OpenEC: Toward Unified and Configurable Erasure Coding Management in Distributed Storage Systems

Xiaolu Li[1], Runhui Li[1], Patrick P. C. Lee[1], Yuchong Hu[2]

The Chinese University of Hong Kong[1]
Huazhong University of Science and Technology[2]

USENIX FAST 2019

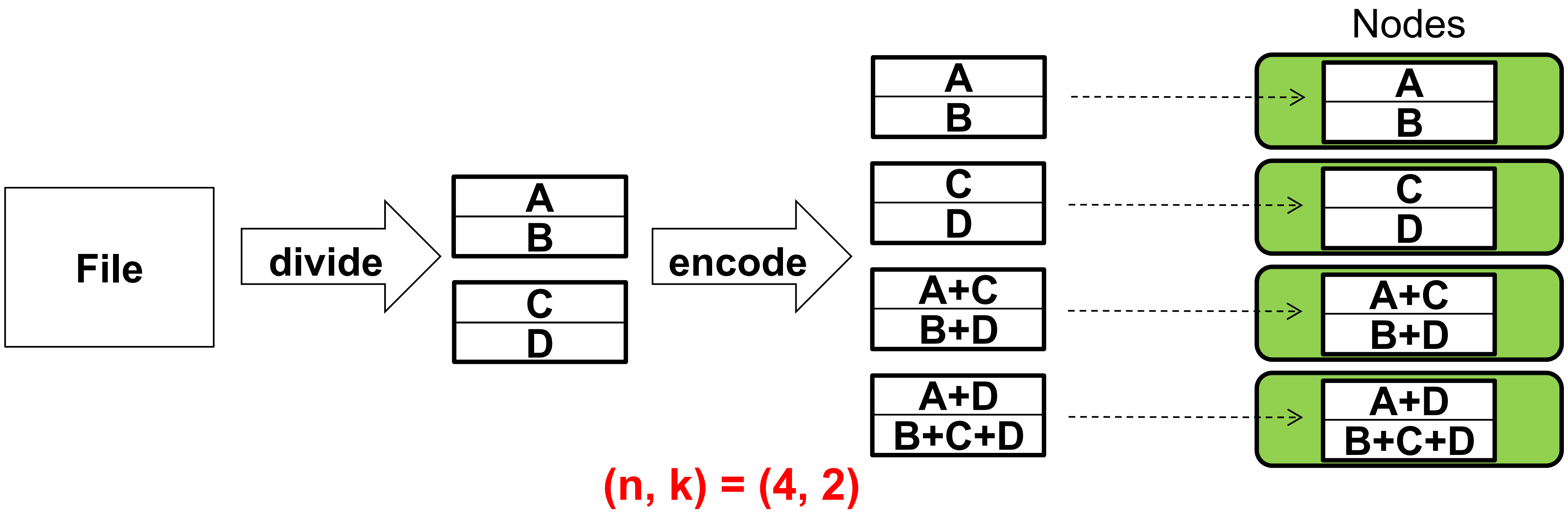
Introduction

- Fault tolerance for distributed storage is critical
  - **Availability**: data remains accessible under failures
  - **Durability**: no data loss even under failures

- **Erasure coding** is a promising redundancy technique
  - Minimum data redundancy via “data encoding”
  - Higher reliability with same storage redundancy than replication
  - Reportedly deployed in Google, Azure, Facebook
    - e.g., Azure reduces redundancy from 3x (replication) to 1.33x (erasure coding) → PBs saving

Erasure Coding

- Divide file data to **k** data blocks
- Encode k data blocks to **n-k** parity blocks
- Distribute the n erasure-coded blocks (coding group) to n nodes
- **Fault-tolerance**: any k out of n blocks can recover file data



Erasure Coding

- Reed-Solomon (RS) codes are widely deployed
  - Storage-optimal
  - Generality for n and k
  - *Drawback: high repair penalty*

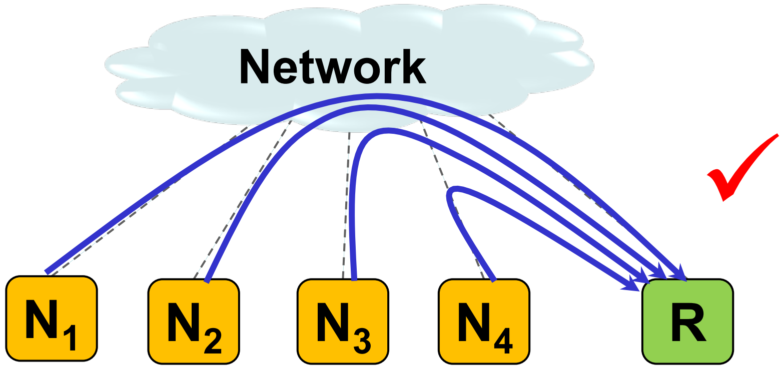
- New erasure coding solutions
  - Repair-optimal erasure codes
    - e.g., regenerating codes [TIT'10]; locally repairable codes (LRCs) [ATC'12, PVLDB'13]; double regenerating codes (DRC) [TOS'17]
  - Repair-efficient algorithms
    - e.g., Partial-parallel-repair (PPR) [Eurosys'16]; Repair pipelining [ATC'17]

Challenge

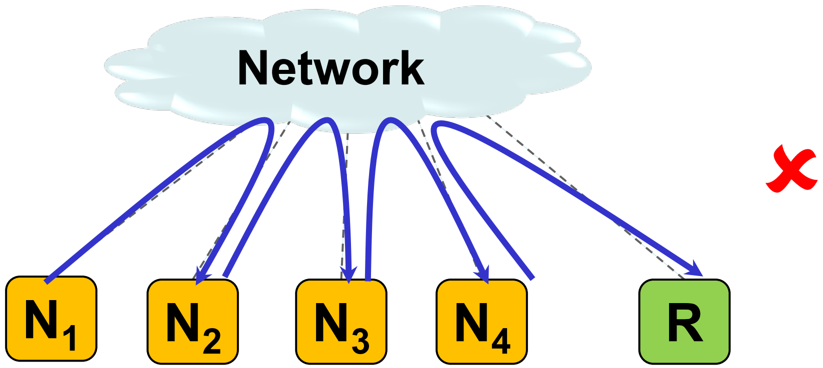
- Deploying new erasure coding solutions in distributed storage systems (DSSs) is a daunting task
  - Re-engineering of DSS workflows (e.g., read/write paths)
  - Hard to generalize for different DSSs
- Our past experience:
  - Over 4K lines-of-code change to HDFS-RAID for adding DRC [TOS'17]
- Review of six DSSs with erasure coding support
  - HDFS-RAID, Hadoop 3.0 HDFS, QFS, Tahoe-LAFS, Ceph and Swift

