香港中文大學
THE CHINESE UNIVERSITY OF HONG KONG

Parity-Only Caching for Robust Straggler Tolerance

Mi Zhang, Qiuping Wang, Zhirong Shen, Patrick P. C. Lee

The Chinese University of Hong Kong

MSST 2019

Background

- ➢ **Stragglers** exist in large-scale storage systems
  - Nodes operate with slow performance
  - Also known as *gray failures* [Huang, HotOS'17] or *fail-slow failures* [Gunawi, FAST'16]
- ➢ Stragglers introduce latency variation [Dean, Comm.'13]
  - Long tail in latency distribution
- ➢ Hard to pinpoint stragglers [Huang, HotOS'17; Gunawi, FAST'16]
  - Varying root causes
  - Long time to detect

Background

- **Caching** accessed data to bypass stragglers
  - Cache space is limited
  - Only caching popular data inevitably hits stragglers

- **Selective replication** creates more replicas for hot data
  - High redundancy overhead
  - Data popularity can change sharply [Huang, HotNets'14]

Erasure Coding

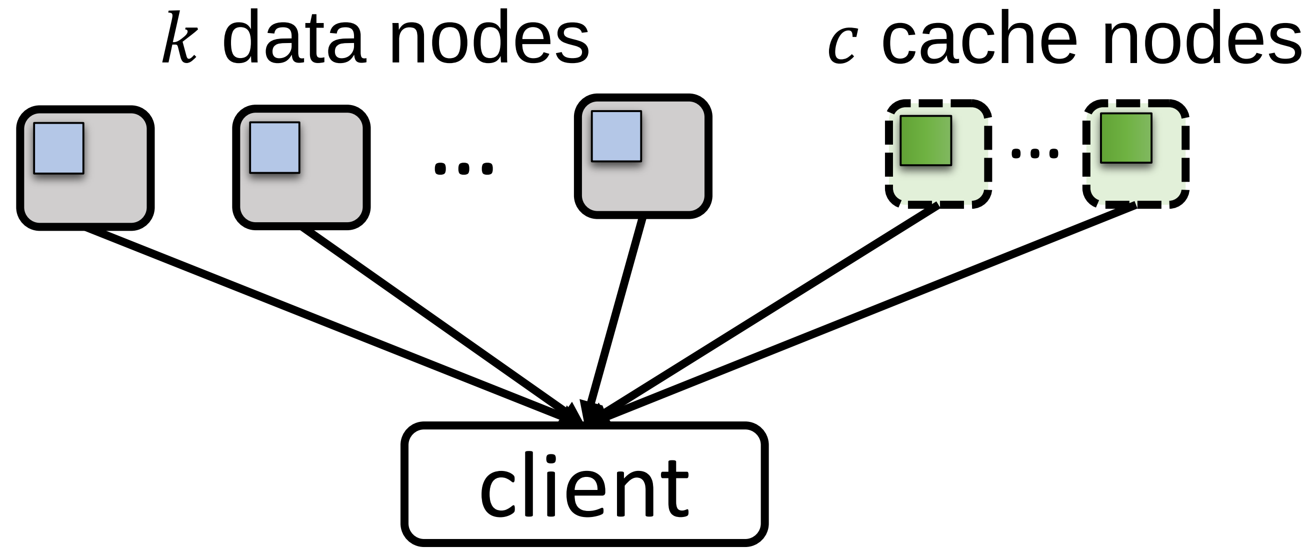
- (k, m) erasure coding
  - Encodes k ***data*** blocks to generate m ***parity*** blocks ($m < k$)
  - Any k blocks suffice to reconstruct original content
- Erasure coding is a promising redundancy technique
  - Storage efficiency
    - Reduce storage overhead from 3x to 1.33x in Azure [Huang, ATC'12]
  - High reliability against fail-stop failures
- Can we combine caching and erasure coding to achieve **robust** straggler tolerance?
  - By robust, we mean straggler tolerance does not rely on accurate detection of stragglers

Our Contributions

- Conduct mathematical analysis
- Design **POCache**, a parity-only caching scheme for robust straggler tolerance
  - Mitigate coding overhead via two mechanisms
  - Straggler-aware cache algorithm
- Implement POCache atop Hadoop 3.1 HDFS
- Evaluate POCache on a local cluster and Amazon EC2
  - Compare POCache with hedge reads and selective replication

Mathematical Analysis

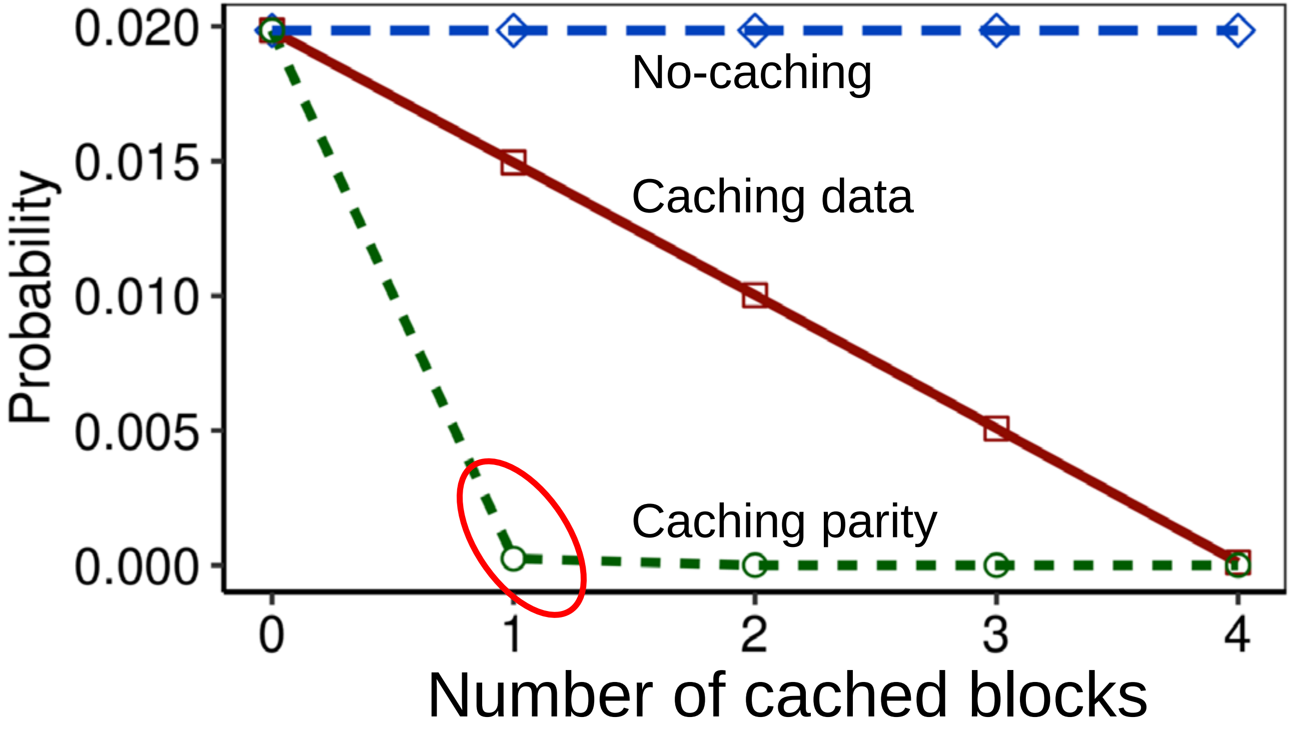
- Retrieve data from k storage nodes



- Assume every node has a probability of 0.5% to become a straggler
- What is the probability of hitting a straggler for a read request?

