CRaft: An Erasure-coding-supported Version of Raft for Reducing Storage Cost and Network Cost

Zizhong Wang, Tongliang Li, Haixia Wang, Airan Shao, Yunren Bai, Shangming Cai, Zihan Xu, and Dongsheng Wang

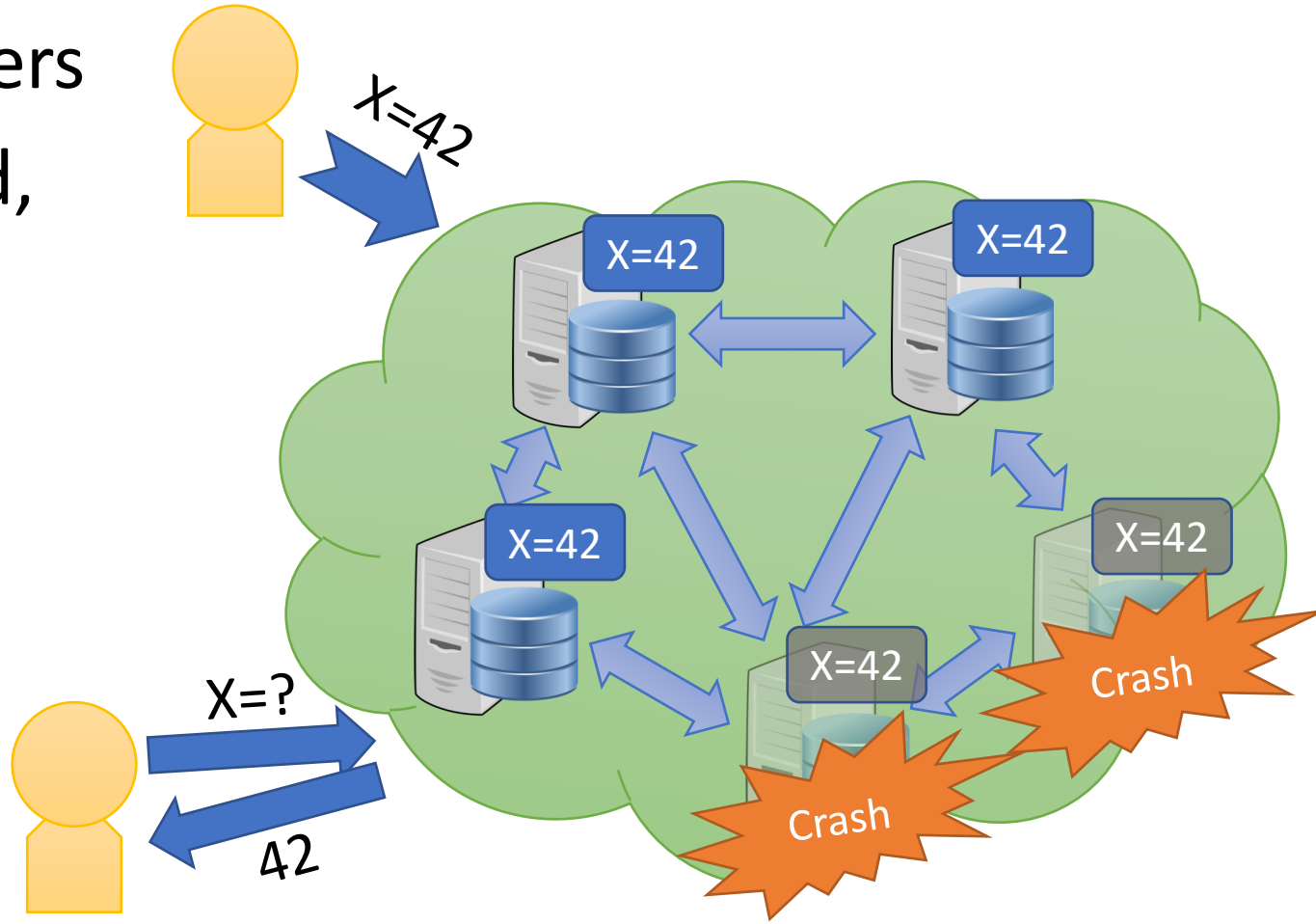
Tsinghua University

Presented by: Xiaoqi Chen

FAST'20, Feb 27th 2020 @ Santa Clara

Data replication: when server crashes…

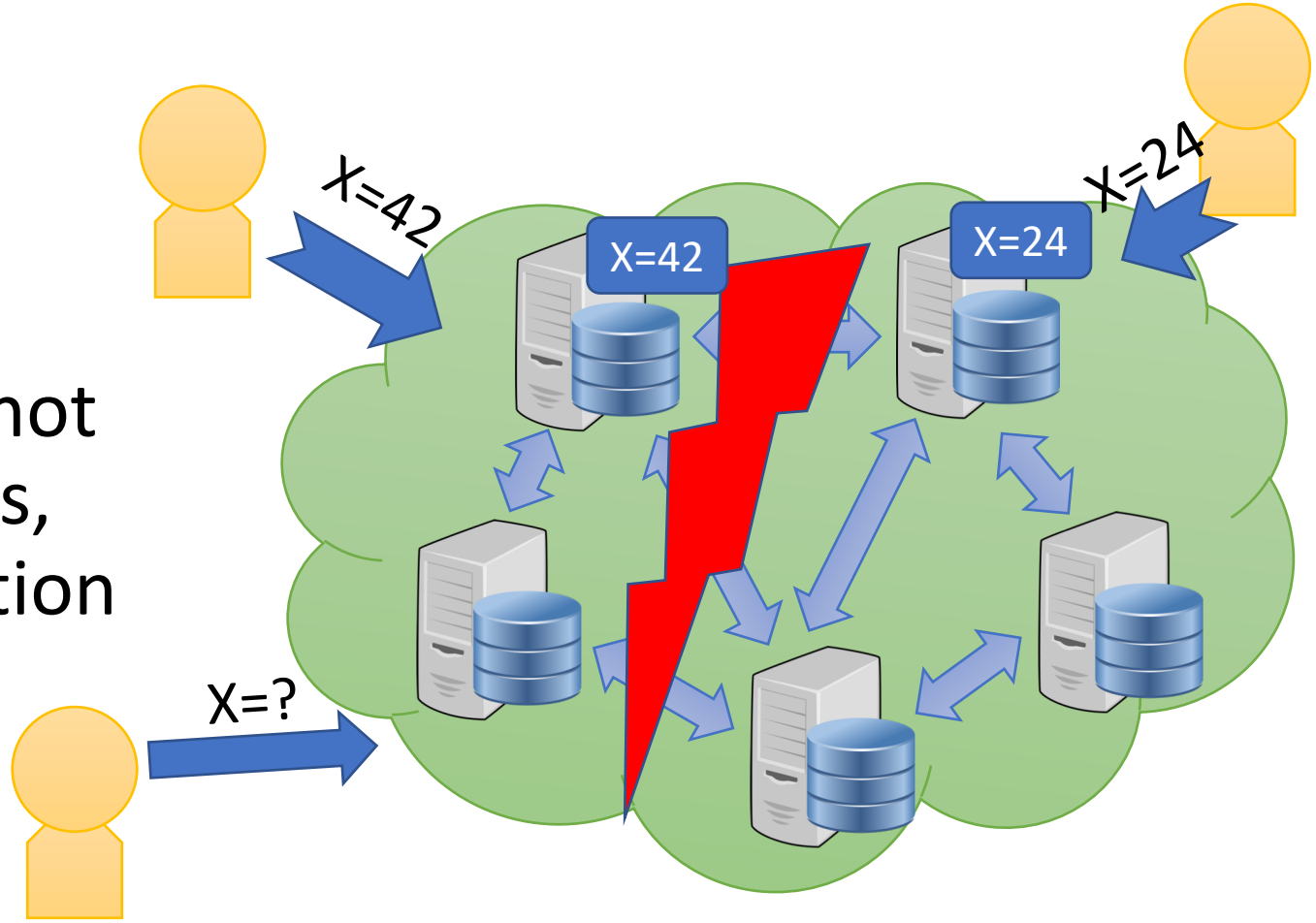
- User data written to **N** servers