Limitations of Current DSSs

- Hard to add advanced erasure codes
  - Existing DSSs only provide interfaces for basic encoding/decoding operations
  - Most DSSs do not support sub-packetization (e.g., regenerating codes)
- Hard to configure the workflows and placement of coding operations



Repair in fetch-and-compute manner



Repair pipelining [ATC'17] cannot be readily realized

Our Contributions

OpenEC: a unified and configurable framework for erasure coding management

- Propose **ECDAG**, a directed-acyclic-graph abstraction for realizing general erasure coding solutions
  - Decoupling erasure coding management from DSS workflows
- Prototype OpenEC on HDFS-RAID, Hadoop 3 HDFS, and QFS
  - Minimal code changes
- Extensive experiments on local and Amazon EC2 clusters

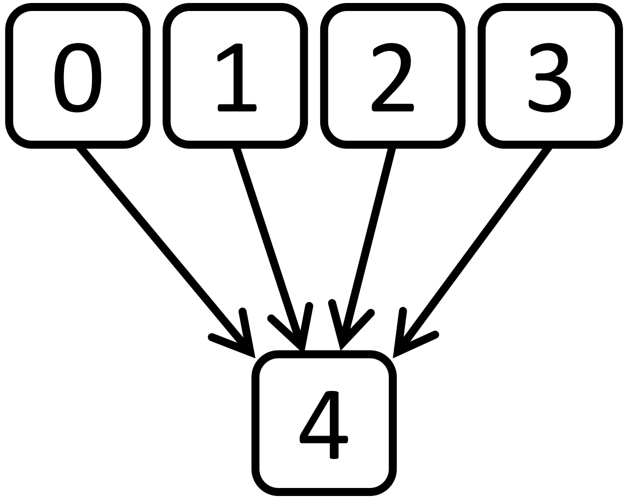
ECDAG

- (n, k) code
  - Data blocks: b_0, …, b_{k-1}
  - Parity blocks: b_k, …, b_{n-1}
  - Virtual blocks: b_i for $i \geq n$

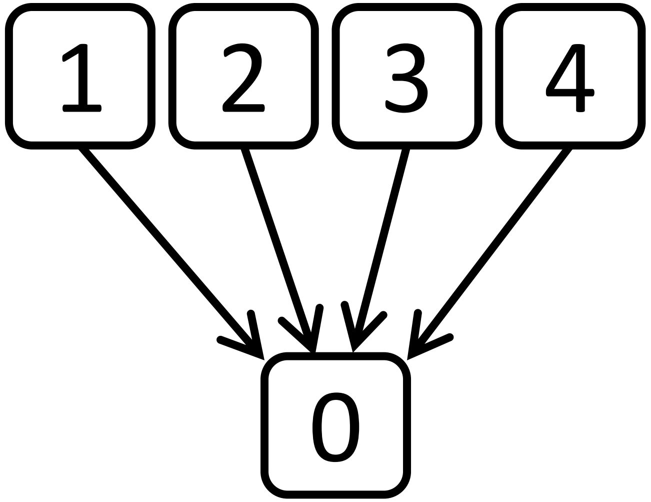
- An ECDAG is a directed acyclic graph that defines either an encoding or a decoding operation
  - Vertex v_i: block b_i in a coding group
  - Edge $e_{i,j}$: block b_i is an input to the linear combination of b_j
    - Each edge is associated with a coding coefficient

ECDAG

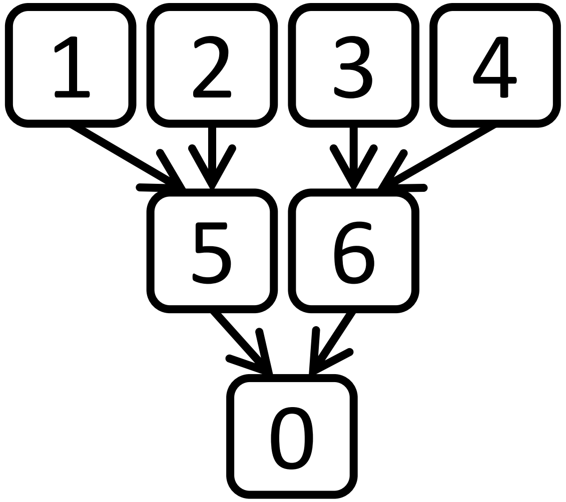
- ECDAGs for a (5,4) code:



Encoding



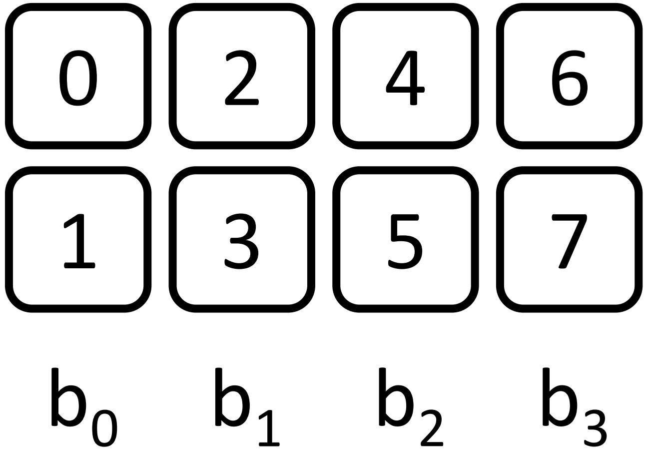
Decoding



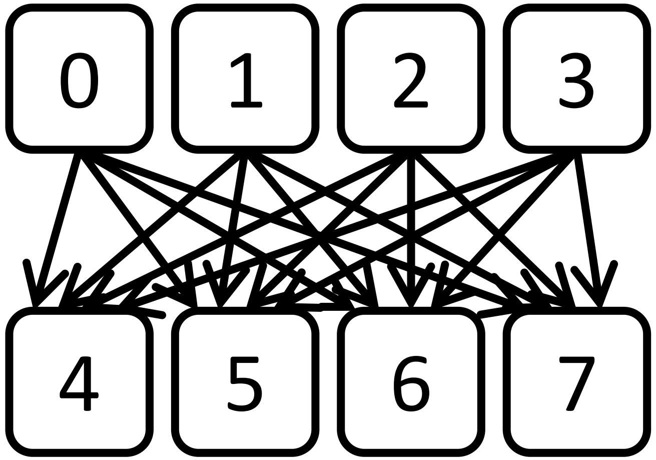
Partial-parallel repair (PPR) [Mitra, EuroSys'16]

