CodePlugin: Plugging Deduplication into Erasure Coding for Cloud Storage

Mengbai Xiao, Qi Wei, Songqing Chen
George Mason University

Mohammed A. Hassan
NetApp Inc.

Weijun Xiao
Virginia Commonwealth University

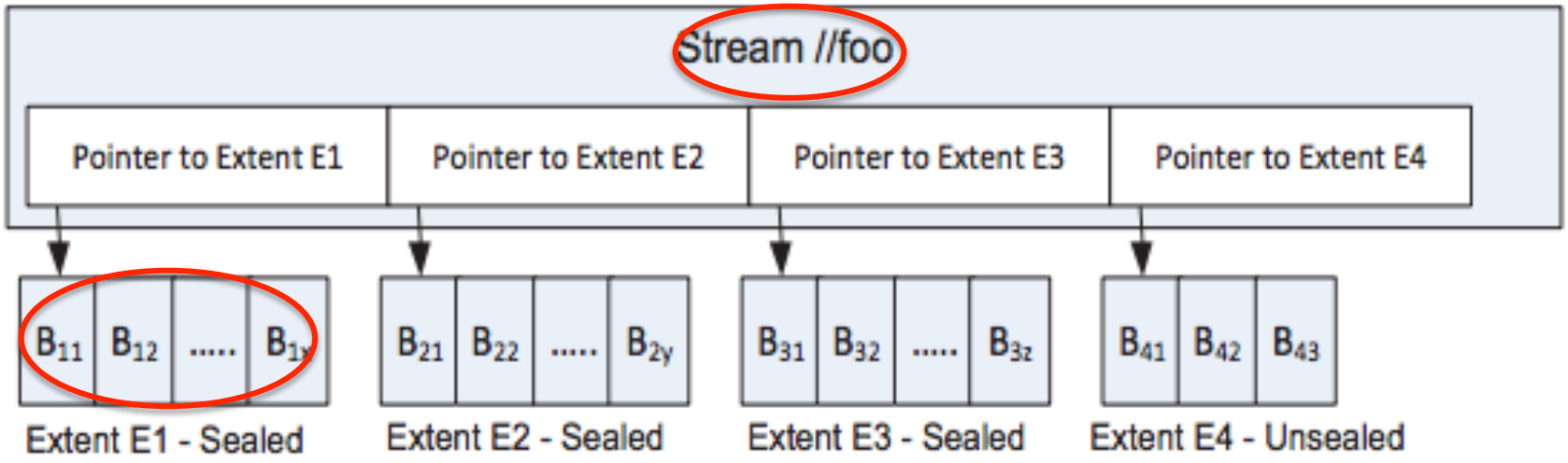
Cloud Storage Systems

- The cloud services mostly rely on the cloud storage systems, e.g., Google File System (GFS) or Windows Azure Storage (WAS)



Cloud Storage System Features

- Fixed size data files
  - WAS [SOSP '11] dumps data as 300 MB files



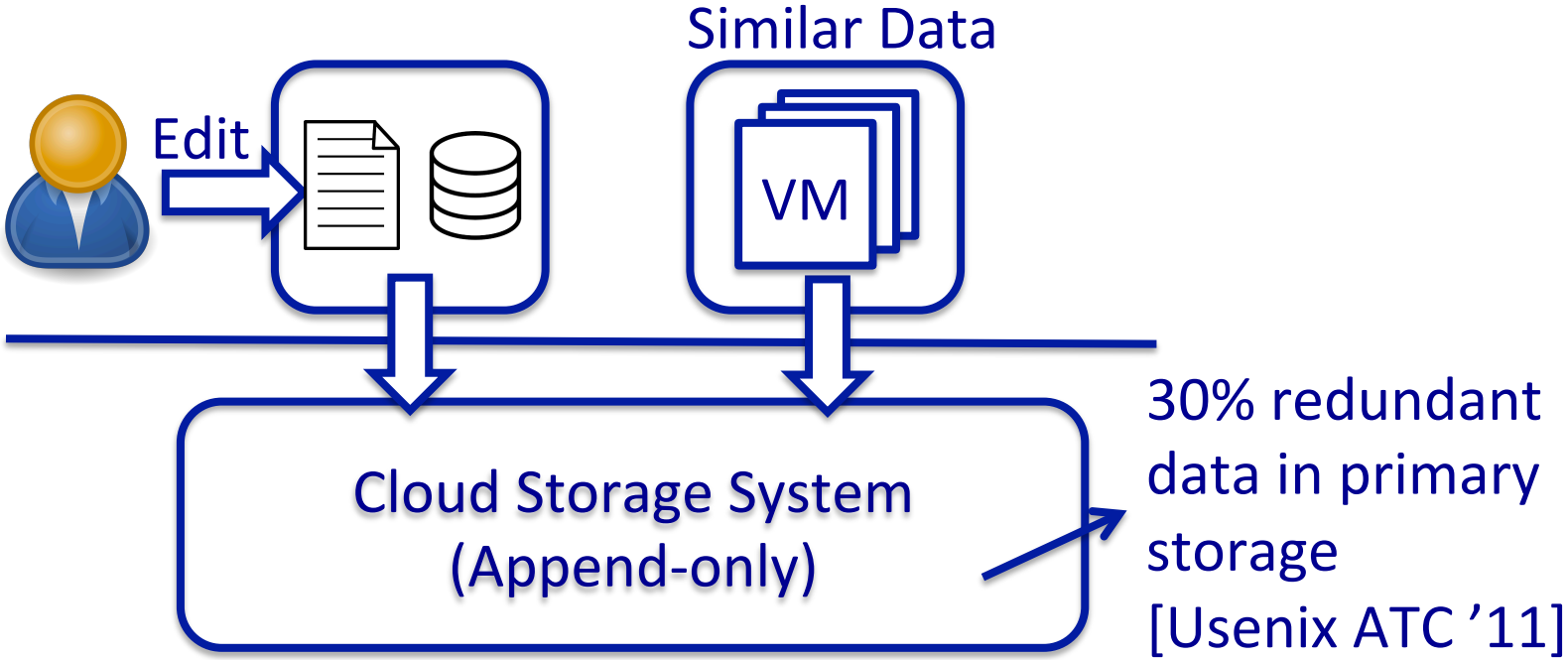
- Availability and reliability
  - Node failures are not uncommon