- When any **F** servers crashed, we still have other copies



Consensus between servers…

- Network may partition…
- Need consensus protocol!

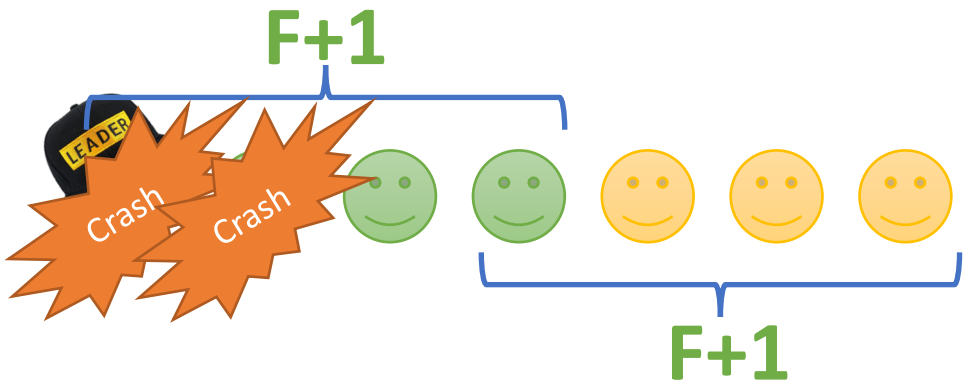
- Observation: **N**=2**F** servers not enough to tolerate **F** crashes, can't distinguish from partition



Raft: distributed consensus

- Use **N=2F+1** servers to tolerate **F** crashes
- A "leader" is elected by at least **F+1** "followers"
  - Data is considered "committed" when stored **F+1** copies