ECDAG

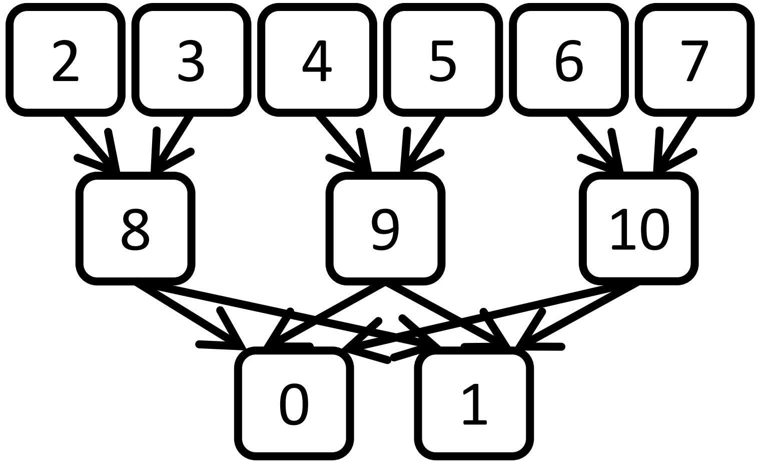
- ECDAGs for regenerating codes [Dimakis, TIT'10] with sub-packetization
  - w: sub-packetization level (number of sub-blocks per block)
  - e.g., n=4, k=2, w=2



Layout



Encoding



Decoding

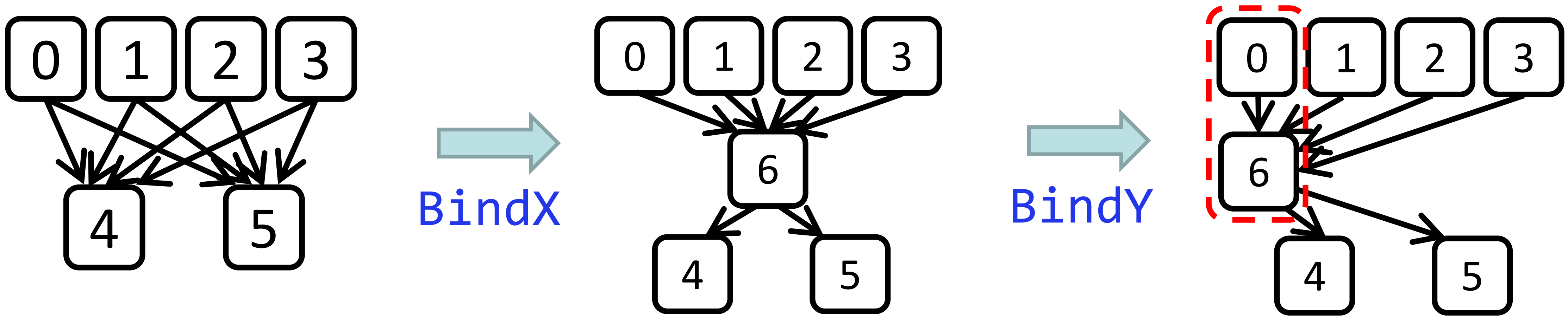
ECDAG Primitives

Construction of an ECDAG:

- **Join**: describes linear combination
- **BindX**: co-locates coding operations at same level (i.e., x-direction)
- **BindY**: co-locates coding operations across levels (i.e., y-direction)

ECDAG Primitives

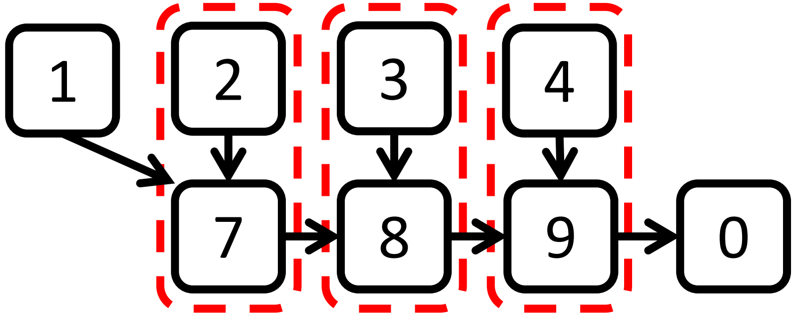
- Encoding of (6,4) RS code




```
ECDAG* ecdag = new ECDAG();
ecdag->Join(4, {0,1,2,3}, {1,1,1,1});
ecdag->Join(5, {0,1,2,3}, {1,2,4,8});
int vidx = ecdag->BindX({4,5});
ecdag->BindY(vidx, 0);
```

ECDAG Primitives

- Decoding via repair pipelining [Li, ATC'17]:
  - e.g., recovering the missing block 0 for (6, 4) RS code




```
ECDAG* ecdag = new ECDAG();
ecdag->Join(7, {1,2}, {1,1});
ecdag->BindY(7, 2);
ecdag->Join(8, {7,3}, {1,1});
ecdag->BindY(8, 3);
ecdag->Join(9, {8,4}, {1,1});
ecdag->BindY(9, 4);
ecdag->Join(0, {9}, {1});
```