Replica Protocol

- Triple replica
  - There are 3 copies for each file
- Erasure code
  - Coding parameter: (k, m)
  - Higher availability and storage efficiency
  - Lower throughput

Redundant Workload

- Redundant data in cloud storage



- Replica protocols have to process the redundant data
  - Extra I/O cost
  - Extra storage cost

Deduplication Technique

- Deduplication is widely used in the backup storage systems

- Typical steps
  - Chunk and fingerprint input data
  - Determine redundancy by searching index table
  - Process unique and redundant data, respectively

Challenges

- Naïve method
  - Deduplication and then encoding
- Disadvantages
  - Demand filesystem support
  - Fragmentation: sequential I/Os turn into random I/Os

Outline

- CodePlugin Design and Implementation
- Preliminary Evaluation
- Conclusion and Future Work

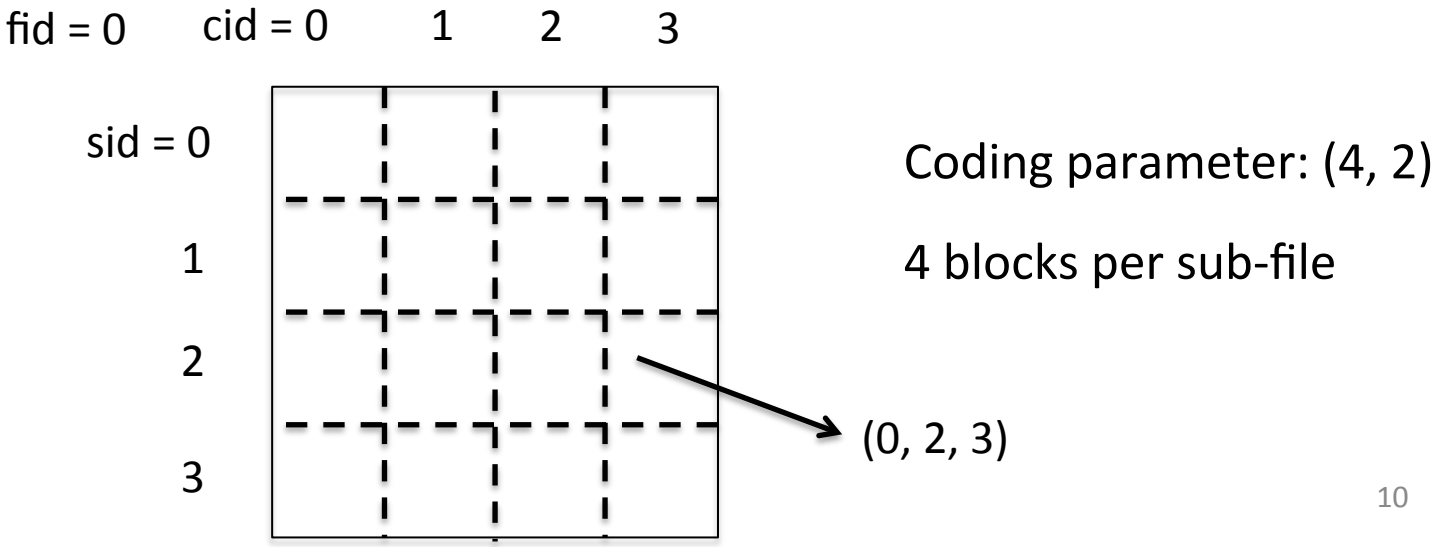
CodePlugin Design and Implementation

- Features
  - CodePlugin is designed as an independent coding scheme
  - CodePlugin keeps the original files untouched
  - CodePlugin can be seamlessly integrated with any erasure code

- Major steps
  - Preprocess (deduplication)
  - Encode and decode

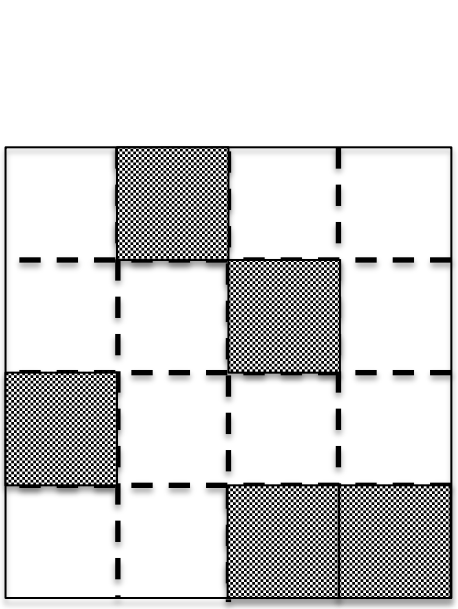
Preprocess – Addressing Blocks

- Data file rasterization
  - Each row is the unit of erasure coding, named sub-file
  - Each block is the unit of deduplication
  - Each block is addressed as *(fid, sid, cid)*



