Rethinking Erasure-Coding Libraries in the Age of Optimized Machine Learning

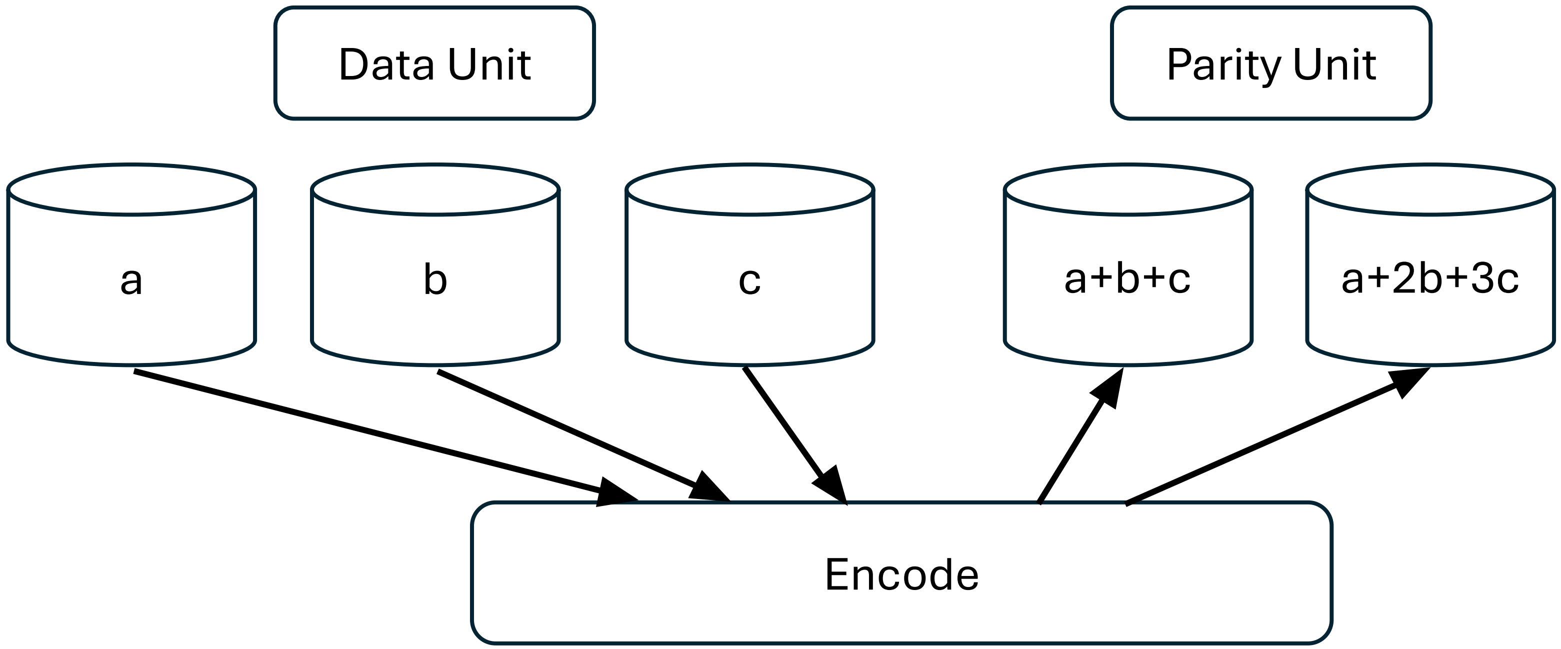
Jiyu Hu, Jack Kosaian, K. V. Rashmi
Computer Science Department
Carnegie Mellon University

Background of Storage Systems

- Failures are common in large scale data centers
- Adopts redundancy for fault-tolerance
- Erasure code achieves the same redundancy level as replication with lower storage overhead