Erasure Coding Interfaces

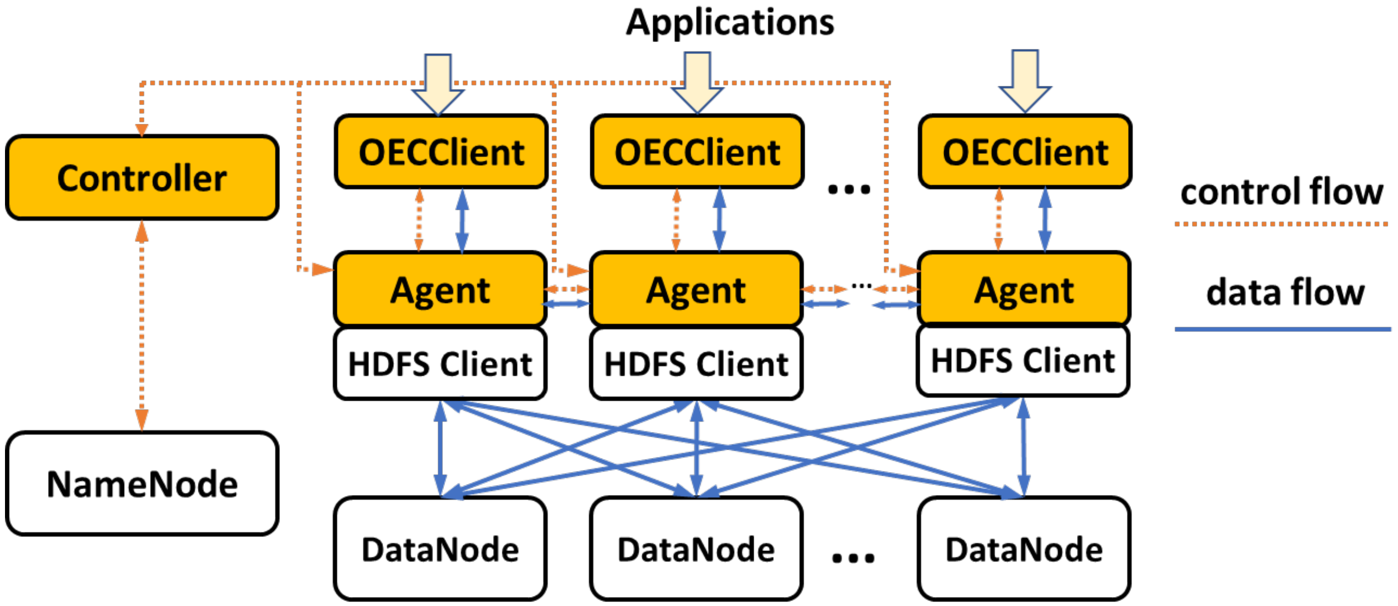
```
class ECBase {
   int n, k, w;
   vector<int> ecoefs;
 public:
   // constructing encoding ECDAGs
   ECDAG* Encode();

   // constructing decoding ECDAGs
   ECDAG* Decode(vector<int> from, vector<int> to);

   // organizing blocks in groups (e.g., racks)
   vector<vector<int>> Place();
}
```

OpenEC Design

- **Controller:**
  - Manages EC metadata
  - Parses ECDAGs and assigns tasks to agents
  - Controls block placement
  - Coordinates repair
- **Agent:**
  - Performs coding operations
- **OECClient:**
  - Interfaces between applications and storage



OpenEC deployment on HDFS

OpenEC Design

Basic operations:

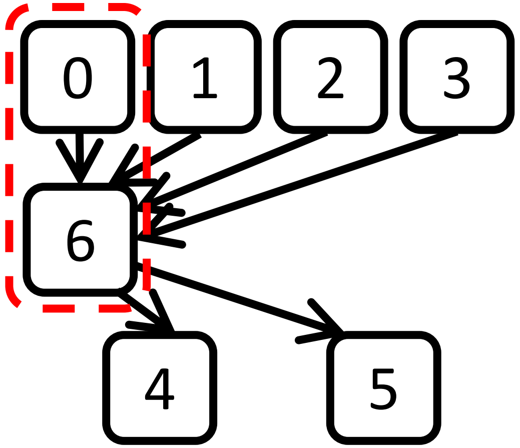
- ➢ Writes
  - Online encoding
  - Offline encoding
- ➢ Normal reads
- ➢ Degraded reads
- ➢ Full-node recovery

Tasks:

- ➢ Load
  - Loads an input block
- ➢ Fetch
  - Retrieves blocks from other agents
- ➢ Compute
  - Computes a new block
- ➢ Persist
  - Returns a block

Parsing an ECDAG

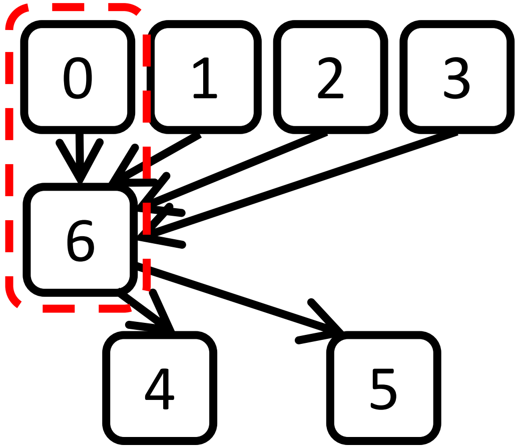
- **Online encoding** for (6,4) RS code
  - On the write path
  - Performed by client C



Vertices	Nodes	Tasks
v_0	C	**Load** b_0
v_1	C	**Load** b_1
v_2	C	**Load** b_2
v_3	C	**Load** b_3
v_6	C	**Compute** b_4 from {b_0, b_1, b_2, b_3} with coding coefficients {1,1,1,1}; **Compute** b_5 from {b_0, b_1, b_2, b_3} with coding coefficients {1,2,4,8};
v_4	C	-
v_5	C	-
-	C	**Persist** b_0; **Persist** b_1; **Persist** b_2; **Persist** b_3; **Persist** b_4; **Persist** b_5;

Parsing an ECDAG

- **Offline encoding** for (6,4) RS code
  - Blocks 0-3 are in nodes 0-3
  - Performed by different nodes

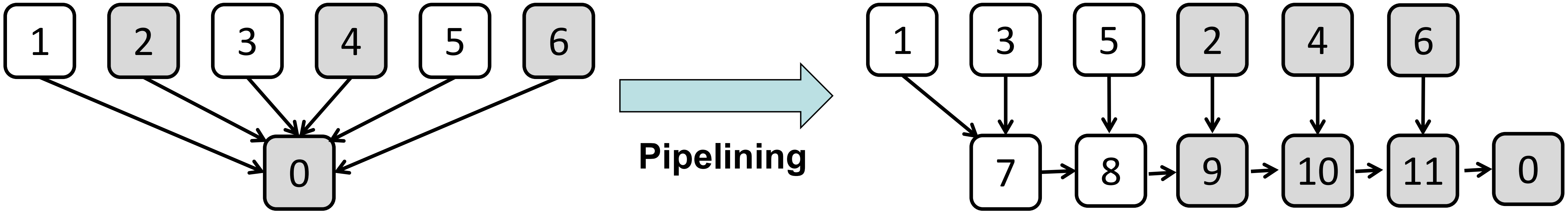


Vertices	Nodes	Tasks
v_0	N_0	**Load** b_0
v_1	N_1	**Load** b_1
v_2	N_2	**Load** b_2
v_3	N_3	**Load** b_3
v_6	N_0	**Fetch** b_1 from N_1 **Fetch** b_2 from N_2 **Fetch** b_3 from N_3 **Compute** b_4 from $\{b_0, b_1, b_2, b_3\}$ with coding coefficients {1,1,1,1}; **Compute** b_5 from $\{b_0, b_1, b_2, b_3\}$ with coding coefficients {1,2,4,8};
v_4	N_4	**Fetch** b_4 from N_0; **Persist** b_4
v_5	N_5	**Fetch** b_5 from N_0; **Persist** b_5

