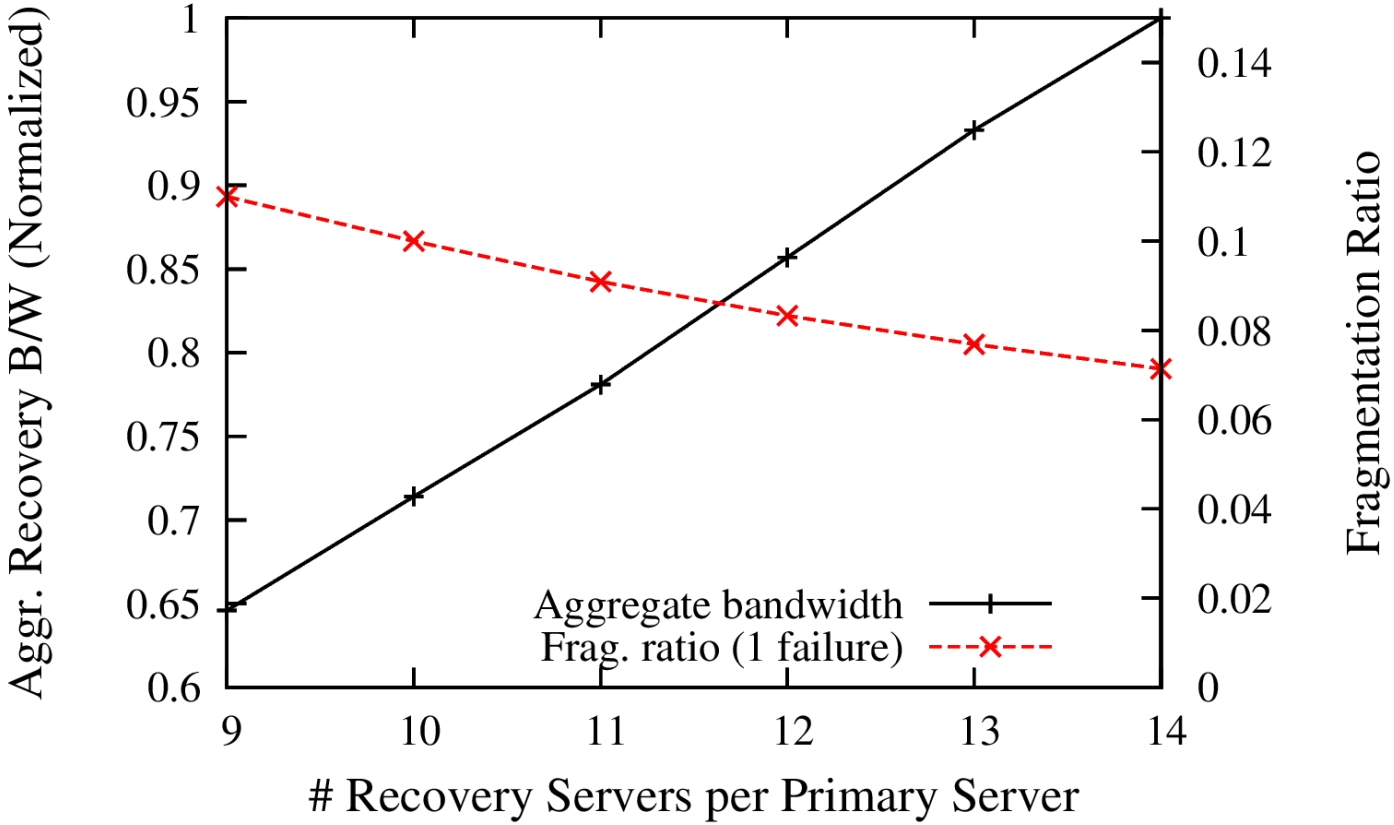
Server Failure Recovery

- Simulations

(*n*,*k*) # servers	**(4,3)** 256	**(8,2)** 512	**(8,3)** 4096	**(16,1)** 256	**(16,2)** 4096
MemCube	1.20	1.01	0.51	1.44	0.48
RAMCloud	26.59	42.98	83.40	22.68	40.59