N=7, **F**=3

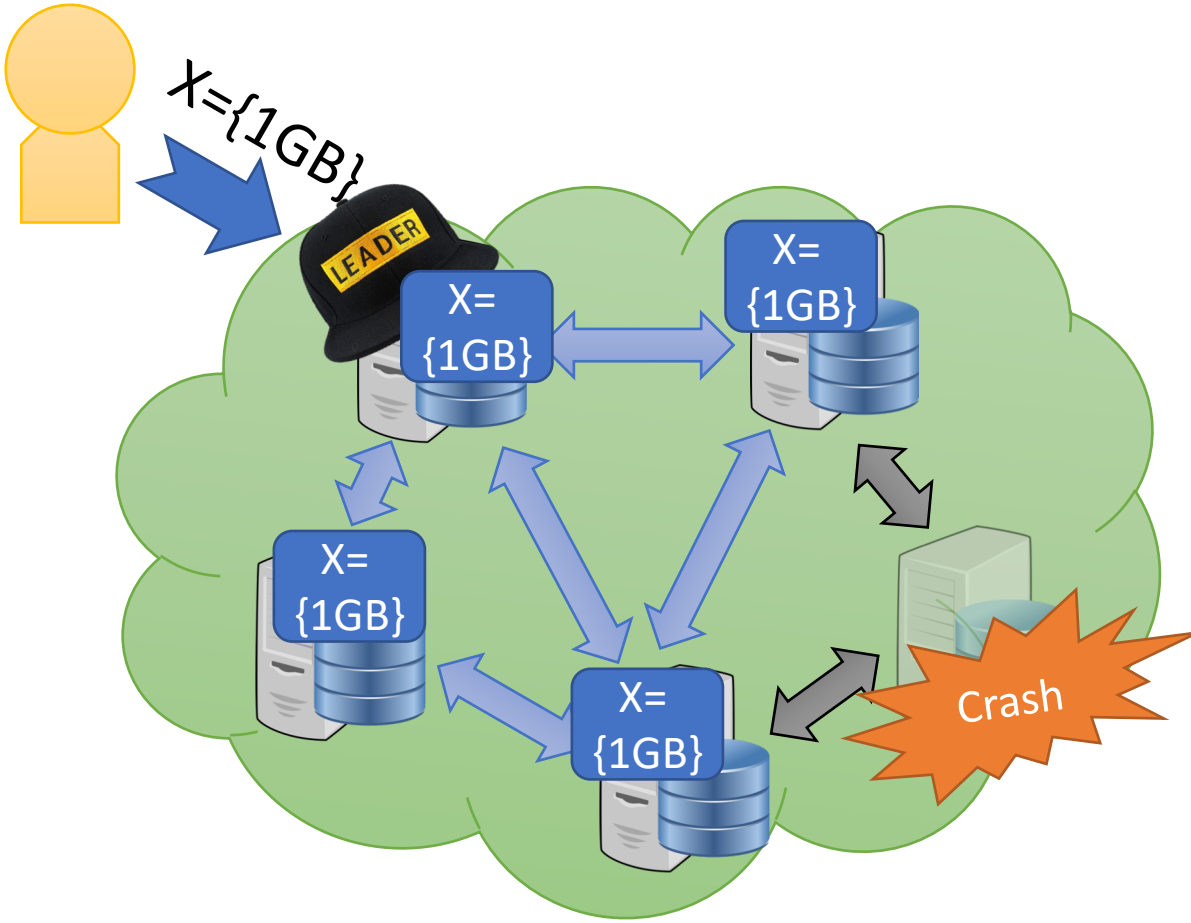


- Leader crash: new election, reconciliation
  - New leader's **F+1** followers has at least 1 in common

Problem: data copied **N** times…

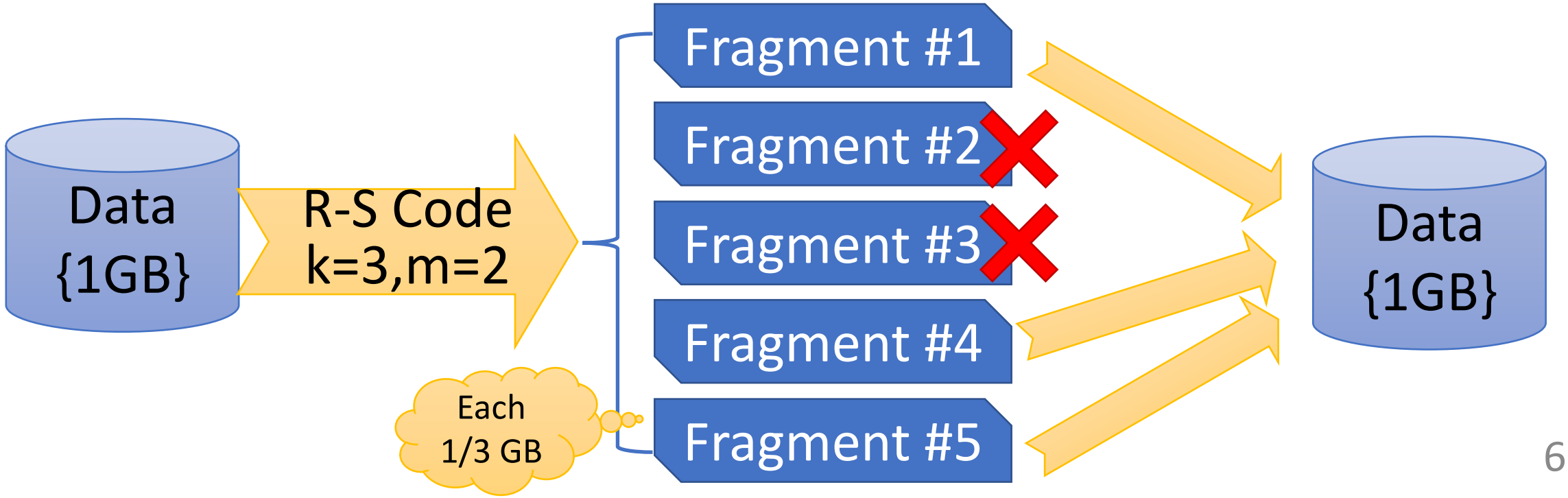
- Replication is costly
  - **N** times **network traffic**!
  - **N** times **storage space**!