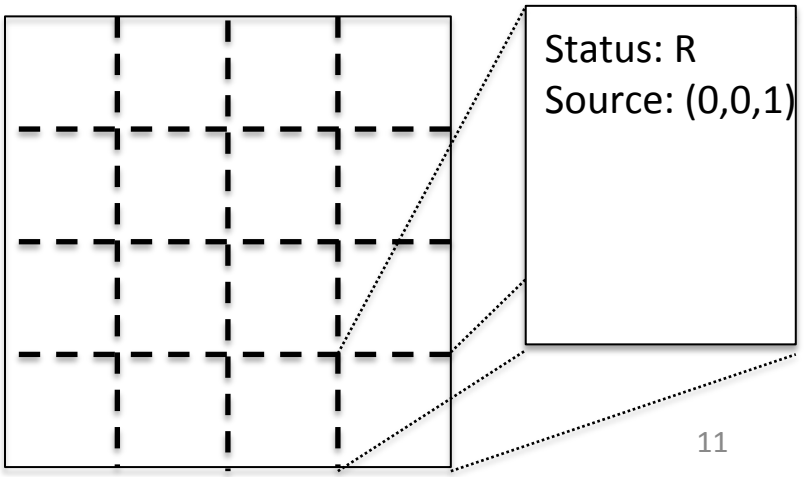
Preprocess – Deuplication

- Identify the redundant and unique blocks
  - Fingerprint table in main memory
  - Eviction algorithm: LRU
- Generate one map file for one data file