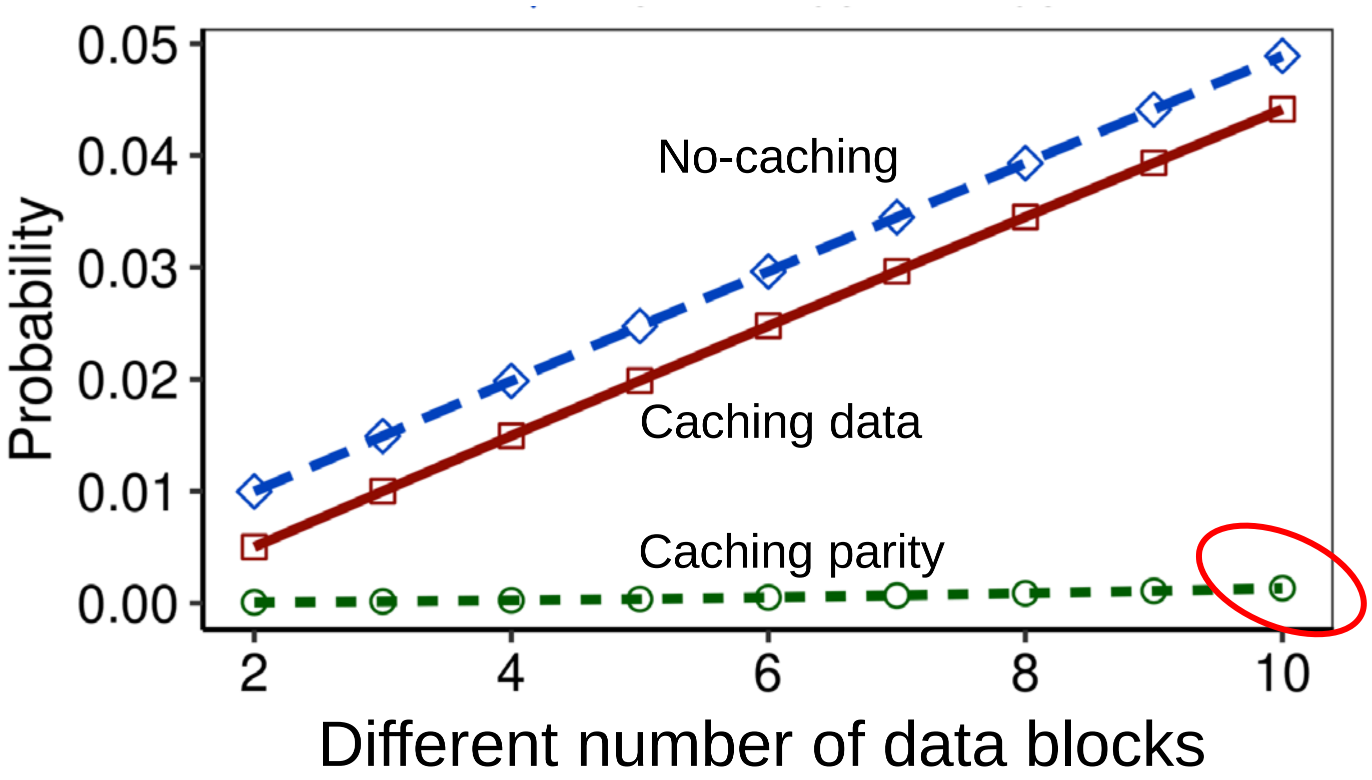
Mathematical Analysis

- Probability of hitting a straggler
  - When reading from $k = 4$ data nodes



Mathematical Analysis

- Probability of hitting a straggler
  - When caching only $c = 1$ block



Main Findings

- Caching parity blocks is more effective than caching data blocks to bypass stragglers

- Caching only one parity block can effectively eliminate the impact of stragglers

- Even with slowdown of cache nodes, caching parity blocks still maintains straggler tolerance

Challenges

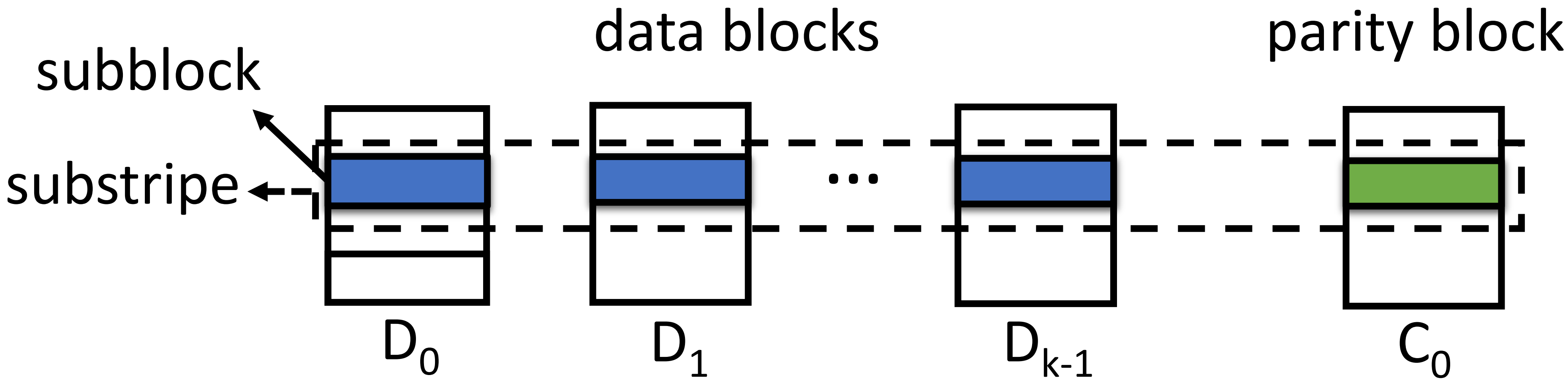
- ➢ Large encoding/decoding overhead
  - Decoding time takes about 30% of read time [Rashmi, OSDI ’16]
  - Degrade normal read/write performance

- ➢ What parity blocks to cache?
  - Manage cache space with considering stragglers

- ➢ Can we support general deployment?
  - Support general storage systems and protocols
  - Support upper-layer applications