Background of Erasure-Coding

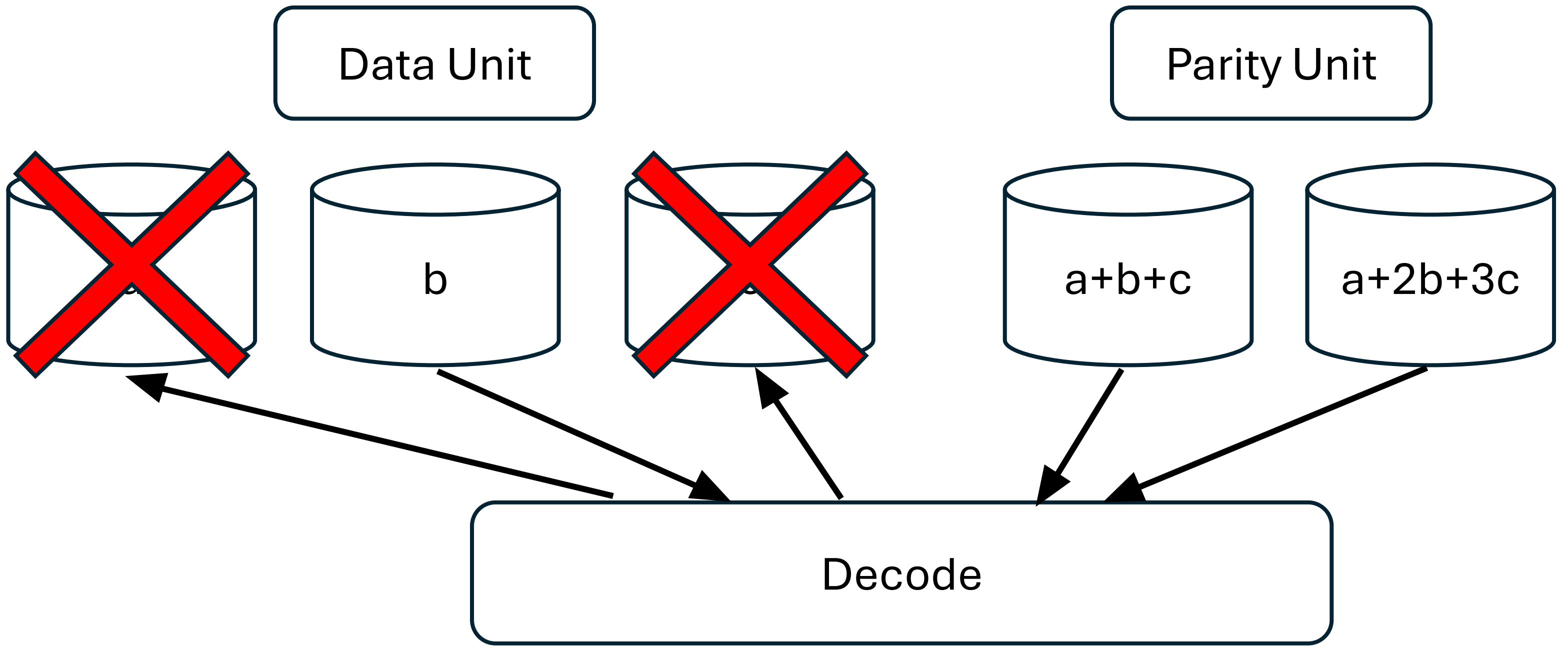
- A tool from the domain of coding theory to achieve redundancy
- Encoding:
  - party units generated as linear combination of data units
  - [n, k] code: n code unit, k data unit, r=(n-k) parity unit



n=5, k=3, r=2

Background of Erasure-Coding

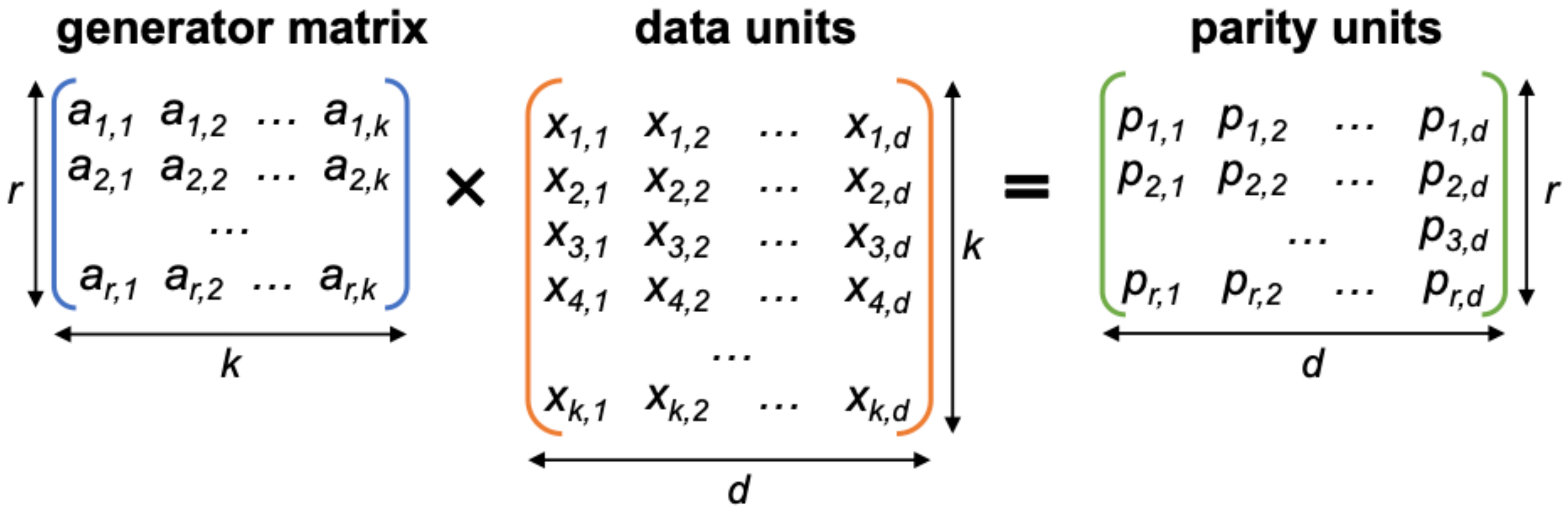
- A tool from the domain of coding theory to achieve redundancy
- Decoding (recovery): withstand up to r failures
  - Lost data units are linear combinations of remaining data and parity units