- **Can we do better?**



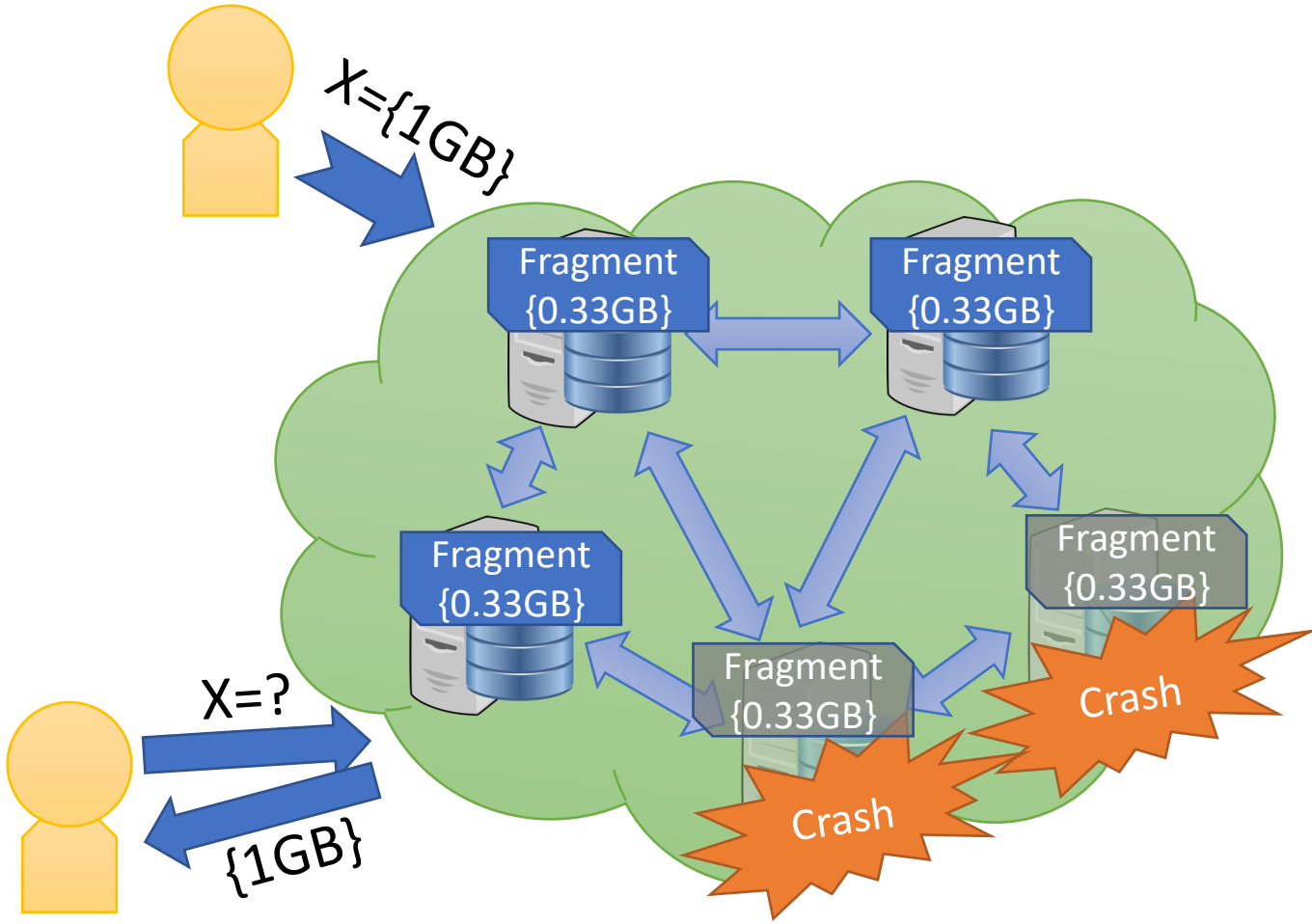
Erasure Coding: Reed-Solomon Code

- **(k,m)**-Reed Solomon code:
  - Data split into **(k+m)** fragments, each **1/k** size
  - Any **k** fragments can recover original data



Combine erasure coding with crash tolerance?

- We already used **N=2F+1=5** servers for fault tolerance
- Correctness requires **F+1=3** healthy servers.
- Why not just recover data from **3 out of 5** coded fragments?

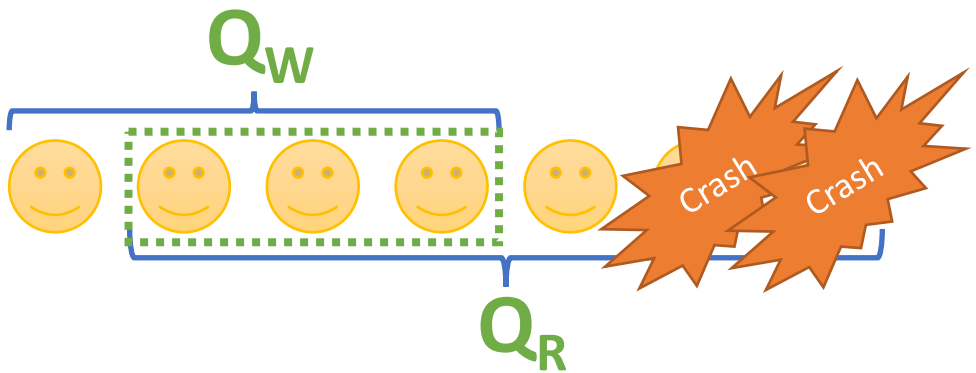


Prior work: RS-Paxos

- Use **(k,m)** Reed-Solomon code
  - Write is committed after written to Q_W servers
  - Read requires reading from Q_R servers for consensus
  - $Q_W + Q_R - N \geq k$ (at least **k** in common, allows decoding)

N=7, **k**=3

Q_W=4, Q_R=6



- **Liveness** issue: when **N=2F+1**, can't tolerate **F** crashes!

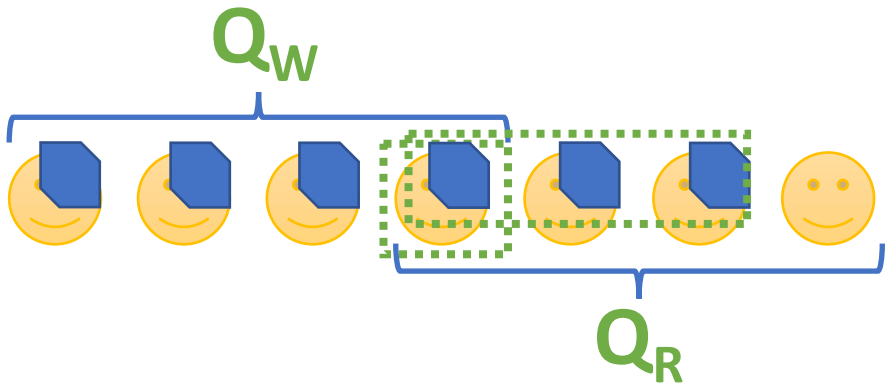
**When Paxos meets erasure code: Reduce network and storage cost in state machine replication.* Shuai Mu et. al., *HPDC'14*

Pando: optimizing RS-Paxos

- Presented on Tuesday's NSDI'20 session
- Shrink the quorum overlap from **k** to 1 to reduce latency:
  - Faster in "normal" case, read from more if observed inconsistency
  - **Liveness** issue persists