POCache Design

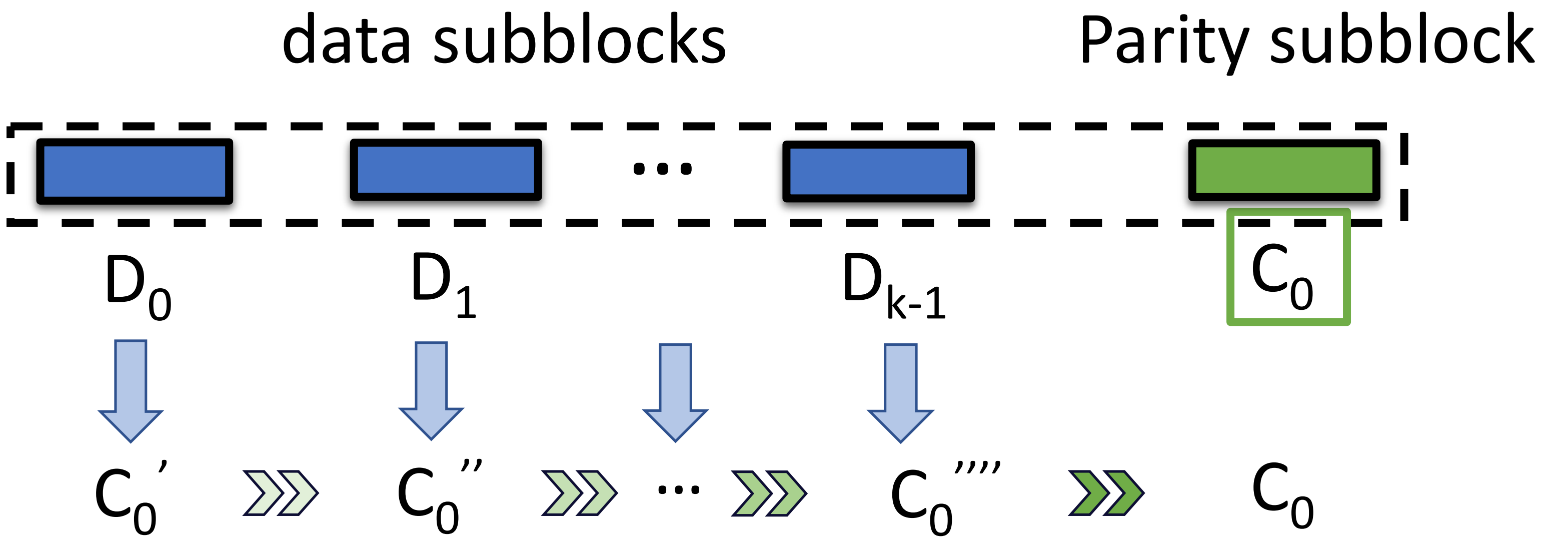
- Block slicing



- Slice blocks into smaller-size subblocks
- Cache parity subblocks rather than the whole block
    - Parallelize coding across different substripes
    - Pipeline the process of caching parity blocks

POCache Design

- ➢ Incremental encoding



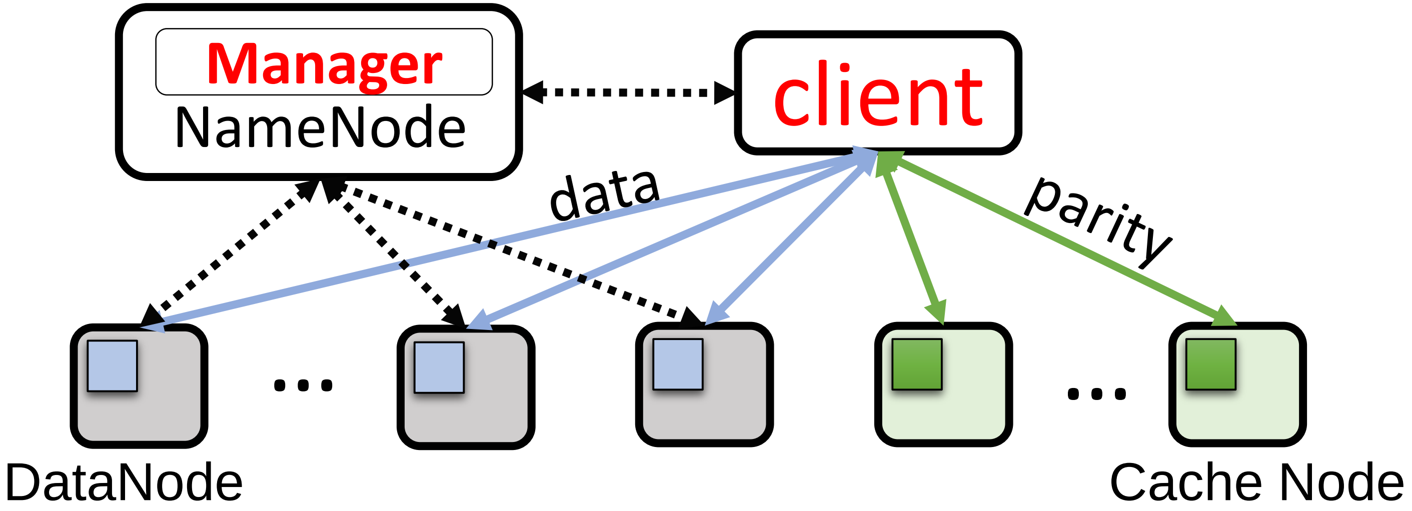
- Addition operations are associative in a linear combination
- Compute parity subblocks one by one incrementally

POCache Design

- How to minimize straggler hit ratio?
    - Avoid accessing stragglers
    - Consider file popularity
- Straggler-aware cache algorithm
    - Admit caching parities for blocks on stragglers
        - Collect each node’s *service rate*
        - Identify stragglers according to *three-sigma rule*
        - Compose a *straggler list*
    - Evict least-recently-accessed files
        - 75% of re-accesses occur within 6 hours [Chen, VLDB’12]
- Details referred to the paper

Implementation

- Implement POCache on Hadoop 3.1 HDFS
  - Use Redis to build cache nodes
  - Add Manager on NameNode for cache management
  - Modify HDFS client



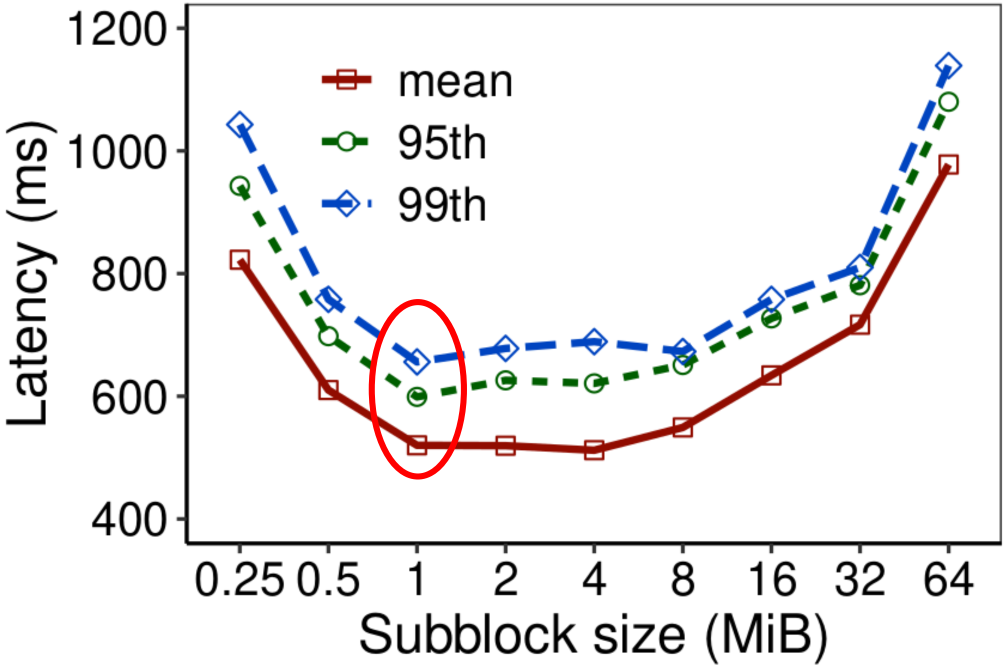
Architecture of POCache

Evaluation Setup

- Local cluster
  - 15 commodity machines
    - Intel core i5, 16 GiB RAM, 1 TiB SATA disk
  - 10 Gbps Ethernet switch
  - Employ benchmark tool *DFS-Perf*
  - Inject stragglers by running Linux *stress*
  - 64-MiB block, 256-MiB file (4 blocks) by default
- Amazon EC2
  - 30 m5.large instances, 2 m5.2xlarge instances
    - Magnetic storage
  - 5 Gbps network bandwidth