n=5, k=3, r=2

Background of Erasure-Coding

- Each parity unit can be viewed as linear combination of k data units
- Process can be viewed as matrix dot product between a generator matrix and data matrix
- The calculation is via finite-field arithmetic



Optimizations in Implementing Erasure-Coding

Algorithmic Optimizations

- **Bitmatrix erasure coding.** Convert expensive finite-field arithmetic into bitwise AND and XOR
- **Generator Matrix with fewer number of 1s**
- **Reschedule matrix calculation.** Find repetitive calculation patterns in the bitmatrix operation to minimize total number of XORs

Coding Theory

System-level optimizations

- **Vectorization**
- **Optimize memory access patterns**
- **Different hardware platforms (e.g. GPU, FPGA)**

Computer Systems

Difficulty of Developing Erasure-Coding Libraries

- Requires knowledge in both EC mathematical underpinnings and hardware features
- Hardware is becoming increasingly heterogeneous
- Growth of accelerator-native applications
- Developing EC libraries will be even more challenging in the future

Desirable Properties of Erasure-Coding Libraries

- Require less development and maintenance effort
- Can run with high performance on a variety of hardware
- Can be easily adapted to future hardware architectures

EC via Machine Learning Libraries

Idea: EC via Machine Learning Libraries

- ML libraries are well-developed and actively maintained
- ML libraries are optimized to achieve high performance on various hardware platforms
- ML libraries are frequently updated to best exploit new hardware features

EC via Machine Learning Libraries

- Erasure codes have a structure closely matching General Matrix Multiplication (GEMM)

```
for i in range(M):
    for j in range(N):
        for k in range(K):
            C[i, j] += (A[i, k] * B[k, j])
```

(Unoptimized) GEMM

```
for i in range(ec_r * ec_w):
    for j in range(ec_d):
        for k in range(ec_k * ec_w):
            C[i, j] ^= (A[i, k] & B[k, j])
```

(Unoptimized) bitmatrix erasure code encoding

Implementation using TVM – TVM-EC

- TVM[1]: an open-source framework for optimizing neural networks

    - Generate high-performance kernels for multiple hardware platforms
    - Performs learning-based autotuning based on underlying hardware

[1]: Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18).

Implementation using TVM – TVM-EC

- Procedure
  - Declare placeholder variables
  - Define bitmatrix computation
  - Autotune on specific hardware
  - Compute
    - Convert the data matrix into bitmatrix
    - Bitmatrix multiplication with generator matrix