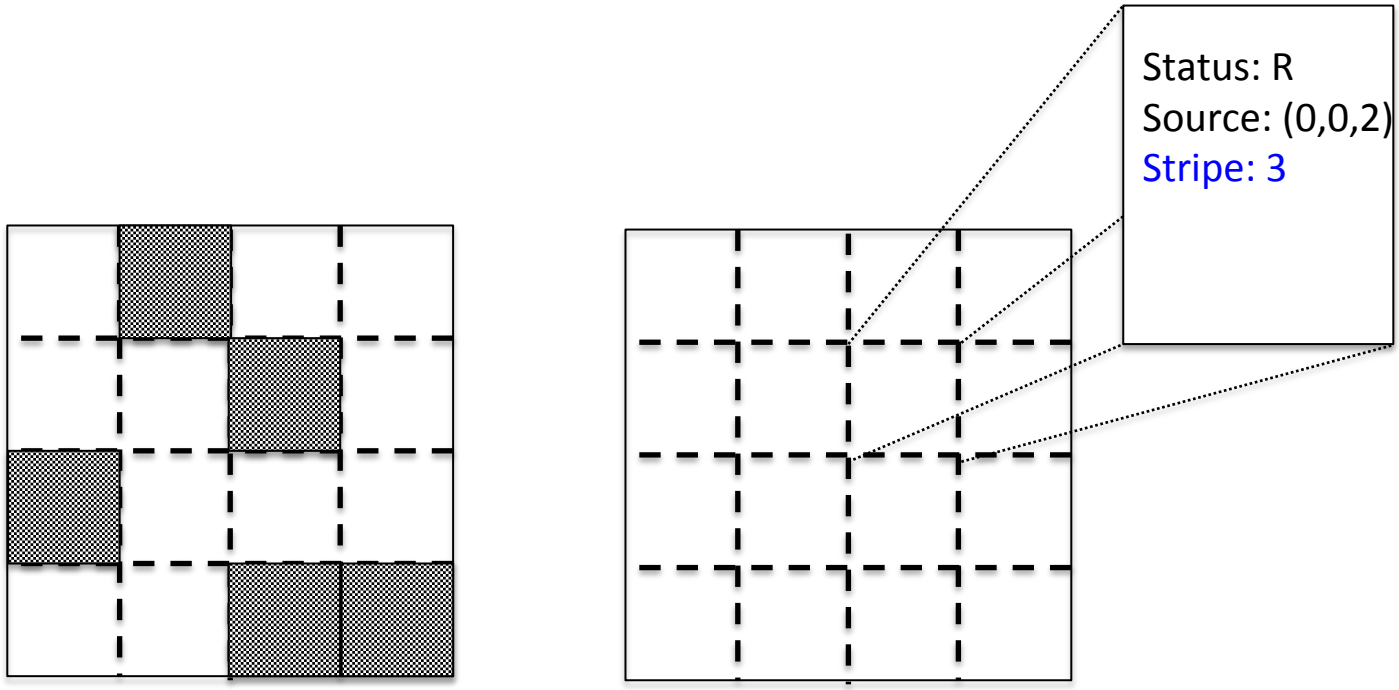
Fingerprint table

FP#1	(0,0,0)
FP#2	(0,0,1)
	……



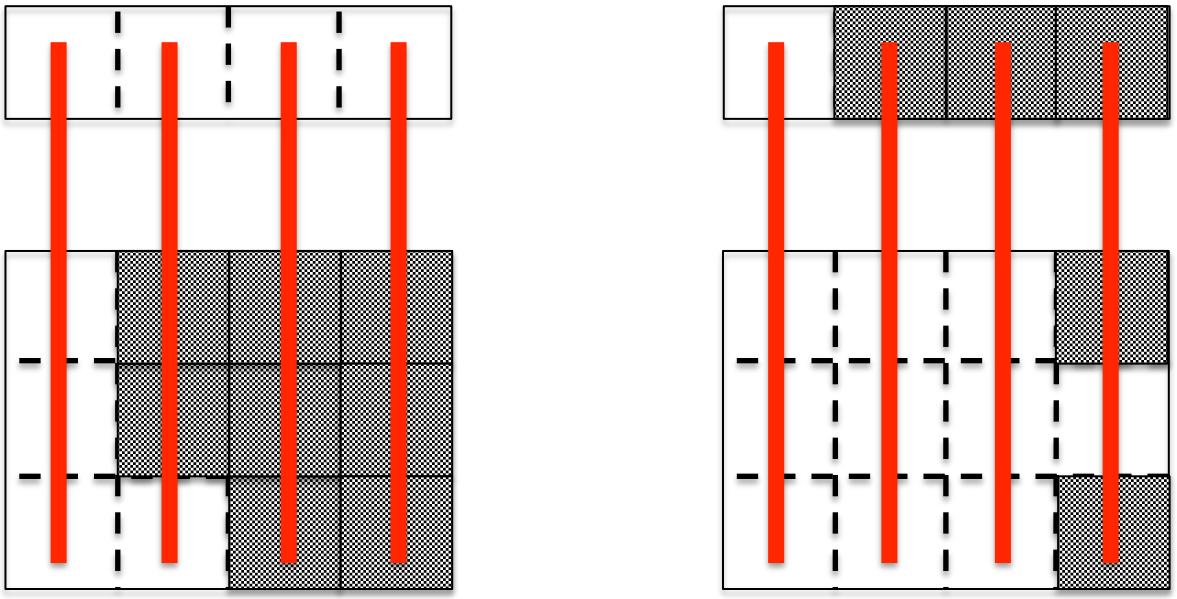
Preprocess – Pseudo-shuffling

- Shuffle blocks for encoding unique blocks only
- Migrations are recorded in the map file



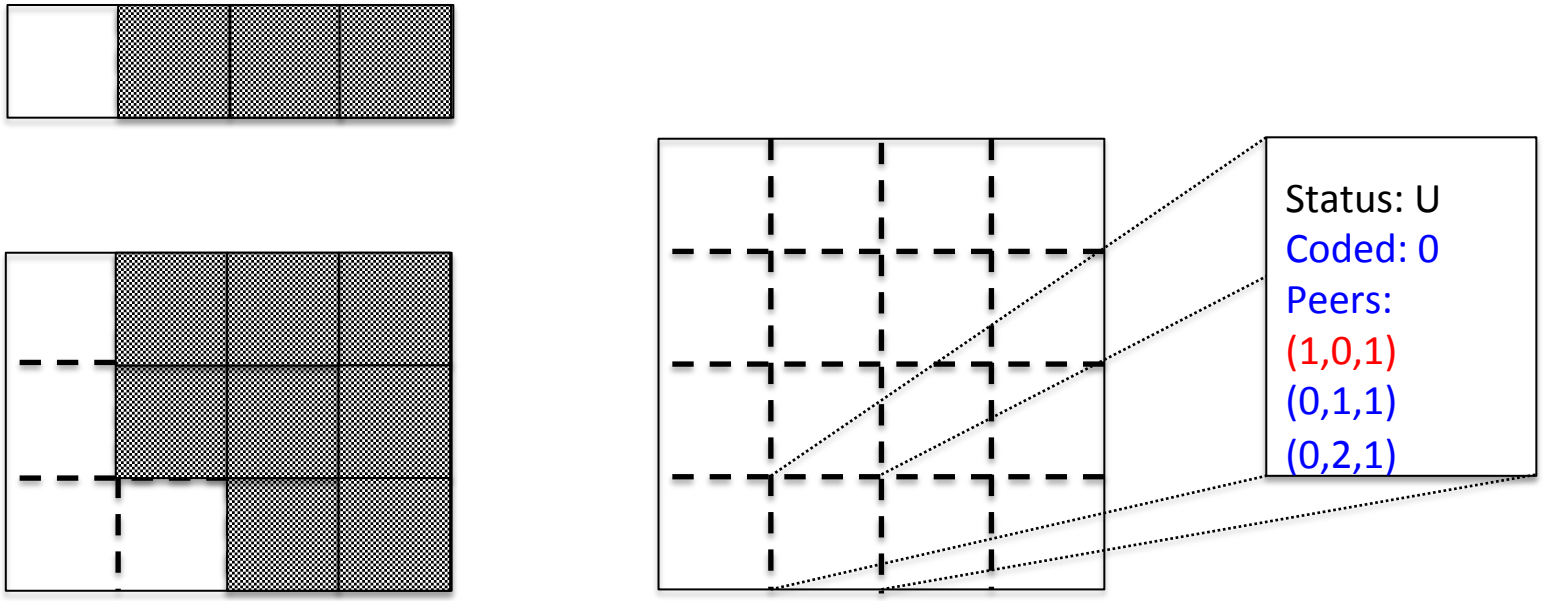
Preprocess – Sub-file Exchange (1)