Automated Optimizations

- Automated BindX and BindY
  - Examines subgraph structures and calls BindX and BindY automatically

- Hierarchy awareness



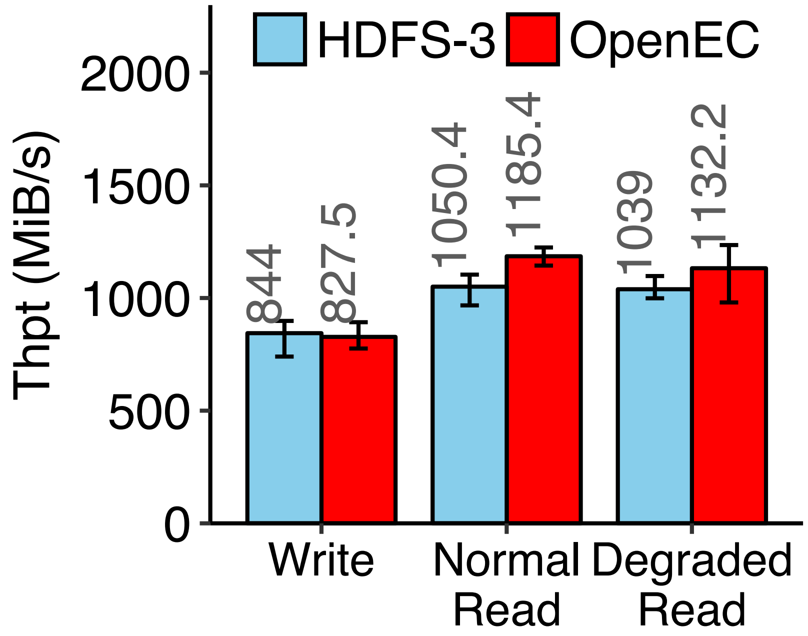
OpenEC Implementation

- Middleware layer (7000+ lines-of-code)
  - Coding operations in units of packets
  - Intel ISA-L for erasure coding
  - Redis for communications
- Integration with existing distributed storage systems
  - HDFS-RAID
  - Hadoop 3.0 HDFS
  - QFS (see technical report)
- Each integration only makes ≤ 450 lines-of-code changes
  - Changes include: (1) interfacing with systems, (2) block placement

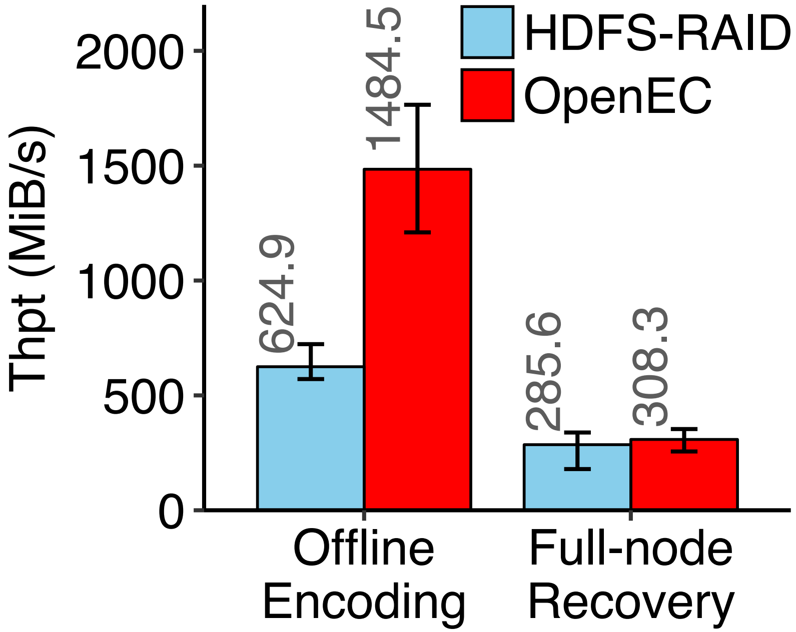
Experiments

- Local cluster
  - 16 machines
  - Quad-core 3.4 GHz Intel CPU
  - 16 GiB RAM
  - 10 Gb/s network
- Amazon EC2
  - Up to 30 instances
  - m5.xlarge instances
  - 10 Gb/s network