Evaluation Results

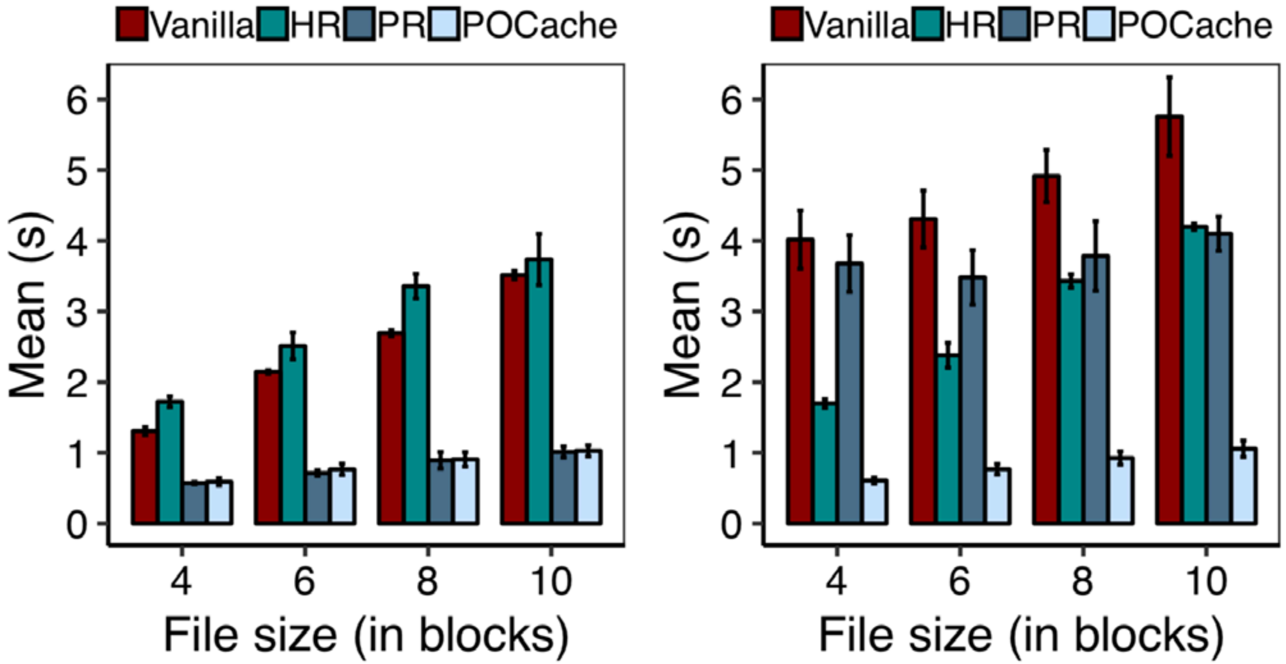
- Effectiveness of block slicing
  - Cache one parity block for a file
    - Lowest latency when subblock is of 1 MiB



Evaluation Results

➢ Single-client reads

- Read mechanisms
  - Vanilla, read sequentially
  - HR, hedged read
  - PR, read blocks in parallel
- Evaluate different file sizes
- POCache reduces the latency with straggler to the latency in normal case

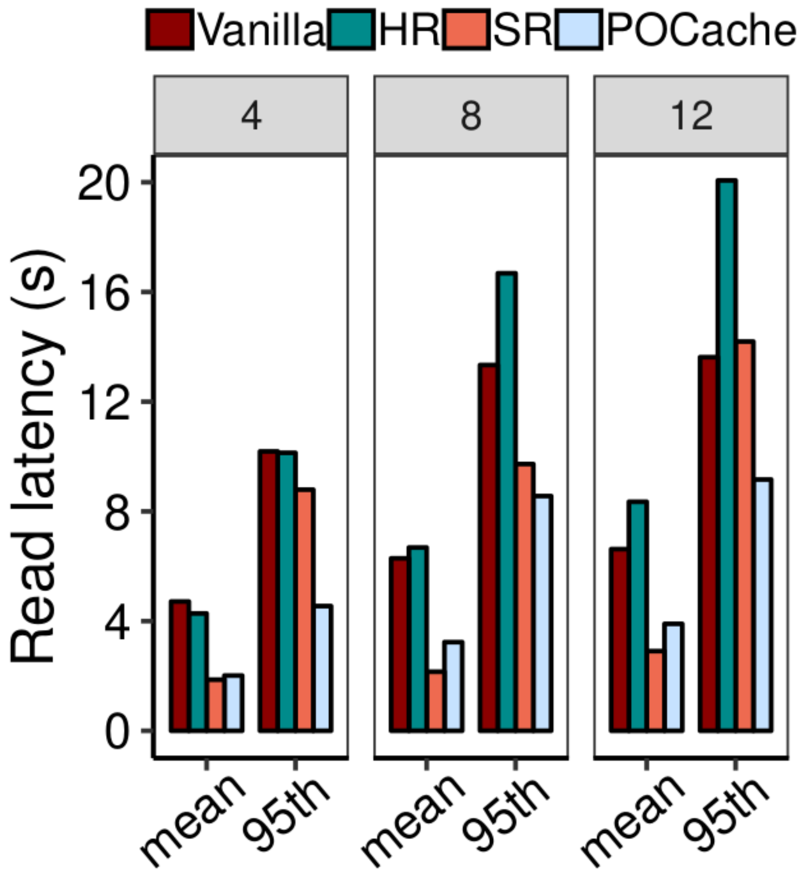


(a) Without any straggler

(b) With one straggler

Evaluation Results

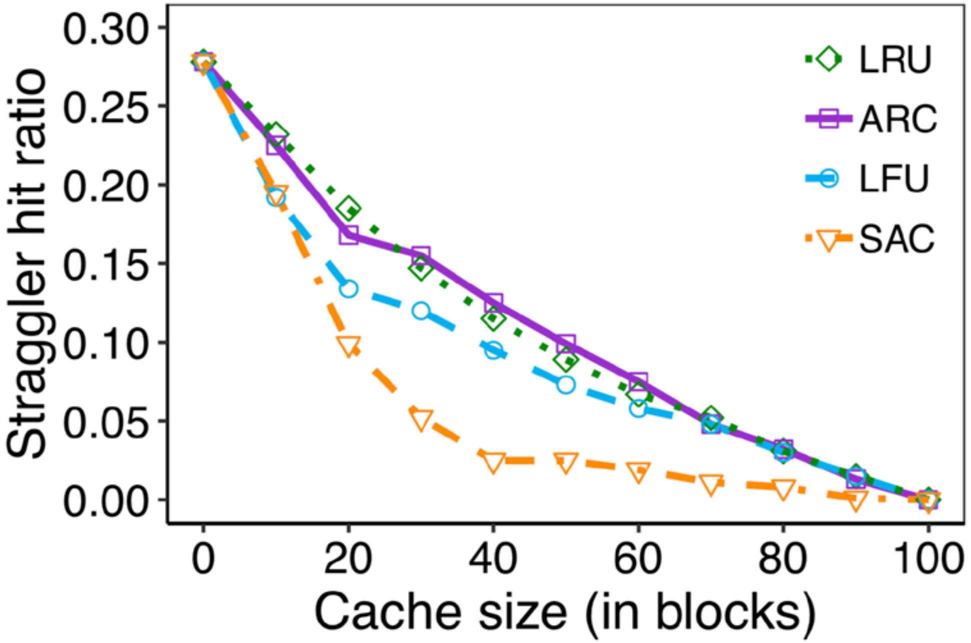
- Multi-client reads
  - Read mechanisms in comparison
    - Vanilla, read blocks sequentially
    - HR, hedged read
    - SR, cache popular data blocks
  - Evaluate with 4, 8, 12 clients
  - POCache achieves low mean and tail latencies