- Optional step
- Exchange sub-files for balancing unique block number



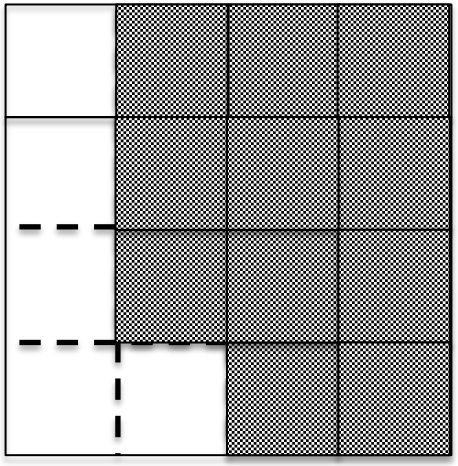
Preprocess – Sub-file Exchange (2)

- Extra information for decoding
  - Encoding peers and coded file location (file id)

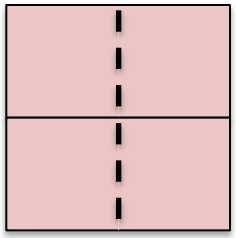


Encoding

- Encode the data file stripe by stripe
  - Skip columns full of redundant block

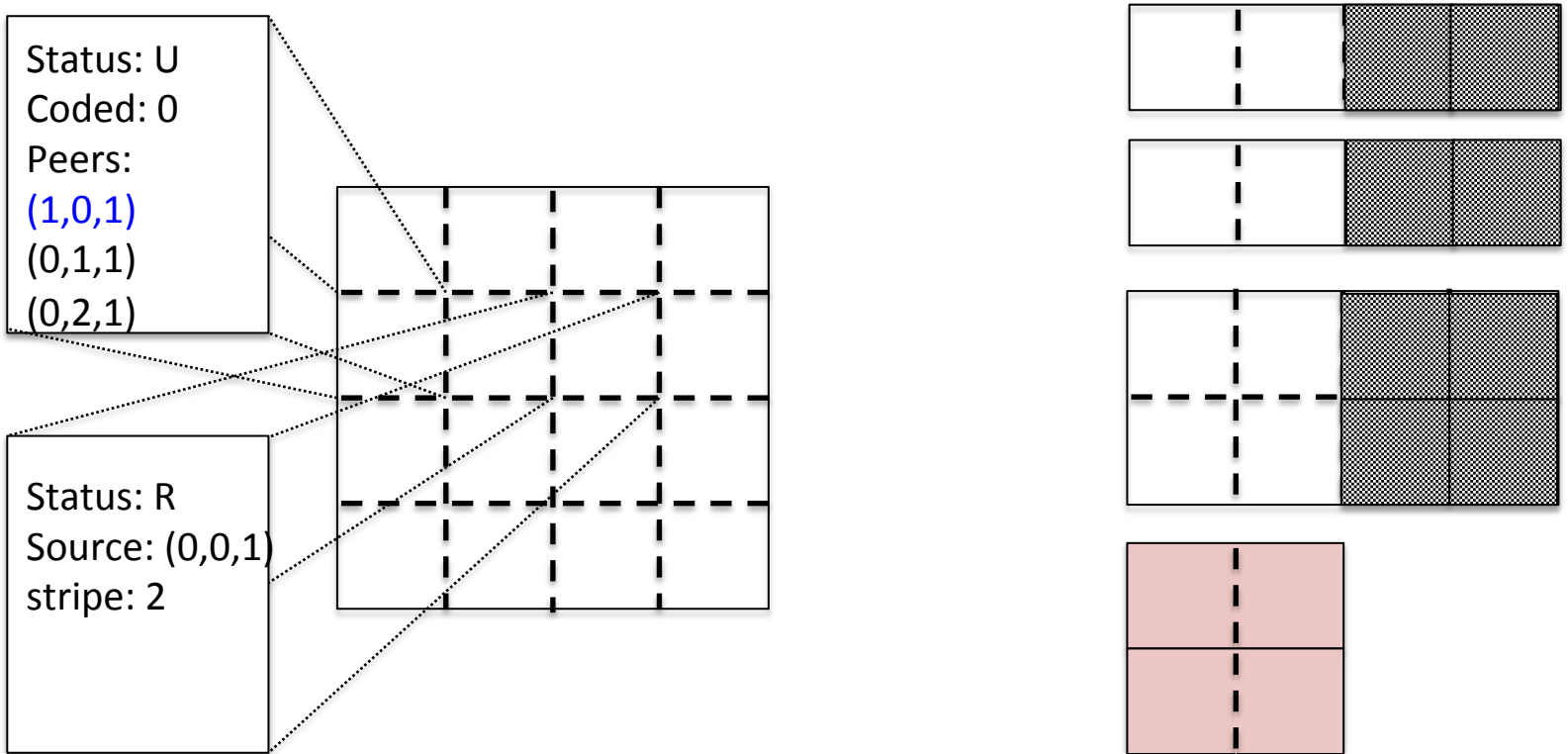


Coded file



Decoding

- Unique blocks: read corresponding blocks and decode
- Redundant blocks: read from the source