N=7, **k**=3

Q_W=4, Q_R=4

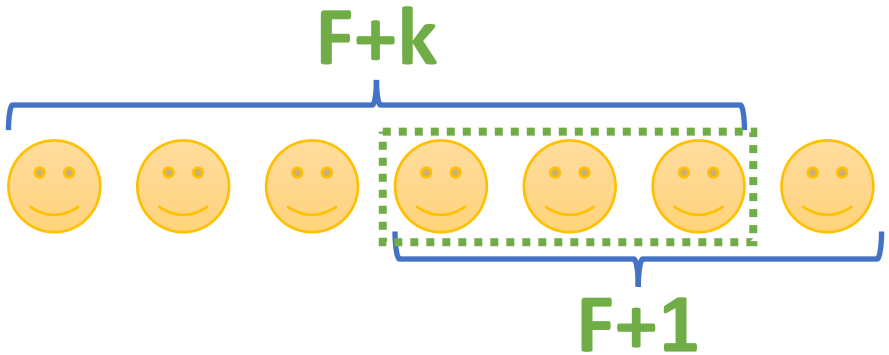


**Near-Optimal Latency Versus Cost Tradeoffs in Geo-Distributed Storage.* Muhammed Uluyol et. al., *NSDI'20*

CRaft: erasure coding + Raft

- Use **N=2F+1** servers to tolerate **F** crashes
- A "leader" is elected by at least **F+1** "followers"
- Leader replicates by sending coded **fragments**
  - Use **(k,m)** Reed-Solomon code, **k+m=N** and **k≤F+1**
  - Data is committed when **(F+k)** fragments are stored, i.e., next elected leader can see at least **k** fragments

N=7, **F**=3, **k**=3



Challenge #1: liveness

- What if there're fewer than **(F+k)** healthy servers?
  - Note: **N=2F+1**, need to tolerate **F** crashes
  - Worst case: only **F+1** healthy servers!

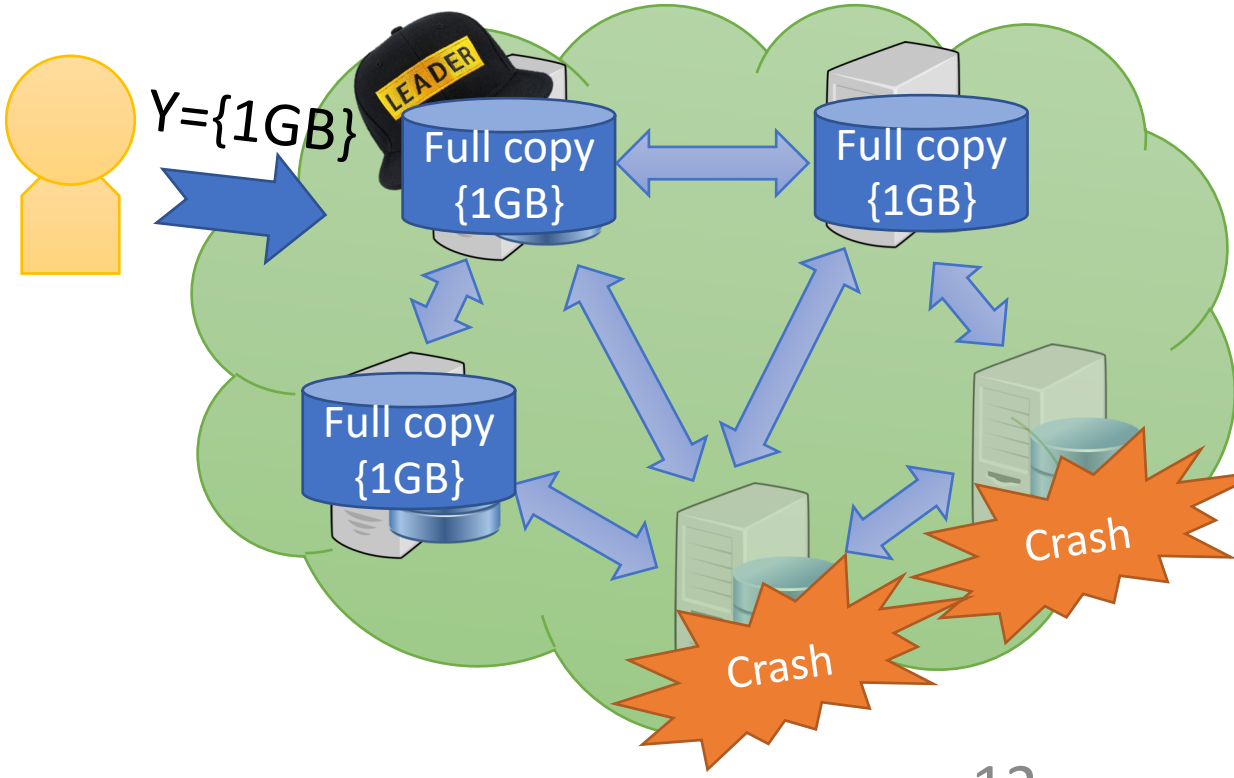
- Solution: full replication when necessary
  - When <**(F+k)** healthy, send full copies (not fragments)
  - "Predict" if there're ≥**(F+k)** healthy using heartbeats

Challenge #1: liveness

- In normal case, replicate coded fragments
- When <(F+k) servers are healthy, send full copies

N=5, **F**=2, **k**=3

F+k=5



Challenge #2: newly-elected leader

- The new leader has coded fragments, not full copies
- For committed entry's fragments:
  - Any **F+1** servers holds at least **k** fragments for the entry, so the new leader can always recover full data by querying its followers
- For entries not yet committed:
  - In the worst case, cannot guarantee recovery…

