EC-Cache: Load-balanced, Low-latency Cluster Caching with Online Erasure Coding

Rashmi Vinayak
UC Berkeley

Joint work with

Mosharaf Chowdhury, Jack Kosaian (U Michigan)
Ion Stoica, Kannan Ramchandran (UC Berkeley)
Caching for data-intensive clusters

- Data-intensive clusters rely on **distributed, in-memory caching** for high performance
  - Reading from memory orders of magnitude faster than from disk/ssds
  - Example: Alluxio
Imbalances prevalent in clusters

Sources of imbalance:

• Skew in object popularity
• Background network imbalance
• Failures/unavailabilities
Imbalances prevalent in clusters

Sources of imbalance:

• Skew in object popularity
• Background network imbalance
• Failures/unavailabilities

Small fraction of objects highly popular

- Zipf-like distribution
- Top 5% of objects 7x more popular than bottom 75%†
  (Facebook and Microsoft production cluster traces)

†Anantthanarayanan et al. NSDI 2012
Imbalances prevalent in clusters

Sources of imbalance:

• Skew in object popularity

• Background network imbalance

• Failures/unavailabilities

Some parts of the network more congested than others

- Ratio of maximum to average utilization more than 4.5x
  with > 50% utilization

  (Facebook data-analytics cluster)
Imbalances prevalent in clusters

Sources of imbalance:

- Skew in object popularity
- **Background network imbalance**
- Failures/unavailabilitys

Some parts of the network more congested than others

- Ratio of maximum to average utilization more than 4.5x with > 50% utilization
  (Facebook data-analytics cluster)
- Similar observations from other production clusters†

† Chowdhury et al. SIGCOMM 2013
Imbalances prevalent in clusters

Sources of imbalance:

• Skew in object popularity
• Background load imbalance
• Failures/unavailabilities

Norm rather than the exception

- median > 50 machine unavailability events every day in a cluster of several thousand servers†
  (Facebook data analytics cluster)

†Rashmi et al. HotStorage 2013
Imbalances prevalent in cluster

Sources of imbalance:

- Skew in object popularity
- Background network imbalance
- Failures/unavailabilities

➡ Adverse affects:
- create load imbalance
- degrade read latency performance
Imbalances prevalent in cluster

Sources of imbalance:

- Skew in object popularity
- Background network imbalance
- Failures/unavailabilities

→ Adverse affects:
  - create load imbalance
  - degrade read latency performance

Single copy in memory not sufficient to get good performance
Popular approach: Selective Replication

- Caching replicas of objects based on their popularity
  - more replicas for more popular objects
Popular approach: Selective Replication

- Caching replicas of objects based on their popularity
  - more replicas for more popular objects
Popular approach: Selective Replication

• Caching replicas of objects based on their popularity
  - more replicas for more popular objects
Popular approach: Selective Replication

- Caching replicas of objects based on their popularity
  - more replicas for more popular objects

- Used in data-intensive clusters† as well as widely used in key-value stores for many web-services such as Facebook Tao‡

†Ananthanarayanan et al. NSDI 2011, ‡Bronson et al. ATC 2013
Read performance & Load balance
Read performance & Load balance

Memory Overhead

Single copy

Selective replication
Quick primer on erasure coding
Quick primer on erasure coding

• Takes in $k$ data units and creates $r$ “parity” units
Quick primer on erasure coding

• Takes in $k$ data units and creates $r$ “parity” units

• *Any* $k$ of the $(k+r)$ units are sufficient to decode the original $k$ data units
Quick primer on erasure coding

- Takes in $k$ data units and creates $r$ “parity” units

- **Any $k$** of the $(k+r)$ units are sufficient to decode the original $k$ data units

\[
\begin{align*}
\text{data units} & \quad \text{parity units} \\
\text{d1} & \quad \text{d2} & \quad \text{d3} & \quad \text{d4} & \quad \text{d5} \quad \text{p1} & \quad \text{p2} & \quad \text{p3} & \quad \text{p4} \\
\end{align*}
\]