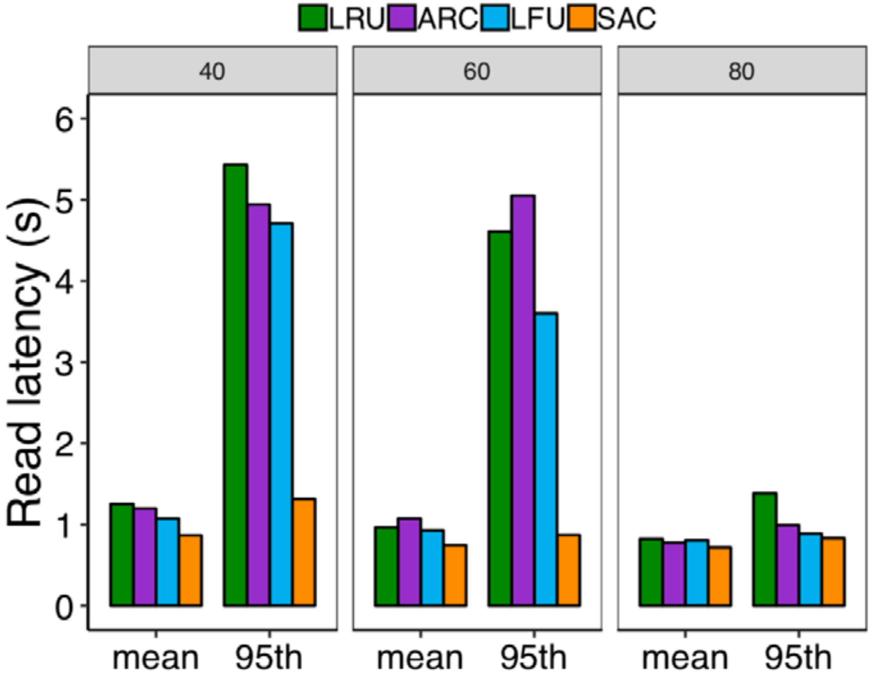
With one straggler

Evaluation Results

- Cache efficiency of Straggler-aware cache algorithm (SAC)



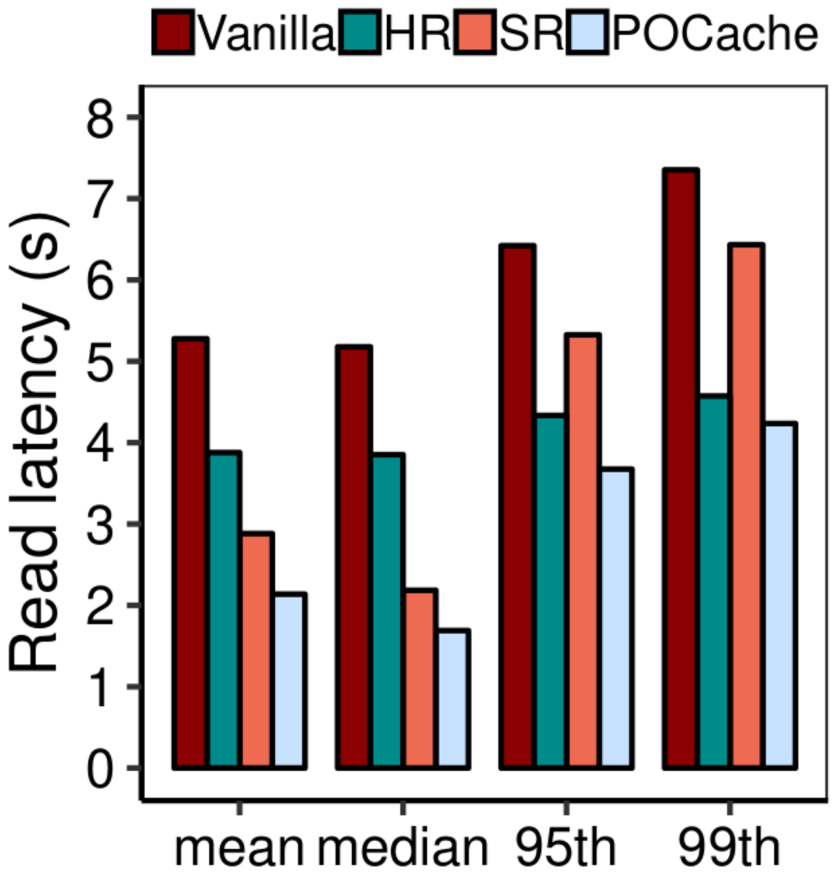
Straggler hit ratio under different cache sizes



Latencies under different cache sizes

Evaluation Results

- Experiments on Amazon EC2
  - Read mechanisms
    - Vanilla, read sequentially
    - HR, hedged read
    - PR, read blocks in parallel
  - Stragglers naturally appear in cloud
  - POCache achieves lowest latency among all four read policies



