EC-Store
Bridging the Gap Between Storage and Latency in Distributed Erasure Coded Systems

Michael Abebe
Khuzaima Daudjee
Brad Glasbergen
Yuanfeng Tian

ICDCS
July 3, 2018
Failures are the Norm

1-5% of cluster unavailable per day
Replicated Storage Systems

S1  Servers  S6

Data Block
Replicated Storage Systems

S1

Servers

Replicate

S6

Data Block
Replicated Storage Systems

Read from Replica

Data Block
Replicated Storage Systems

Servers

Read from Available Replica

Data Block

Unavailable Server

S1

S6
Overhead of Replicated Storage

To tolerate $R$ failures need $R + 1$ copies of data
Towards Lower Overhead Storage

Data Block
Towards Lower Overhead Storage

Fragment into $K$ chunks

Data Block

1
2
$\ldots$
$K$
Towards Lower Overhead Storage

Fragment into $K$ chunks

$K = \# \text{ chunks needed for reconstruction}$

1

2

...  

$K$

Data Block
Towards Lower Overhead Storage

Fragment into $K$ chunks

$K = \# \text{ chunks needed for reconstruction}$

1
2
...
K

Data Block

Generate $R$ parity chunks

K+1
...
K+R
Towards Lower Overhead Storage

Fragment into $K$ chunks

$K = \# \text{ chunks needed for reconstruction}$

Generate $R$ parity chunks

$R = \# \text{ faults tolerated}$

Data Block

1

2

$\vdots$

$K$

$K+1$

$\vdots$

$K+R$
Towards Lower Overhead Storage

To tolerate $R$ faults, a storage overhead of $\frac{(K+R)}{K}$

$K = \# \text{ chunks needed for reconstruction}$

$R = \# \text{ faults tolerated}$

$1$

$2$

$\ldots$

$K$

$K+1$

$\ldots$

$K+R$
Erasure Coded Storage Systems

Encode and Store

Data Block

(K=2, R=2)
Erasure Coded Storage Systems

Retrieve and decode

(Data Block)

(K=2, R=2)
Erasure Coded Storage Systems

Read any 2 of 4!

Retrieve and decode

(K=2, R=2)
Erasure Coded Storage Systems

Read any 2 of 4!
Storage Overhead = 2

Data Block

S1

S6

Retrieve and decode

(K=2, R=2)
Breakdown of Response Times (YCSB)

Data access latencies are main difference
Straggling Data Access

Retrieve and decode (K=2, R=2)
Straggling Data Access

Retrieve, but cannot decode until retrieve 2 chunks

Data Block

(K=2, R=2)
Straggling Data Access

Retrieve (K=2, R=2)

Avoid slow servers

Data Block

Servers

S1

S6

Straggling Server
Latency and Storage Gap

Replication
  + Fast data retrieval
  - High storage overhead
Latency and Storage Gap

Replication
  + Fast data retrieval
  - High storage overhead

Erasure Coding
  + Low storage overhead
  - Slow data retrieval
Latency and Storage Gap

Replication
  + Fast data retrieval
  - High storage overhead

Erasure Coding
  + Low storage overhead
  - Slow data retrieval
How to Improve Storage System Performance

• *Intelligently access data* to minimize access costs

• Reduce number of sites accessed through data co-access

• *Dynamically move data* to improve future accesses
Data Access Strategies

- Data Access Strategies
- Data Movement Strategies

- Complementary Strategies
- Experimental Evaluation

ICDCS
July 3, 2018
The Netherlands, also known informally as Holland, is a country in Western Europe with a population of seventeen million. It is the main constituent country of the Kingdom of the Netherlands, which is further comprised of three island territories in the Caribbean: Bonaire, Sint Eustatius and Saba.

Gallery of Dutch Art
Initial Data Placement

(K=2, R=1)
Initial Data Placement

(K=2, R=1)
Multi-Item Data Access Strategy

(K=2, R=1)
Multi-Item Data Access Strategy

Retrieve: A, B, C

(K=2, R=1)
Multi-Item Data Access Strategy

Retrieve: A, B, C

(K=2, R=1)
Intelligent Multi-Item Data Access Strategy

Retrieve: A, B, C

Previously Retrieved: A, B, C

(K=2, R=1)
Intelligent Multi-Item Data Access Strategy

Retrieve: A, B, C
Minimizes number of sites accessed, reducing likelihood of stragglers

Previously Retrieved: A, B, C

(K=2, R=1)
Intelligent Data Access Strategy

- Retrieve enough \((K)\) chunks to reconstruct block
Intelligent Data Access Strategy

- Retrieve enough \(K\) chunks to reconstruct block

- Quantify cost of access
  - Cost of accessing site (load)
  - Cost of accessing chunk at site (I/O)
Intelligent Data Access Strategy

• Retrieve enough \((K)\) chunks to reconstruct block

• Quantify cost of access
  • Cost of accessing site (load)
  • Cost of accessing chunk at site (I/O)

• Find minimal cost access strategy with Integer Linear Programming

[Details in Paper tiny.cc/ecstore]
Data Movement Strategies

- Data Access Strategies
- Data Movement Strategies
- Complementary Strategies
- Experimental Evaluation

ICDCS
July 3, 2018
Workload Change

What if: A, E becomes a popular access pattern?
Workload Change

What if: A, E becomes a popular access pattern?