- $k = 5$
- $r = 4$
Quick primer on erasure coding

• Takes in \( k \) data units and creates \( r \) “parity” units

• \textbf{Any} \( k \) of the \((k+r)\) units are sufficient to decode the original \( k \) data units

\[\begin{array}{ccccccc}
\text{data units} & & & & & & \text{parity units} \\
d1 & d2 & d3 & d4 & d5 & p1 & p2 & p3 & p4 \\
\end{array}\]

- \( k = 5 \)
- \( r = 4 \)
Quick primer on erasure coding

- Takes in \( k \) data units and creates \( r \) “parity” units

- \textbf{Any} \( k \) of the \((k+r)\) units are sufficient to decode the original \( k \) data units

- \( k = 5 \)
- \( r = 4 \)
Quick primer on erasure coding

• Takes in *k* data units and creates *r* “parity” units

• *Any k* of the (k+r) units are sufficient to decode the original k data units

![Diagram showing data units and parity units with k=5 and r=4]
Quick primer on erasure coding

- Takes in $k$ data units and creates $r$ “parity” units
- Any $k$ of the $(k+r)$ units are sufficient to decode the original $k$ data units

```plaintext
Read

<table>
<thead>
<tr>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>d4</th>
<th>d5</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>p4</th>
</tr>
</thead>
</table>

• $k = 5$
• $r = 4$

Decode```
Quick primer on erasure coding

- Takes in $k$ data units and creates $r$ “parity” units
- *Any* $k$ of the $(k+r)$ units are sufficient to decode the original $k$ data units

![Diagram showing data units and parity units with k=5 and r=4]
EC-Cache bird’s eye view: Writes
EC-Cache bird’s eye view: Writes

Put

X
EC-Cache bird’s eye view: Writes

- Object \textit{split} into \( k \) data units
EC-Cache bird’s eye view: Writes

- **Object** split into k data units
- **Encoded** to generate r parity units

Diagram:

- **Put** X
- **Split**
  - d1
  - d2
  - k = 2
- **Encode**
  - d1
  - d2
  - p1
  - k = 2
  - r = 1

...
EC-Cache bird’s eye view: Writes

- Object **split** into \( k \) data units
- **Encoded** to generate \( r \) parity units
- \((k+r)\) units cached on distinct servers chosen at **random**
EC-Cache bird’s eye view: Reads
EC-Cache bird’s eye view: Reads

- Read from \((k + \Delta)\) units of the object chosen at random
  - “Additional reads”
- Use the first \(k\) units that arrive
EC-Cache bird’s eye view: Reads

- Read from \((k + \Delta)\) units of the object chosen at random
  - “Additional reads”
- Use the first \(k\) units that arrive
EC-Cache bird’s eye view: Reads

- Read from \((k + \Delta)\) units of the object chosen at random
  - “Additional reads”
- Use the first \(k\) units that arrive
EC-Cache bird’s eye view: Reads

- Read from \((k + \Delta)\) units of the object chosen at random
  - “Additional reads”
- Use the first \(k\) units that arrive

\[
\begin{align*}
k &= 2 \\
r &= 1 \\
\Delta &= 1 \\
k + \Delta &= 3
\end{align*}
\]
EC-Cache bird’s eye view: Reads

- Read from \((k + \Delta)\) units of the object chosen at random
  - “Additional reads”
- Use the first \(k\) units that arrive

\[ k = 2 \quad r = 1 \]
\[ \Delta = 1 \]
\[ k + \Delta = 3 \]
EC-Cache bird’s eye view: Reads

- Read from \((k + \Delta)\) units of the object chosen at random
  - “Additional reads”
- Use the first \(k\) units that arrive
EC-Cache bird’s eye view: Reads

- Read from \((k + \Delta)\) units of the object chosen at random
  - “Additional reads”
- Use the first \(k\) units that arrive
- Decode the data units

\[
k = 2 \\
r = 1 \\
\Delta = 1 \\
k + \Delta = 3
\]
EC-Cache bird’s eye view: Reads

- Read from \((k + \Delta)\) units of the object chosen at random
  - “Additional reads”
- Use the first \(k\) units that arrive
- Decode the data units
- Combine the decoded units
Erasure coding: Why and How?
Erasure coding: Why and How?

1. Finer control over memory overhead
   - Selective replication allows only integer control
   - Erasure coding allows fractional control