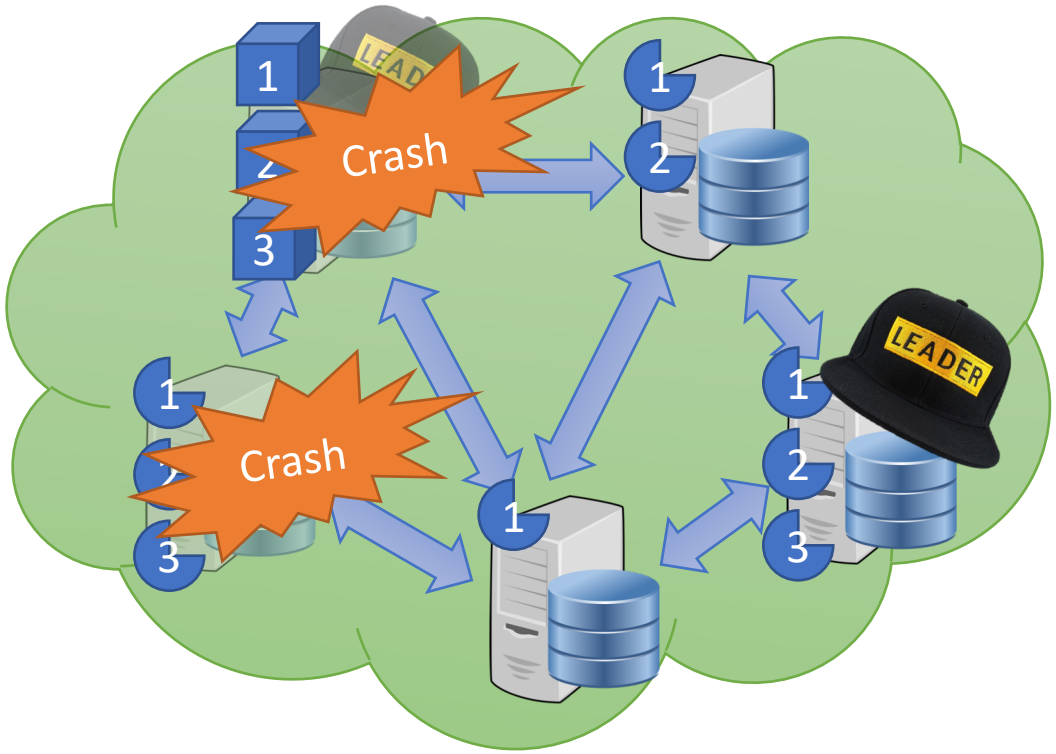
- Solution: a post-election "***LeaderPre***" phase.
  Try recovery (collect fragments), or discard the entry!

Challenge #2: newly-elected leader

- Leader re-election follows normal Raft procedure (requires **F+1** followers)
- In the ***LeaderPre*** phase, new leader tries recovery using data fragments
  - If recovery failed, delete later entries

N=5, **F**=2, **k**=3

F+k=5



CRaft: erasure coding + Raft

- **Efficient**: replicate coded fragments when possible, save network and storage (**1/k** fraction)
- **Liveness**:
  - When there're <**(F+k)** healthy servers, send full copies
  - Newly-elected leader reads uncommitted fragments in the ***LeaderPre*** phase, trying to recover data from fragments
- **Correctness**: if recovery failed in ***LeaderPre***, delete this entry and all following entries does not affect correctness

Comparing consensus protocols

	CRaft	Raft	RS-Paxos
Disk Space & I/O	**2F/k+1**	**2F+1**	**2F/k+1**
Network Throughput	**2F/k**	**2F**	**2F/k**
Liveness (max #crash)	**F**	**F**	**F-(k-1)/2**

Cannot tolerate **F** crashes (**N**=2**F**+1)

Evaluation highlights

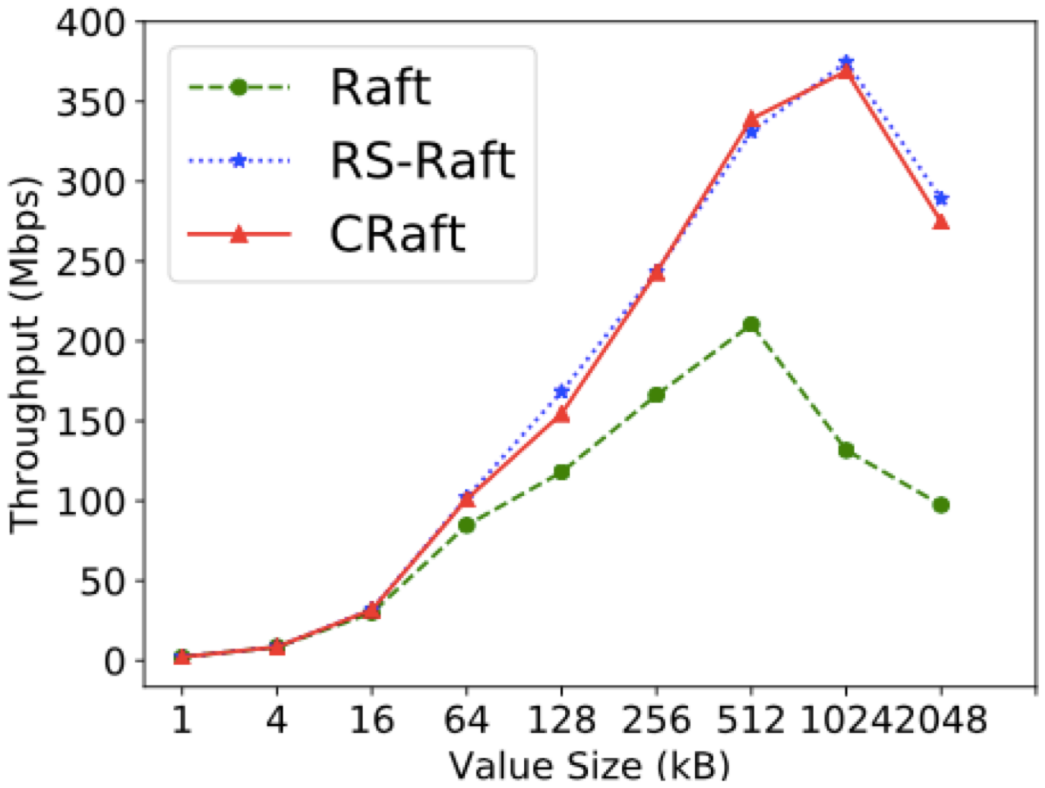
- Can **C**Raft achieve higher throughput than vanilla Raft?
  - Yes! Thanks to the network/storage saving
- Can **C**Raft achieve liveness upon **F** crashes?
  - Yes, **C**Raft degrades into Raft in this case
- What about latency?
  - Write latency: improved (less network send per write)
  - Read latency: new leader's recovery reads are slower