Basic Operations in Local Cluster



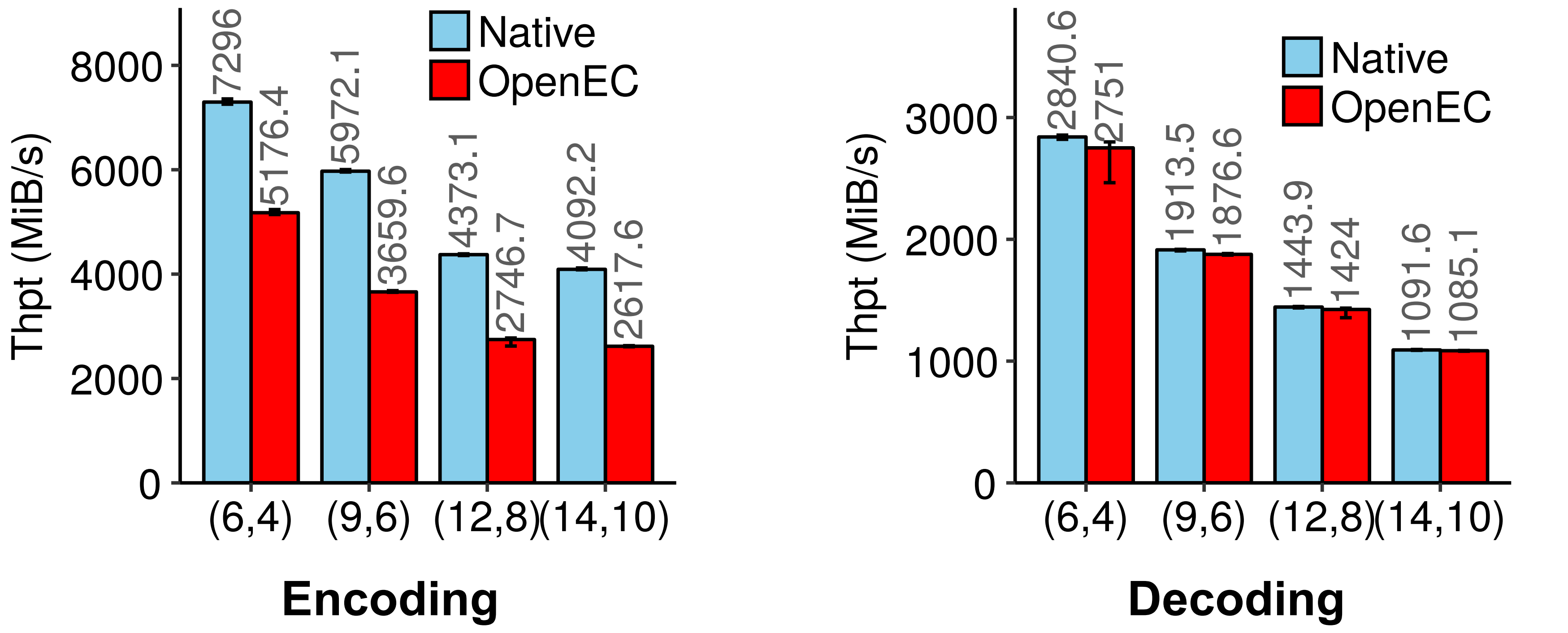
Comparisons with HDFS-3



Comparisons with HDFS-RAID

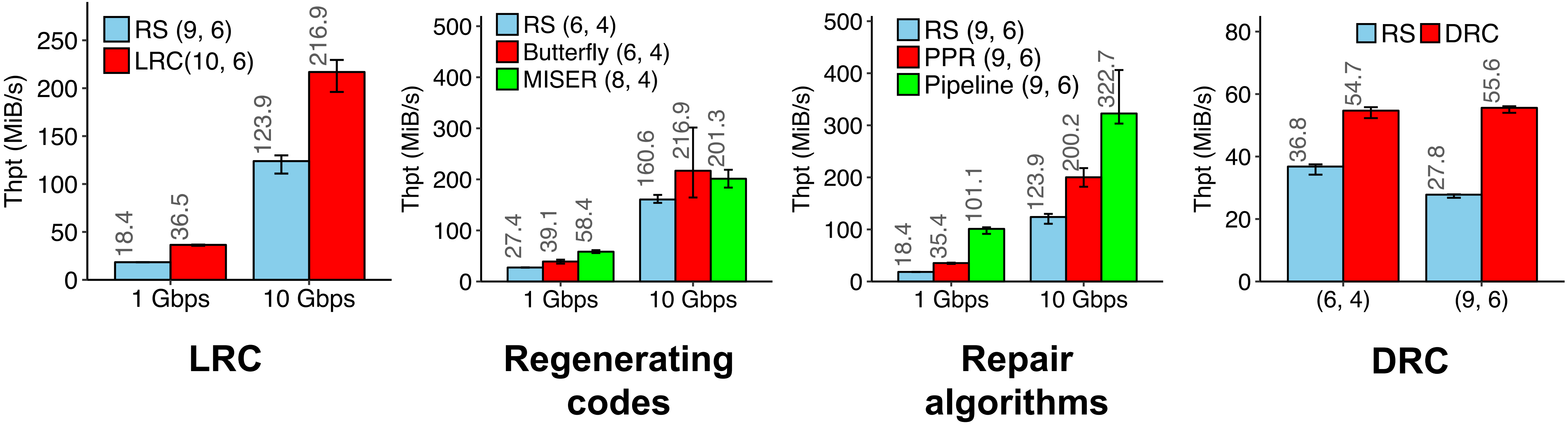
- ➢ OpenEC preserves original HDFS performance
- ➢ OpenEC achieves much faster offline encoding than HDFS-RAID with a simpler workflow

Comparisons with Native Coding (without I/O)



- ECDAG coding computations are slower than ISA-L
  - 29-38% lower in encoding; 0.6-3.15% lower in decoding
- Remains much faster than I/O; limited overhead overall

Support of Erasure Coding Designs

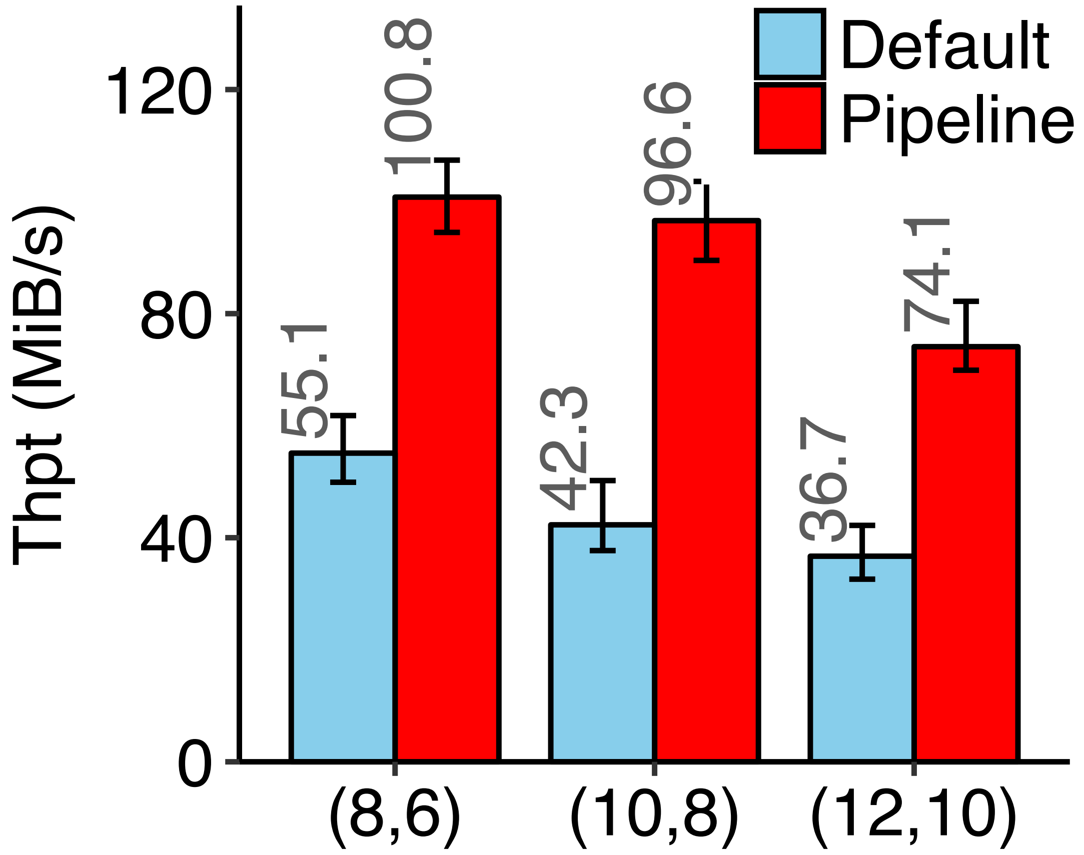


- ➢ Comparisons with six state-of-the-art erasure coding designs
- ➢ OpenEC's performance conforms to the theoretical gains in network-bound environments