   - E.g., $k = 10$ allows increments of 0.1
1. **Finer control over memory overhead**
   - Selective replication allows only integer control
   - Erasure coding allows fractional control
   - E.g., $k = 10$ allows increments of 0.1

2. **Object splitting helps in load balancing**
   - Smaller granularity reads help to smoothly spread load
   - Analysis on a simplified model:
     \[
     \frac{\text{Var}(L_{\text{EC-Cache}})}{\text{Var}(L_{\text{Selective Replication}})} = \frac{1}{k}
     \]
3. **Object splitting reduces median latency but hurts tail latency**
   - **Read parallelism** helps reduce median latency
   - **Straggler effect** hurts tail latency (without additional reads, i.e., $\Delta=0$)
3. Object splitting reduces median latency but hurts tail latency
   - Read parallelism helps reduce median latency
   - Straggler effect hurts tail latency (without additional reads, i.e., $\Delta=0$)

4. “Any-k-out-of-n” property helps to reduce tail latency
   - Read from $(k + \Delta)$ and use the first $k$ that arrive
   - $\Delta = 1$ sufficient to reign in tail latency
Design considerations
## Design considerations

### 1. Purpose of erasure codes

<table>
<thead>
<tr>
<th>Storage systems</th>
<th>EC-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Fault tolerance</td>
<td>• Reduce latency</td>
</tr>
<tr>
<td></td>
<td>• Load balance</td>
</tr>
</tbody>
</table>
Design considerations

2. Choice of erasure code

<table>
<thead>
<tr>
<th>Storage systems</th>
<th>EC-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Design considerations

2. Choice of erasure code

<table>
<thead>
<tr>
<th>Storage systems</th>
<th>EC-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimize resource usage during reconstruction operations(^\d)</td>
<td></td>
</tr>
<tr>
<td>Some codes do not have “any-k-out-of-n” property</td>
<td></td>
</tr>
</tbody>
</table>

\(^\d\)Rashmi et al. SIGCOMM 2014, Sathiamoorthy et al. VLDB 2013, Huang et al. ATC 2012
## Design considerations

### 2. Choice of erasure code

<table>
<thead>
<tr>
<th>Storage systems</th>
<th>EC-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Optimize resource usage during reconstruction operations†</td>
<td>• No reconstruction operations in caching layer; data persisted in underlying storage</td>
</tr>
<tr>
<td>• Some codes do not have “any-k-out-of-n” property</td>
<td>• “Any-k-out-of-n” property helps in load balancing and reducing latency when reading objects</td>
</tr>
</tbody>
</table>

†Rashmi et al. SIGCOMM 2014, Sathiamoorthy et al. VLDB 2013, Huang et al. ATC 2012
Design considerations

3. **Encoding across vs. within objects**

<table>
<thead>
<tr>
<th>Storage systems</th>
<th>EC-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Design considerations

#### 3. Encoding across vs. within objects

<table>
<thead>
<tr>
<th>Storage systems</th>
<th>EC-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Some systems encode across objects (e.g., HDFS-RAID); some within (e.g., Ceph)</td>
<td></td>
</tr>
</tbody>
</table>
3. **Encoding across vs. within objects**

<table>
<thead>
<tr>
<th>Storage systems</th>
<th>EC-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Some systems encode across objects (e.g., HDFS-RAID); some within (e.g., Ceph)</td>
<td>• Across objects not suitable; need to encode within</td>
</tr>
<tr>
<td>• Does not affect fault tolerance</td>
<td>- To spread load across both data &amp; parity</td>
</tr>
<tr>
<td></td>
<td>- Encoding across: Too high BW overhead for reading object using parities†</td>
</tr>
</tbody>
</table>

†Rashmi et al. SIGCOMM 2014, HotStorage 2013
Implementation

• EC-Cache on top of Alluxio
  - Backend servers: cache data — unaware of erasure coding
  - EC-Cache client library: all read/write logic handled
Implementation

- EC-Cache on top of Alluxio