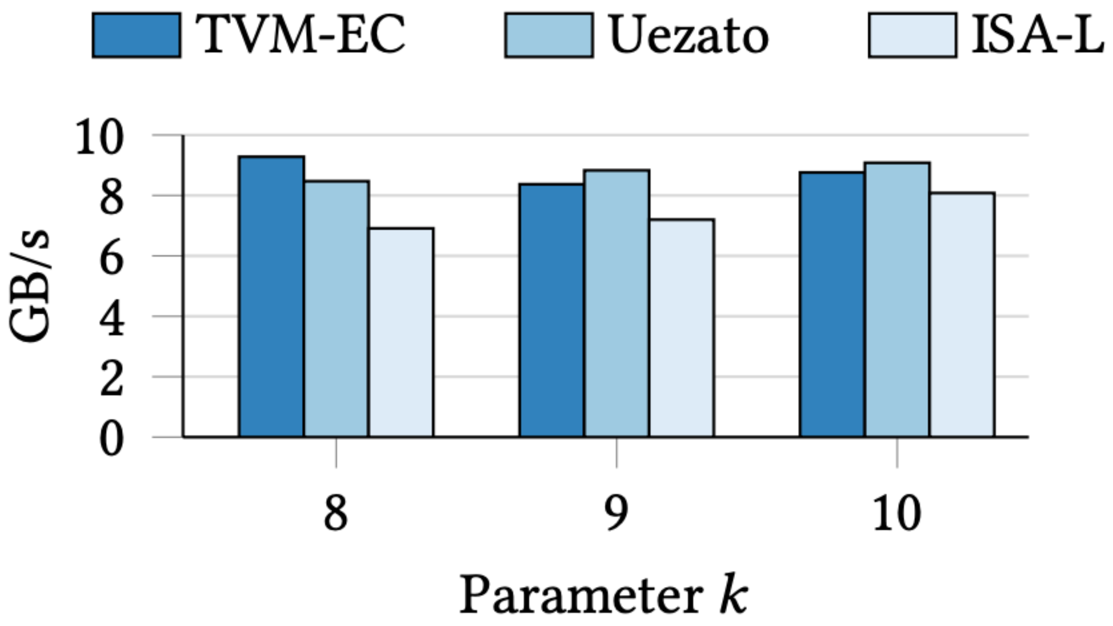
```
A = te.placeholder((M, K), name="A")
B = te.placeholder((K, N), name="B")
k = te.reduce_axis((0, K), name="k")

# GEMM
te.compute((M, N),
    lambda i,j: sum(A[i,k] * B[k,j], axis=k))

# Bitmatrix erasure code
xor = te.comm_reducer(lambda i,j: i ^ j, name="xor")
te.compute((M, N),
    lambda i,j: xor(A[i,k] & B[k,j], axis=k))
```

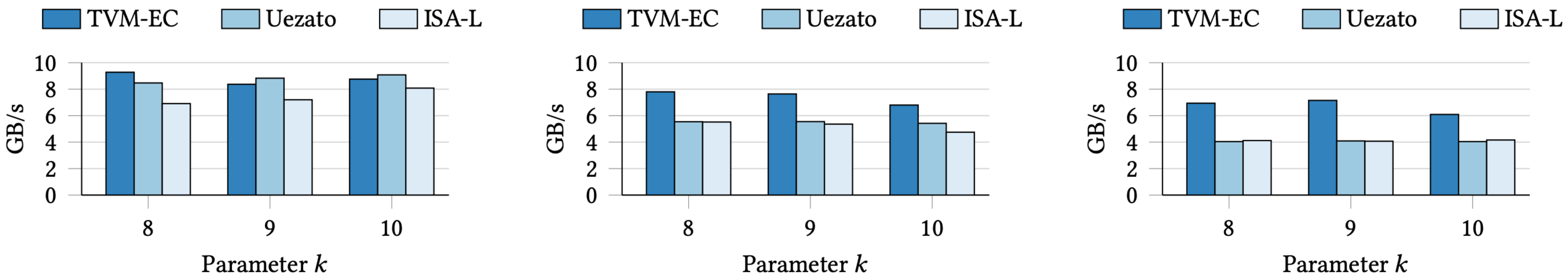
Evaluation

- Platform: eight-core Intel Xeon at 2.0 GHz with 64 GB of memory
- Baselines:
  - Uezato: state-of-the-art hand-optimized (research) EC library
  - Intel ISA-L: production-grade EC library optimized for CPUs



Evaluation

- Effect of parameter r
- TVM-EC shows better relative performance with larger parameter r
- Up to 1.75✕ throughput with TVM-EC when r = 4 compared to Uezato and ISA-L



(a) $r = 2$ **(b)** $r = 3$ **(c)** $r = 4$

Discussion

- Integration Effort
  - ML libraries in high-level languages, storage systems in low-level languages
  - Modifications might be required for the target storage system
    - ML-library-specific data structure
    - Change data layout for faster data retrieval for the EC library

- Potential limitations
  - EC specific (algorithmic) optimizations are hard to apply
  - GEMM-like optimizations may lead to higher CPU utilization

Conclusion

- Developing optimized EC libraries is hard
  - Understanding of mathematical underpinnings and hardware features
  - High performance on different hardware platforms
  - Frequent updates to include new hardware features

- Presented a case for automating development of EC libraries using ML libraries
  - Eases the development and maintenance effort
  - Feasible due to similar mathematical operations performed

- Implementation using TVM: TVM-EC

- Evaluation comparing to state-of-the-art EC libraries
  - Achieves up to 1.75× performance benefit over state-of-the-art EC libraries

Future Work

- Explore other classes of codes
- More full-pledged evaluation
  - Different r and w parameters
  - CPU utilization comparison
- Investigate the effect of learning-based tuning
- Develop prototypes on more varieties of hardware
- Integrate our prototypes into real storage systems