Outline

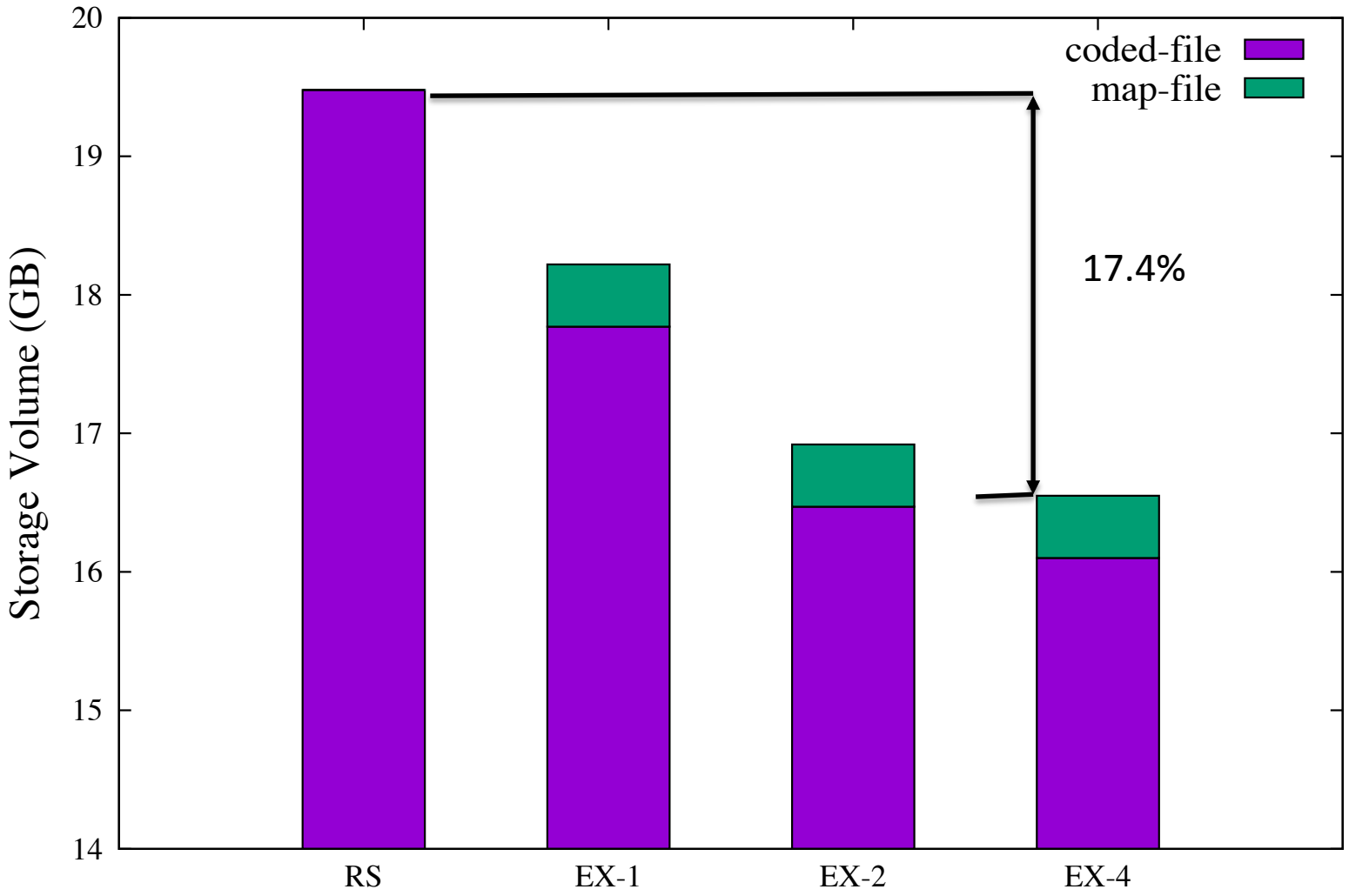
- CodePlugin Design and Implementation
- Preliminary Evaluation
- Conclusion and Future Work

Experiment Setup

- Dataset
  - 17 VM images of different OS, including Ubuntu, Fedora, Debian and etc.
  - Dump these images into 300 MB files, totally ~40 GB
  - 23% redundant data in the workload