Retrieve: A, E

(K=2, R=1)
Can We Improve Access Costs?

What if: A, E becomes a popular access pattern?

Retrieve: A, E

We could move: A₁, E₂, A₃, E₁

(K=2, R=1)
Can We Improve Access Costs?

What if: $A, E$ becomes a popular access pattern?

Retrieve: $A, E$

We could move: $E_2$

Where to move?

(K=2, R=1)
Data Movement Decisions

(K=2, R=1)
Data Movement Decisions

Retrieve: A, E

(K=2, R=1)
Data Movement Decisions

Retrieve: A, E

Previously Retrieved: D, E

(K=2, R=1)
Data Movement Decisions

Retrieve: A, E

Previously Retrieved: D, E

Currently Retrieve: D, E

(K=2, R=1)
Data Movement Decisions

Movement should minimize cost over all known access patterns

Retrieve: A, E

Previously Retrieved: D, E

Currently Retrieve: D, E

(K=2, R=1)
Improved Data Movement Decisions

Movement should minimize cost over all known access patterns

Previously Retrieved: D, E

(K=2, R=1)
Improved Data Movement Decisions

Movement should minimize cost over all known access patterns

(K=2, R=1)
Improved Data Movement Decisions

Movement should minimize cost over all known access patterns

Currently Retrieve: D, E

(K=2, R=1)
Dynamic Data Movement

- Move data in response to changes in access patterns and load
Dynamic Data Movement

• Move data in response to changes in access patterns and load

• Consider future access costs and load balance

Details in Paper

tiny.cc/ecstore
Dynamic Data Movement

• Move data in response to changes in access patterns and load

• Consider future access costs and load balance

• Move recently or frequently accessed chunks to sites with co-accessed chunks or lighter load

Details in Paper tiny.cc/ecstore
Complementary Strategies

- Data Access Strategies
- Data Movement Strategies
- Complementary Strategies
- Experimental Evaluation
Standard Request Model

Request chunks from: K sites

Wait for K responses

(K=2, R=2)
Standard Request Model

Request chunks from: K sites

Wait for K responses

(K=2, R=2)
Late Binding Model

Request chunks from: $K + \Delta$ sites

Wait for first $K$ responses

(K=2, R=2)
Intelligent Access Strategy + Late Binding Model

Request chunks from: $K + \Delta$ sites

Intelligently select $\Delta$ extra requests

Wait for first $K$ responses

Details in Paper tiny.cc/ecstore

(K=2, R=2)
Experimental Evaluation

- Data Access Strategies
- Data Movement Strategies
- Complementary Strategies
- Experimental Evaluation
Experimental Setup

x 32 Storage Nodes

System should tolerate 2 faults
Experimental Setup

- Replication

- Erasure Coding
  - + Late Binding

System should tolerate 2 faults
Experimental Setup

- Replication
- Erasure Coding
  - + Late Binding
- Erasure Coding
  - + Access Strategies (EC-Store Access)
  - + Movement Strategies (EC-Store)
  - + Late Binding (EC-Store + LB)

System should tolerate 2 faults

x 32 Storage Nodes
Response time Over time (YCSB)

- **Erasure Coding**
- **EC-Store Access**
- **Replication**
- **EC-Store**
Response time Over time (YCSB)

![Graph showing response time over time with different replication methods: Erasure Coding, EC-Store Access, Replication, EC-Store.](image)
Response time Over time (YCSB)

Improves over time!