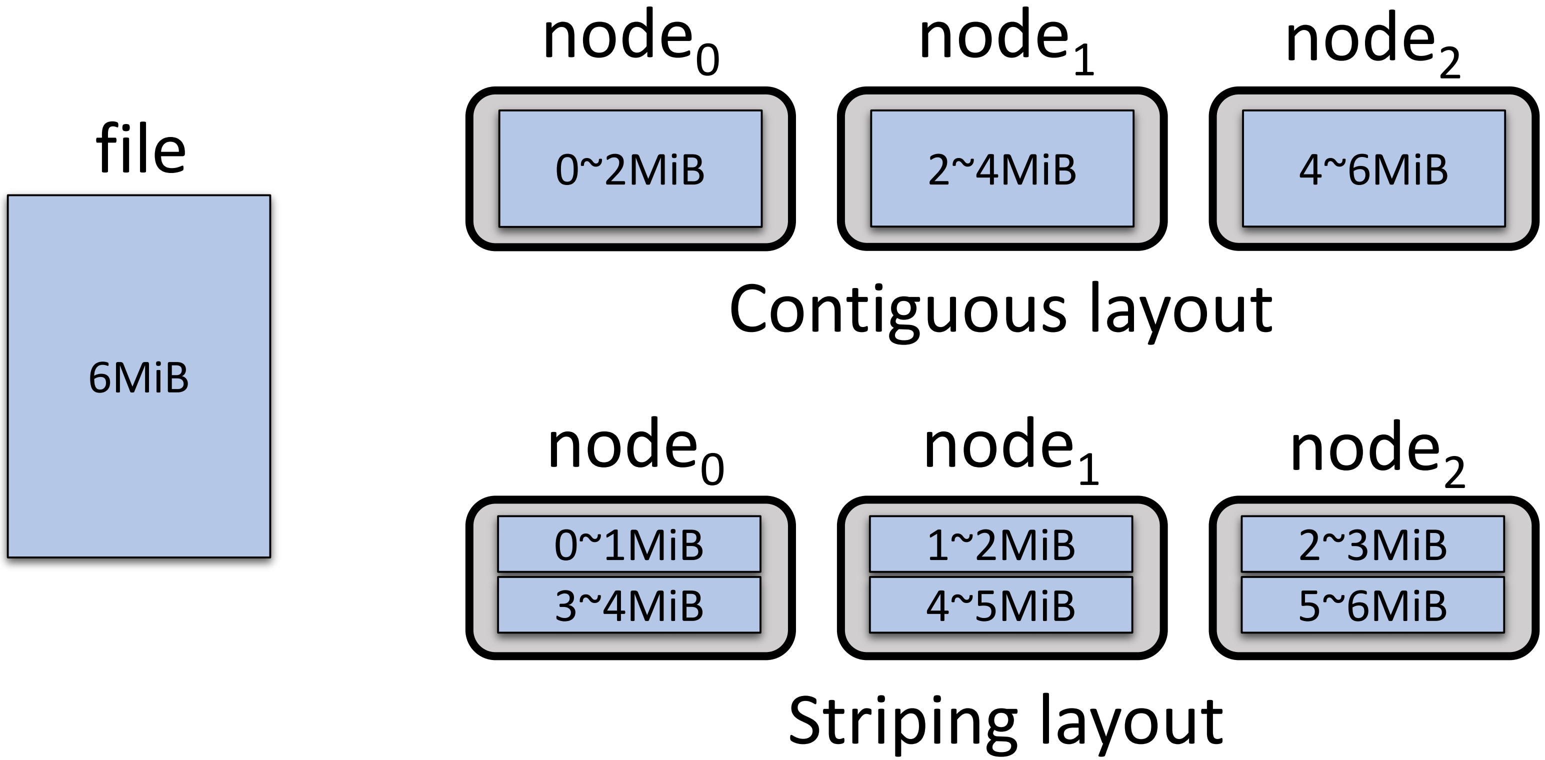
Conclusion

- ➢ Present **POCache**, a parity-only caching design for robust straggler tolerance
  - Minimize coding overhead
  - Straggler-aware cache algorithm
  - Preserve original workflow and performance
- ➢ Implement POCache on Hadoop 3.1 HDFS
- ➢ Conduct experiments on a local cluster and Amazon EC2
- ➢ Source code
  - **http://adslab.cse.cuhk.edu.hk/software/pocache**