Evaluation setup

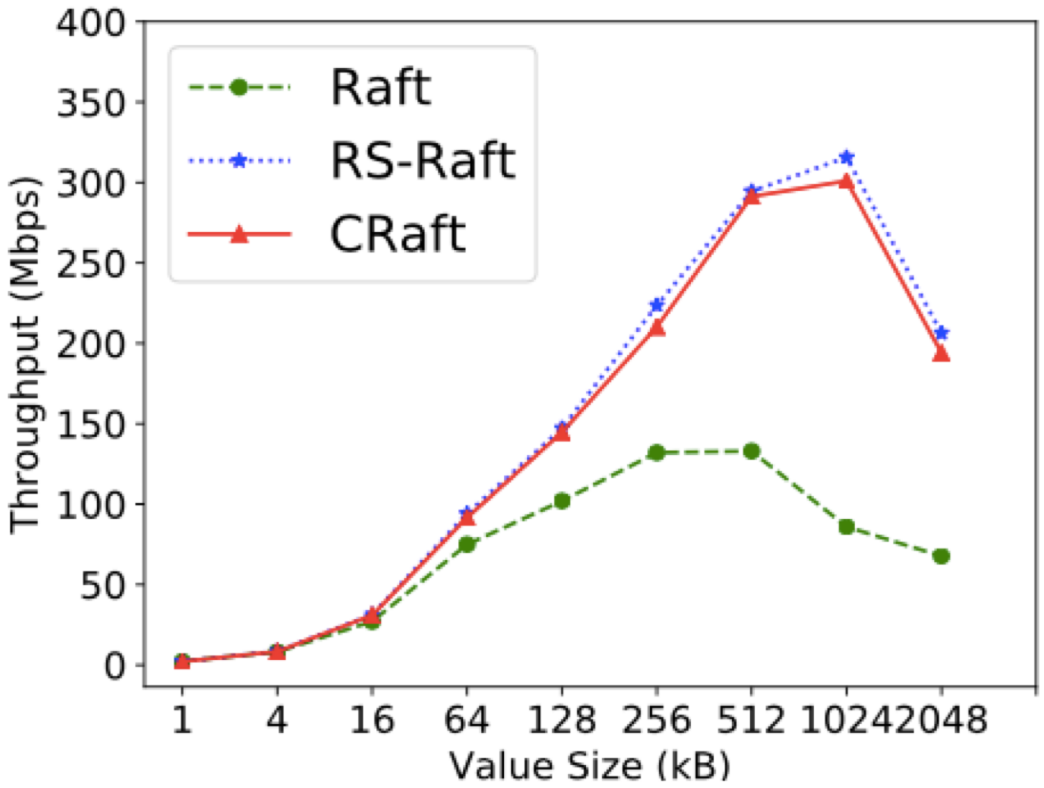
- We implement a Key-Value store application on top of Raft
- Tolerate **F**=2 or 3 crashes, using **N**=5 or 7 servers (**N**=2**F**+1)
- Using Reed-Solomon Code **k**=3, **m**=2 or 4 (satisfying **k**+**m**=**N**)

- Clients run on a separate machine, send read/write requests
- To compare against RS-Paxos, we implement "RS-Raft", using very similar consensus procedure (read/write quorum Q_R/Q_W)