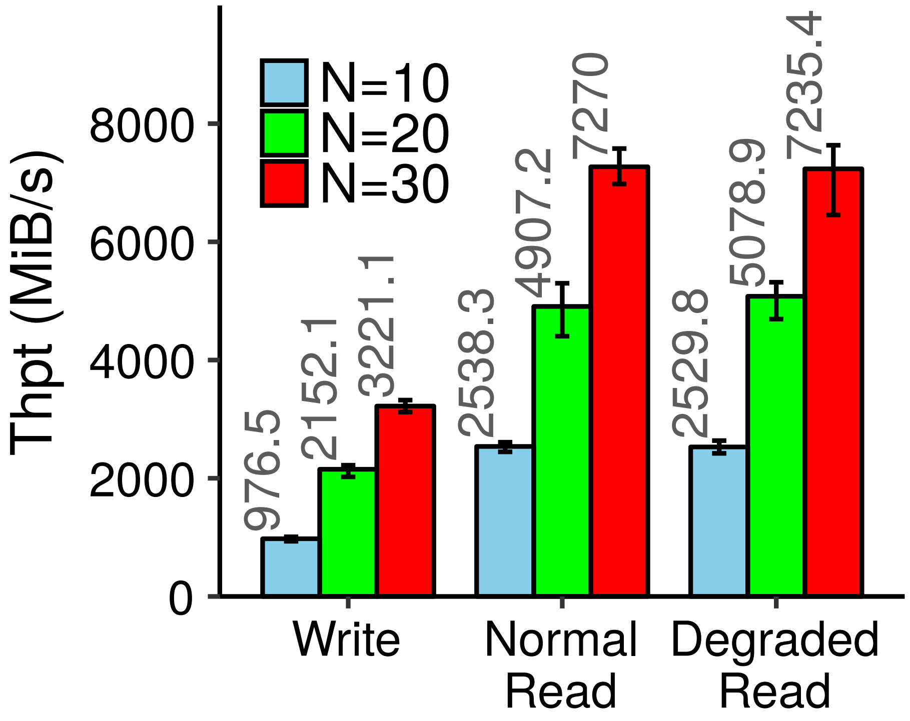
Automated Optimizations

- Automated ECDAG customization for a hierarchical topology

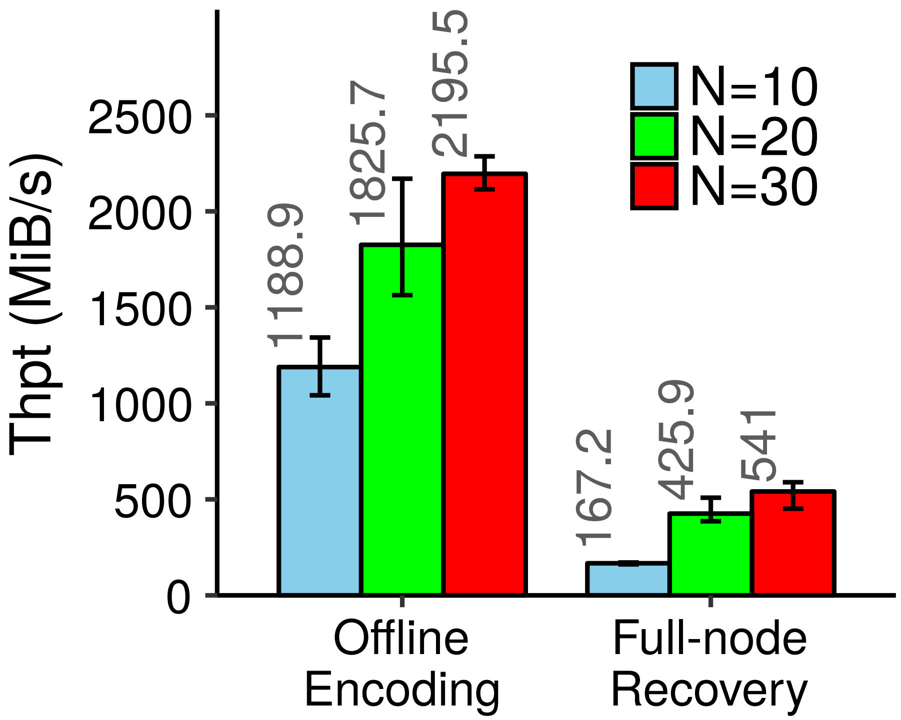
- Up to 82% repair throughput gain



Scalability in Amazon EC2



Online Encoding



Offline Encoding

➢ OpenEC scales well with number of instances

Conclusions

- OpenEC is a unified and configurable framework for flexible erasure coding management

- Future work:
  - Integration with more systems (e.g., Ceph, Swift)
  - Combined with software-defined storage for better configurability

- Source code:
  - **http://adslab.cse.cuhk.edu.hk/software/openec**

Backup

Question

- How to construct decoding ECDAGs for different combinations of lost blocks?
- The Decode() function should construct different decoding ECDAGs for two cases:
  - **Decoding one lost block**: uses any repair-efficient approach
  - **Decoding multiple lost blocks**: picks the first k available blocks