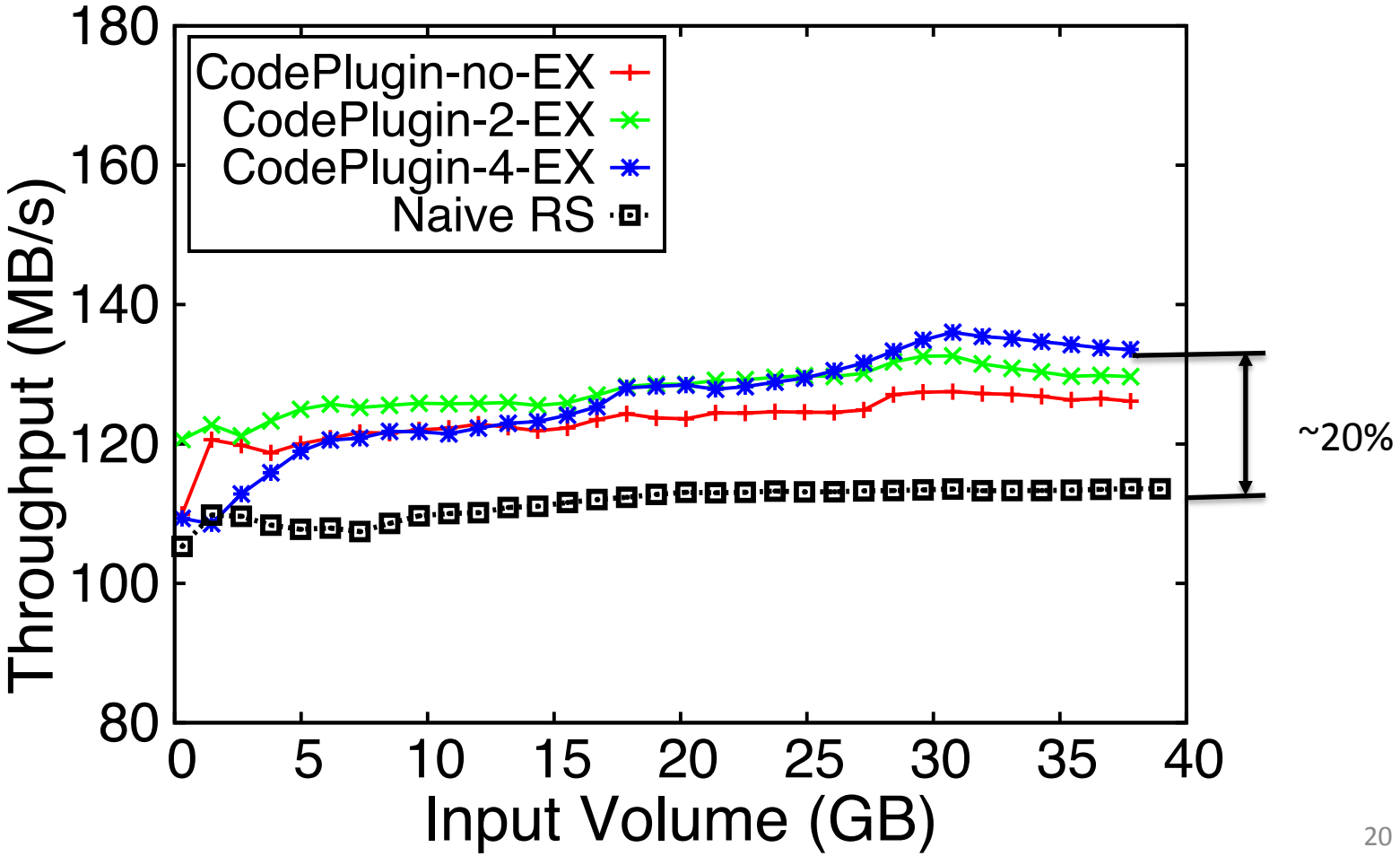
- Experiment factors
  - Coding parameters: (4, 2)/(6, 3)/(10, 5)
  - Sub-file exchange number: 1(No)/2/4
  - Cache size: 128/256/512 MB

Storage Efficiency



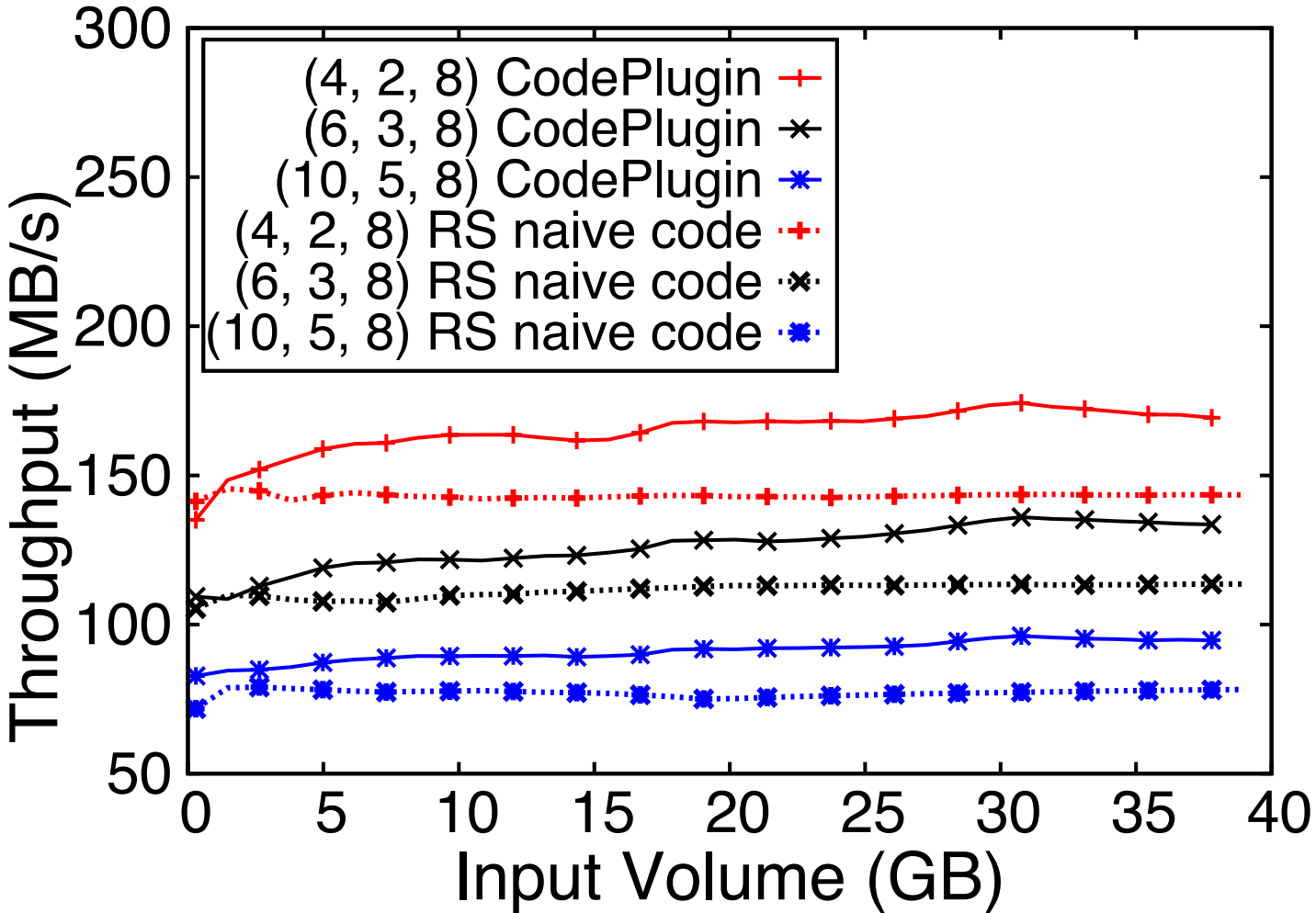
Encoding Throughput (1)