Evaluation: throughput



N=5 servers



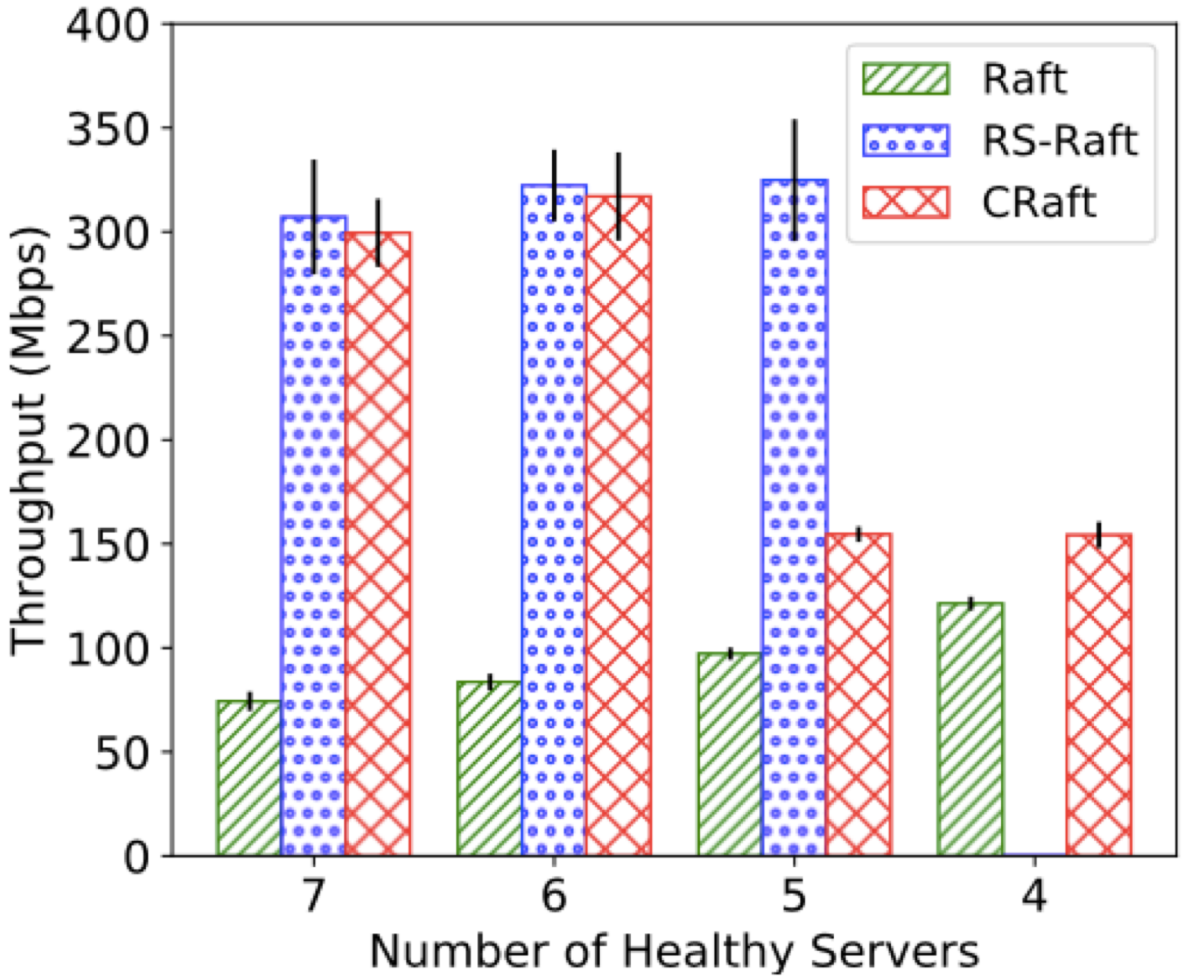
N=7 servers

- CRaft and RS-Raft enjoy network/storage saving

Evaluation: liveness

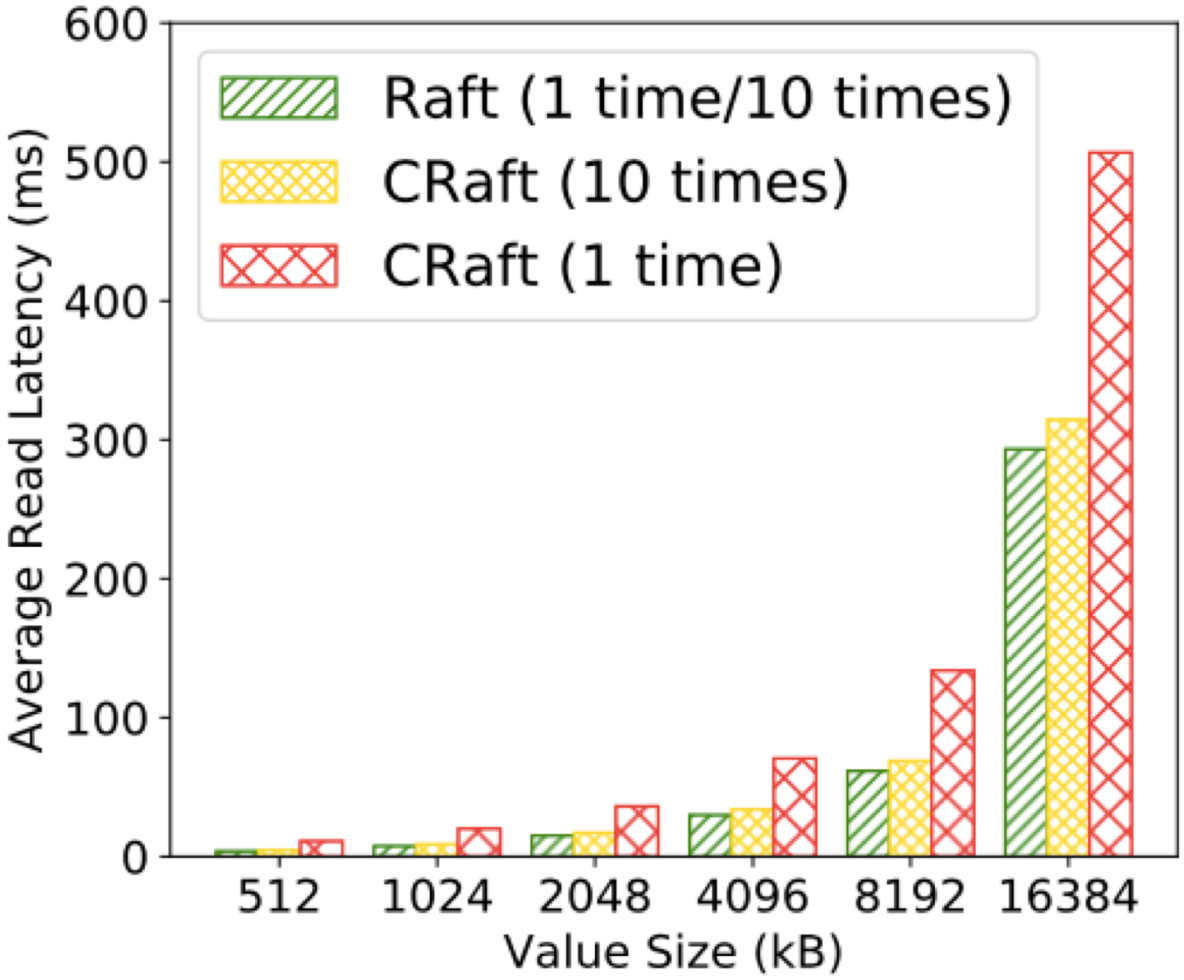
- **k**=3, **F**=3, **N**=2**F**+1=7

- RS-Raft: max(Q_R,Q_W) ≥ 5
- CRaft:
  - Replicate coded fragment when healthy ≥**F**+**k**=6
  - Send full data when <6



Evaluation: read latency

- After leader crash and re-election, first read is slower (data recovery by collecting fragments)
- Subsequent reads have the same latency



Summary: CRaft

- Uses ***erasure coding*** to lower Raft's overhead in network & storage
- Ensures correctness and ***liveness*** upon **F** crashes (with **N=2F+1**)
- Boosts throughput significantly over original Raft