Question

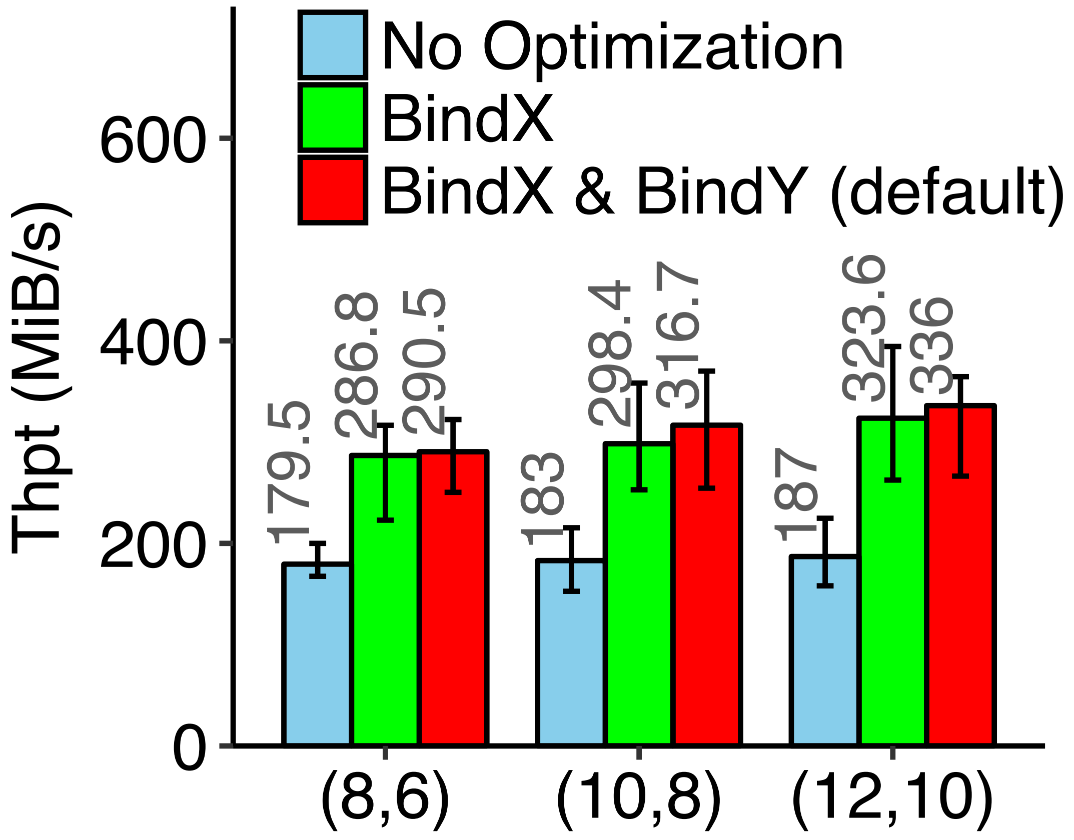
- What happens if there is a failure during repair?
- We assume that OpenEC restarts the repair process by connecting to the new set of available nodes.

Question

- What types of codes are supported or not supported?
- Supported:
  - Linear codes (e.g., RS codes, regenerating codes, LRC)
- Not supported:
  - Non-linear codes
  - Sector-disk codes

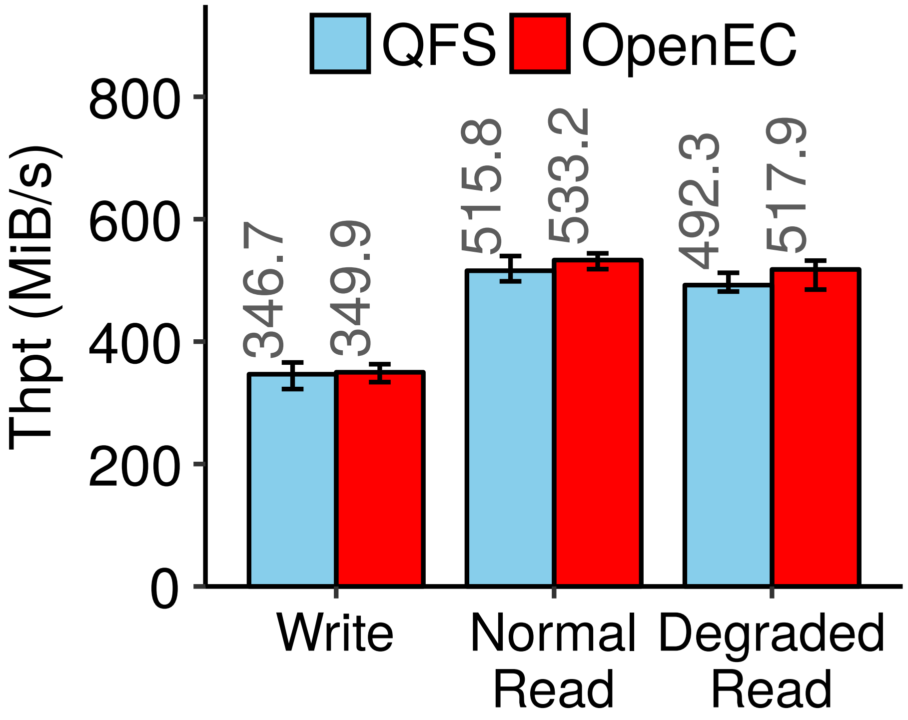
Question

➢ Performance of automated BindX and BindY?

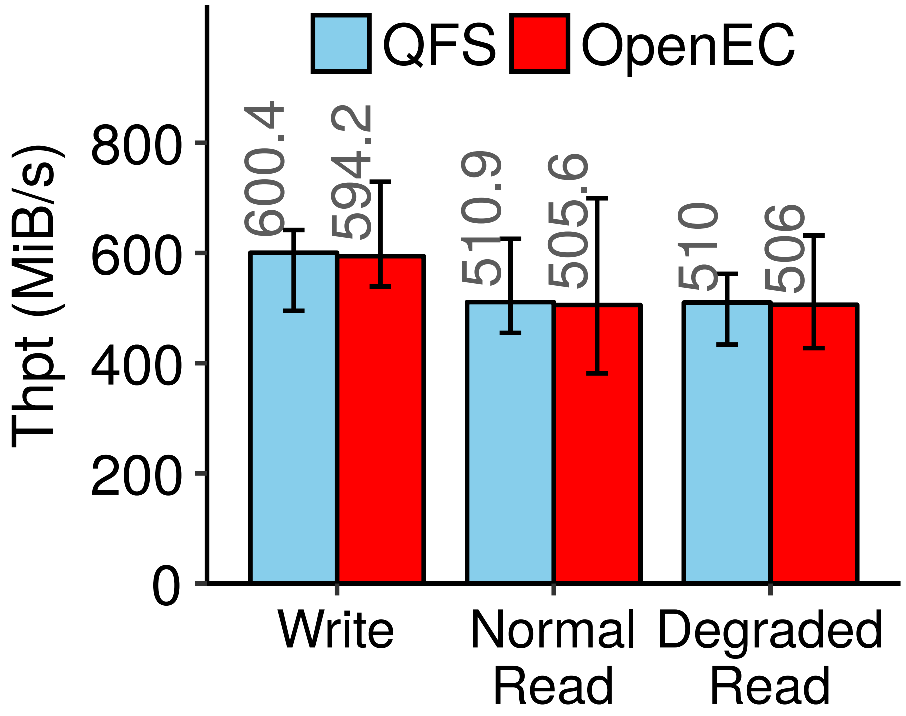


Question

➢ Performance in QFS



Single Client



Multiple Clients