Thank you!
Q&A

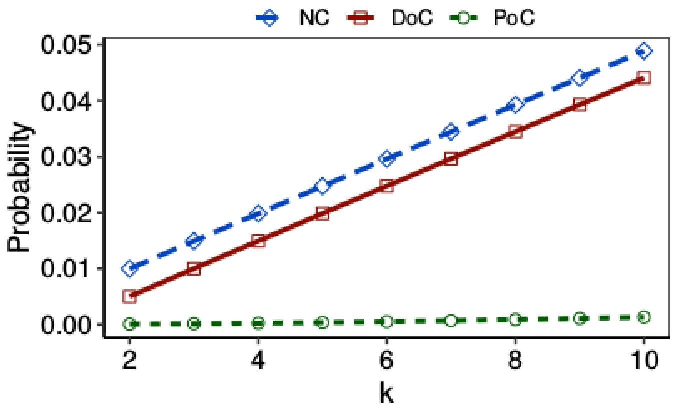
Background and Motivation

- Stragglers affect both data layouts
  - Contiguous layout
  - Striping layout

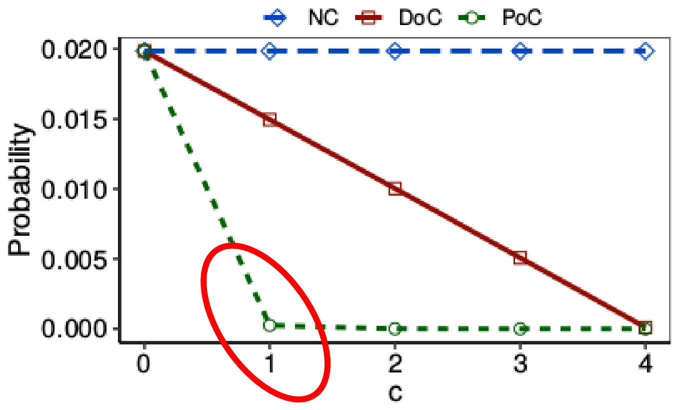


Numerical Analysis

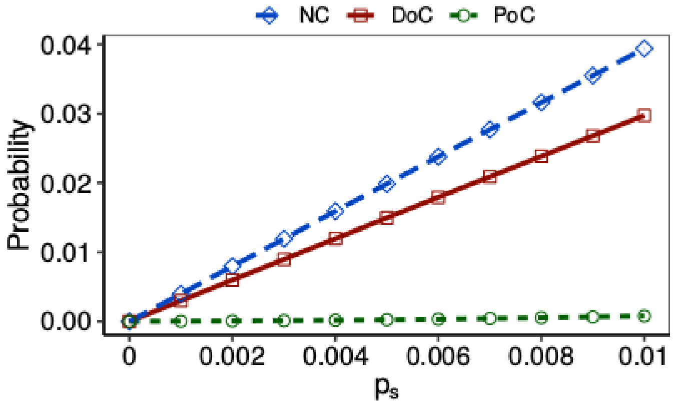
- Probability of hitting straggler



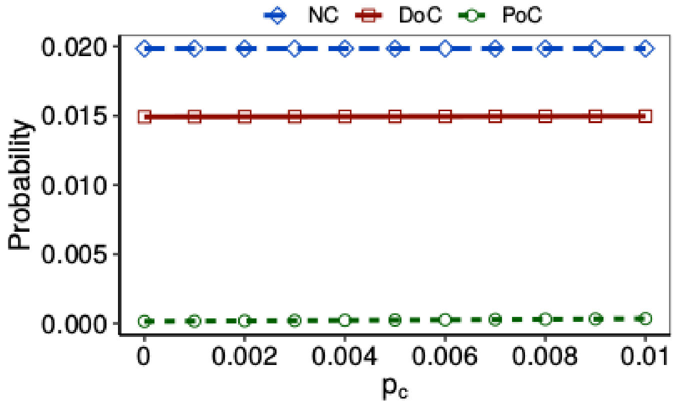
(a) $c = 1,\ p_s = 0.005,\ p_c = 0.005$



(b) $k = 4,\ p_s = 0.005,\ p_c = 0.005$



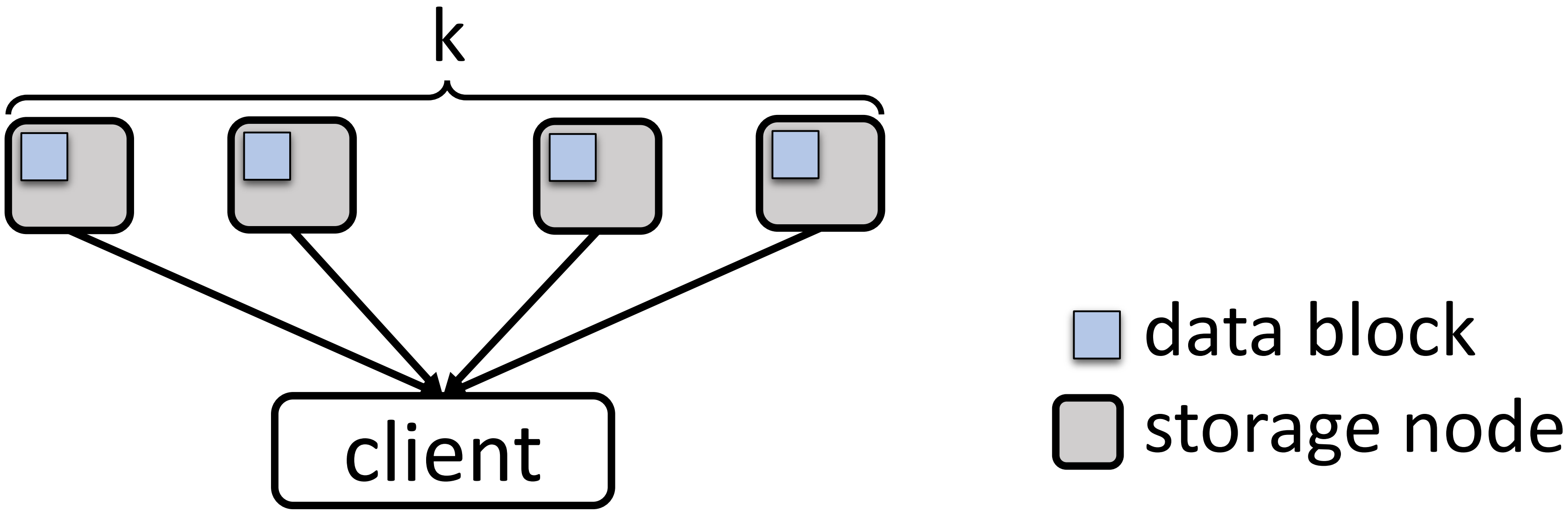
(c) $k = 4,\ c = 1,\ p_c = 0.005$



(d) $k = 4,\ c = 1,\ p_s = 0.005$

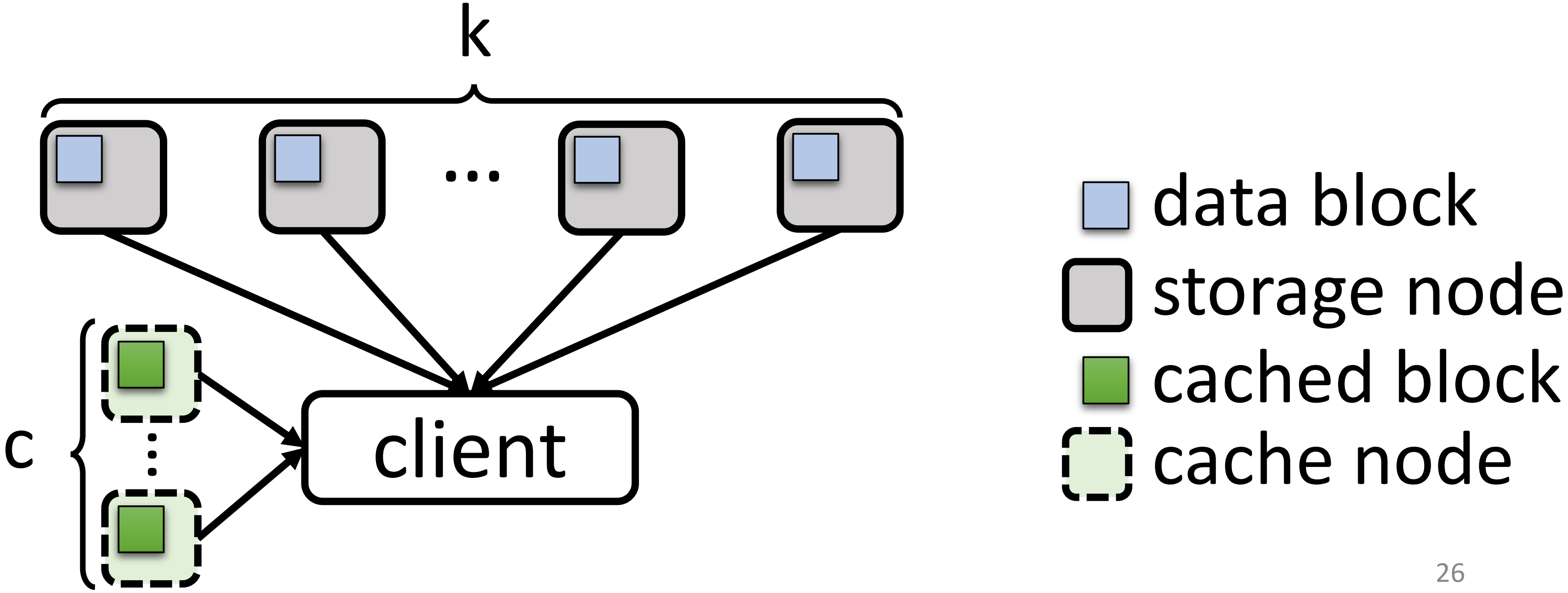
Motivation

- Stragglers slow down read request
  - Read a file ***f***, consisting of ***k*** blocks residing on k nodes
  - Each node behaves abnormally with probability $\boldsymbol{p_s}$



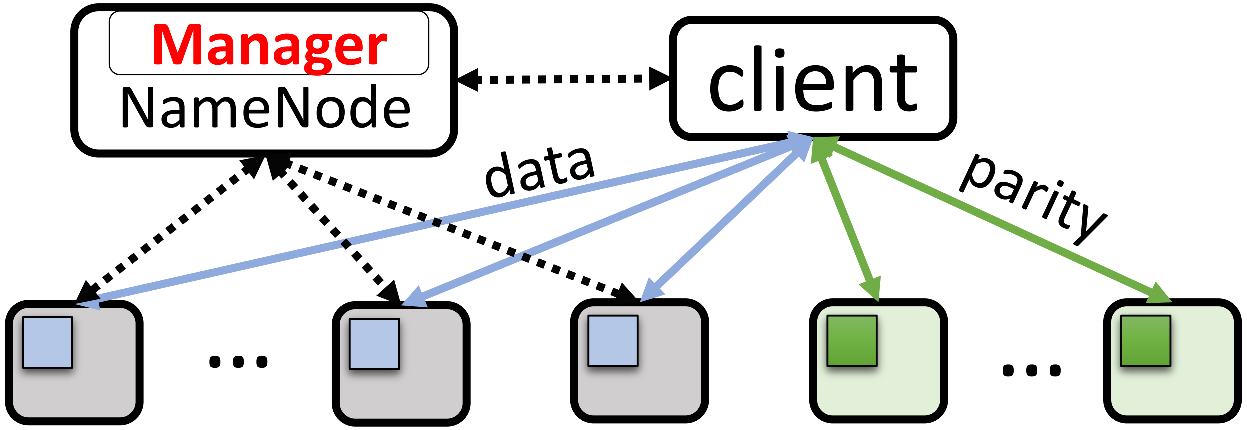
Mathematical Analysis

- Two caching strategies
  - Data-only cache
    - ***c*** data blocks out of *k* data blocks
  - Parity-only cache
    - ***c*** parity blocks generated from *k* data blocks