Erasure Coding
EC-Store Access
Replication
EC-Store
Breakdown of Response Times (YCSB)

- Replication
- Erasure Coding
- EC + Late Binding
- EC-Store Access
- EC-Store + Late Binding

<table>
<thead>
<tr>
<th>Component</th>
<th>Avg. response time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata access</td>
<td>20.9</td>
</tr>
<tr>
<td>Access planning</td>
<td>31.9</td>
</tr>
<tr>
<td>Chunk retrieval</td>
<td>24.2</td>
</tr>
<tr>
<td>Block decoding</td>
<td>27.4</td>
</tr>
<tr>
<td>EC-Store Access</td>
<td>17.4</td>
</tr>
<tr>
<td>EC-Store + Late Binding</td>
<td>14.9</td>
</tr>
</tbody>
</table>
Breakdown of Response Times (YCSB)

Latency reduction

<table>
<thead>
<tr>
<th>Category</th>
<th>Avg. response time (ms)</th>
<th>Latency reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication</td>
<td>20.9</td>
<td>15%</td>
</tr>
<tr>
<td>Erasure Coding</td>
<td>31.9</td>
<td>45%</td>
</tr>
<tr>
<td>EC + Late Binding</td>
<td>24.2</td>
<td>30%</td>
</tr>
<tr>
<td>EC-Store Access</td>
<td>27.4</td>
<td></td>
</tr>
<tr>
<td>EC-Store + Late Binding</td>
<td>17.4</td>
<td></td>
</tr>
<tr>
<td>EC-Store + Late Binding</td>
<td>14.9</td>
<td></td>
</tr>
</tbody>
</table>

- Metadata access
- Access planning
- Chunk retrieval
- Block decoding
Tail Latencies (Wikipedia)

![Graph showing tail latencies for different storage technologies: Replication, Erasure Coding, EC-Store Access, EC-Store, EC+Late Binding, and EC-Store + LB. The x-axis represents percentile and the y-axis represents average response time in milliseconds.]
EC-Store Takeaways

- Achieves low storage overhead and low latency data access

- EC-Store uses dynamic data access and data movement strategies for erasure coded storage systems
EC-Store Takeaways

• Achieves low storage overhead and low latency data access

• EC-Store uses dynamic data access and data movement strategies for erasure coded storage systems

---

EC-Store: Bridging the Gap Between Storage and Latency in Distributed Erasure Coded Systems

Michael Abebe, Khuzaima Daudjee, Budel Glasbergen, Yung-Feng Tian
Chenian School of Computer Science, University of Waterloo
{mabebe, kdaudjee, bglasbe, yftian}@watarloo.ca

Fig. 1: Response time breakdown for replication and erasure coding under skewed access. Data retrieval times dominate the overall response time.

The need to store and retrieve big data has fueled the development and adoption of cloud storage systems. In cloud deployments, however, machines frequently experience downtime. For example, Google observed that at any point in time, up to 5% of the nodes within their storage system were shut down [11]. To ensure data remains available in the presence of these failures, systems must be fault tolerant. Large-scale distributed storage systems typically provide fault tolerance either by replicating [4,14] or erasure encoding data [11,15,19,23,95,52]. Replication creates complete copies of data, incurring a significant storage overhead over erasure coding that partitions data and stores the partitions on parity fragments on multiple nodes to provide the same level of fault tolerance as replication. Consequently, while erasure encoding stores less data, accessing it requires additional retrieval resulting in an increase in data access cost compared to replication [15].

To demonstrate that performance in erasure-coded distributed storage systems is largely determined by the cost of data retrieval, we show a breakdown of average response times in Figure 1 for a workload that retrieves multiple 100 KB blocks. The response time is divided into four categories: the cost of locating data (metadata access), determining which data chunks to retrieve (access planning), retrieving data, and decoding data. As Figure 1 shows, the performance difference between replication and erasure coding is primarily due to the time it takes to retrieve data, which dominates overall response times. However, while both systems can tolerate the same number of faults (two), in the example of Figure 1, the replicated system stores 50% more data than the erasure-coded system. These differences motivate a fault-tolerant storage system that can achieve the best of both worlds: low storage overhead and low latency data retrieval.

When compared to replicated data storage, retrieval costs are higher for erasure coded storage systems because of the effects of straggling: the time taken to retrieve the slowest, or straggling, data chunk dominates retrieval time [19,29]. Even when parallelism is leveraged, straggler effects are pronounced in systems that must wait for multiple requests to complete e.g., in erasure coded storage systems that are deployed in distributed environments, concurrent clients issuing requests in parallel over the distributed storage system inevitably result in the occurrence of stragglers [9].

In our erasure-coded storage system, EC-Store, we propose a novel approach to the straggling problem by intelligently selecting chunks to retrieve so as to avoid stragglers. This Dynamic Data Access Strategy uses chunk location information to generate a cost-effective strategy on-the-fly for data retrieval. By incorporating this strategy in EC-Store, we reduce data access latencies and satisfy our best of both worlds goal.

To mitigate the effects of stragglers, some systems use a late binding strategy [19,38,49] in which additional requests are made and the slower responses are ignored. Late binding can reduce response time but places additional load on the system, responses that will be ignored must still be generated. In contrast, EC-Store’s dynamic data access strategy offers excellent performance and reduces this additional load on the system.