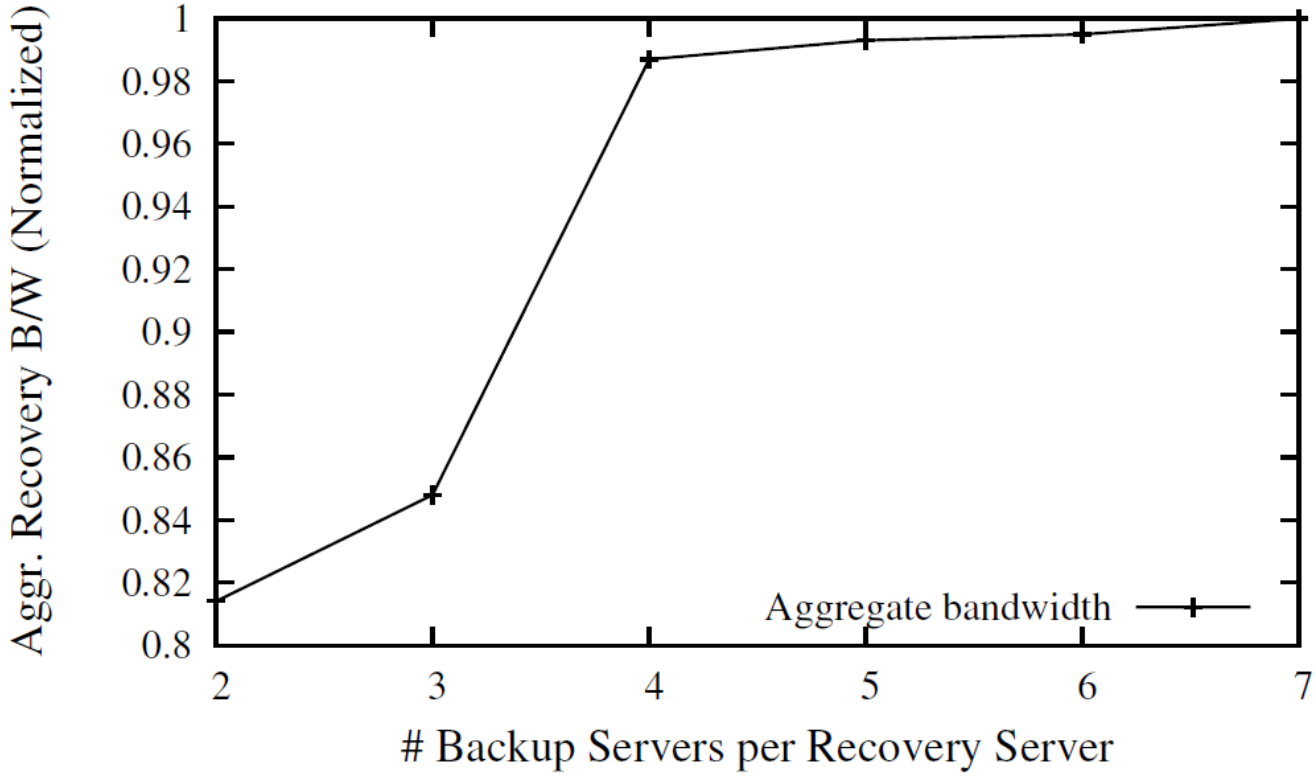
Different # Recovery Servers

- Recovery bandwidth vs. fragmentation
  - ✓More recovery servers result in higher throughput
  - ✓And higher fragmentation



Different # Backup Servers

- Recovery bandwidth
  - ✓When # backup servers is small, their aggregate bandwidth may become the bottleneck



Related Work

- In-memory storage
  - Redis
  - CMEM
  - **RAMCloud**
- Failure detection and recovery
  - Phi-accrual detector
  - Falcon and Pigeon
  - Host failure recovery (e.g., microreboot)
  - Flat Datacenter Storage
    - Locality-oblivious parallel recovery

Conclusion

- CubicRing
  - Exploits network proximity to restrict failure detection and recovery within 1-hop
- MemCube: in-memory key-value store
  - Leverages the CubicRing structure for fast failure detection and recovery
  - Maintains the CubicRing structure against failures

Q & A