- Varying sub-file exchange number



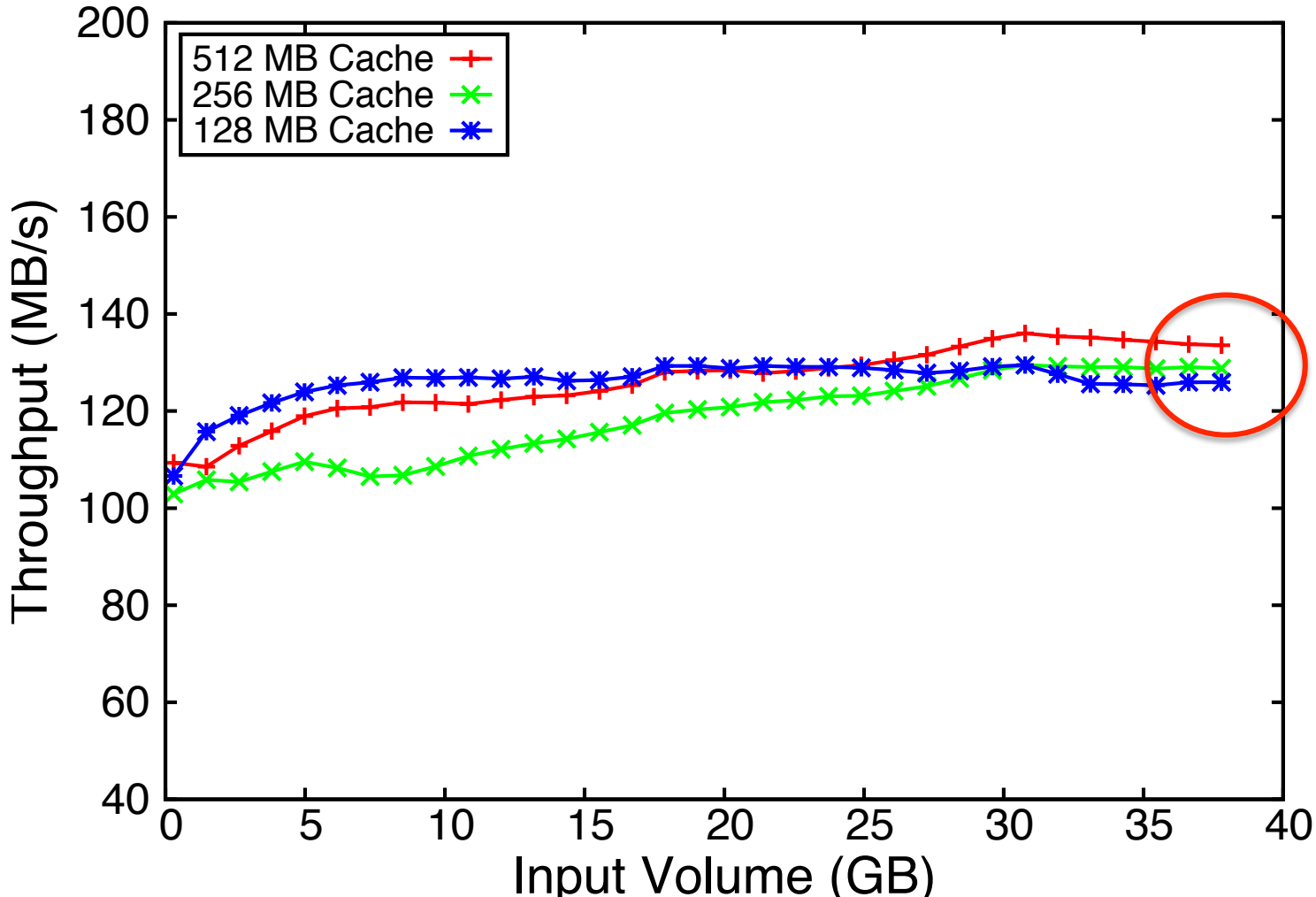
Encoding Throughput (2)

- Varying coding parameters



Encoding Throughput (3)

- Varying cache size



Conclusion and Future Work

- We design and implement the CodePlugin, which integrates the deduplication into the erasure coding
- CodePlugin is designed as a system-independent coding scheme with the input files untouched
- According to preliminary evaluation, upto ~17% storage space can be saved and upto ~20% encoding throughput can be improved on the fresh VM images workload.

Thanks