Implementation

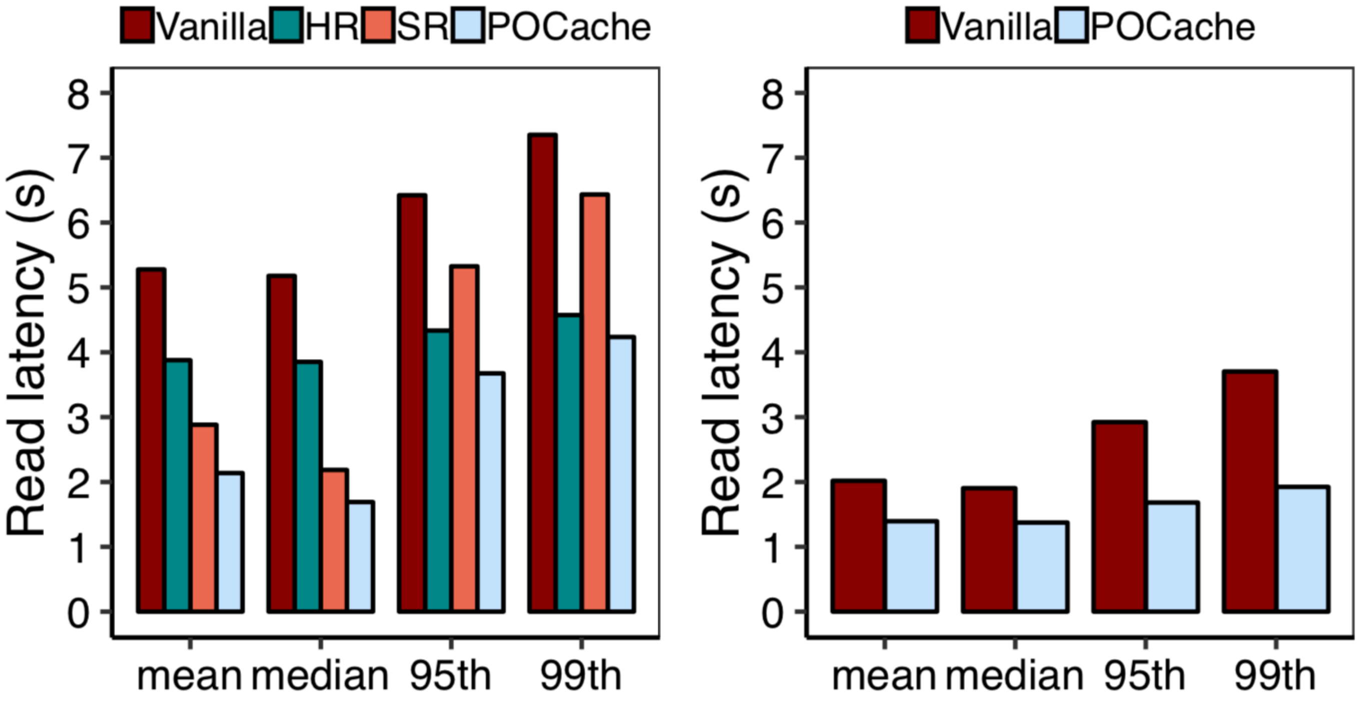
- Add Manager on NameNode
  - Store metadata on cached parities
  - Support different cache algorithms via two primitives:
    - `Query`, return admission decision
    - `Update`, update related information and return eviction decision if needed



Architecture of POCache

Evaluation Results

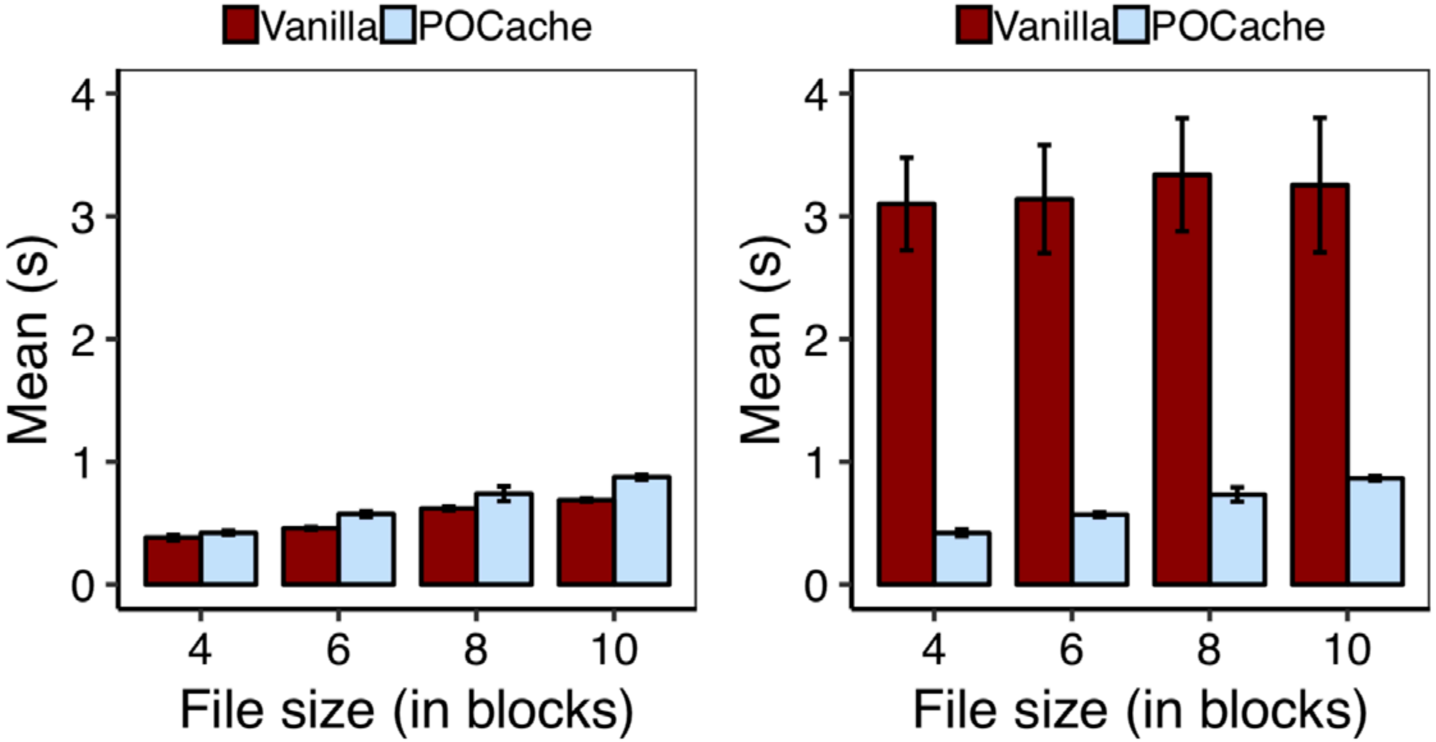
- Experiments on Amazon EC2 cloud



(a) Contiguous layout (b) Striping layout

Evaluation Results

- Single-client reads under striping layout



(a) Without any straggler

(b) With one straggler