  - Backend servers: cache data — unaware of erasure coding
  - EC-Cache client library: all read/write logic handled

- Reed-Solomon code
  - Any-k-out-of-n property
Implementation

• EC-Cache on top of Alluxio
  - Backend servers: cache data — unaware of erasure coding
  - EC-Cache client library: all read/write logic handled

• Reed-Solomon code
  - Any-k-out-of-n property

• Intel ISA-L hardware acceleration library
  - Fast encoding and decoding
  - Critical as every read request needs decoding
Suitable workloads
Suitable workloads

• **Immutable** data
  - As mutating objects require updating parities as well
  - Common model in object stores and cluster file systems
Suitable workloads

• **Immutable** data
  - As mutating objects require updating parities as well
  - Common model in object stores and cluster file systems

• **Not too small** objects (current implementation for $> 1\text{MB}$)
  - Due to overhead of connecting/reading from multiple servers
  - Operating regime: number of requests not the bottleneck
  - Facebook data-analytics cluster trace: Smaller than 1MB reads are $< 7\%$
Evaluation set-up
Evaluation set-up

- Amazon EC2
- 25 backend servers and 30 client servers
Evaluation set-up

- Amazon EC2
- **25 backend** servers and **30 client** servers
- Object popularity: *Zipf distribution* with high skew
Evaluation set-up

- Amazon EC2
- **25 backend** servers and **30 client** servers
- Object popularity: *Zipf distribution* with high skew
- **15% memory overhead** allowed for both SR and EC-Cache
Evaluation set-up

- Amazon EC2
- **25 backend** servers and **30 client** servers
- Object popularity: **Zipf distribution** with high skew
- **15% memory overhead** allowed for both SR and EC-Cache
- EC-Cache uses $k = 10$, $\Delta = 1$
  - BW overhead = 10%
Evaluation set-up

- Amazon EC2
- **25 backend** servers and **30 client** servers
- Object popularity: *Zipf distribution* with high skew
- **15% memory overhead** allowed for both SR and EC-Cache
- EC-Cache uses \( k = 10, \Delta = 1 \)
  - BW overhead = 10%
- Object size:
  - Fixed size of 40MB
  - Varying object sizes
Load balancing

Selective Replication

EC-Cache
Load balancing

- Percent imbalance metric:

\[ \lambda = \left( \frac{L_{\text{max}} - L_{\text{avg}^*}}{L_{\text{avg}^*}} \right) \times 100 \]
Load balancing

Selective Replication

- Percent imbalance metric:

\[ \lambda_{SR} = 43.45\% \]

EC-Cache

\[ \lambda = \left( \frac{L_{\text{max}} - L_{\text{avg}}^*}{L_{\text{avg}}^*} \right) \times 100 \]

\[ \lambda_{EC} = 13.14\% \]
Read latency

![Chart showing read latency comparison between Selective Replication and EC-Cache]

- **Mean**: 242 ms (Selective Replication), 96 ms (EC-Cache)
- **Median**: 238 ms (Selective Replication), 90 ms (EC-Cache)
- **95th Percentile**: 283 ms (Selective Replication), 134 ms (EC-Cache)
- **99th Percentile**: 340 ms (Selective Replication), 193 ms (EC-Cache)
- **99.9th Percentile**: 881 ms (Selective Replication), 492 ms (EC-Cache)
Read latency

- Median: 2.64x improvement
- 99th and 99.9th: ~1.75x improvement
Varying object sizes
Varying object sizes

Median latency

- Improvement increase from 1.33x for 1 MB to 5.5x for 100MB
Varying object sizes

Median latency

- Improvement increase from 1.33x for 1 MB to 5.5x for 100MB

Tail latency

- Improvement increases from 1.25x at 10 MB to 3.85x for 100 MB
Varying object sizes

Median latency

- Improvement increase from 1.33x for 1 MB to 5.5x for 100MB

Tail latency

- Improvement increases from 1.25x at 10 MB to 3.85x for 100 MB

Greater improvement in read latency for larger object sizes
Role of additional reads ($\Delta$)
Role of additional reads ($\Delta$)

Significant degradation in tail latency without additional reads ($\Delta = 0$)
Role of additional reads ($\Delta$)

- Selective replication with object splitting not sufficient
  - $2x$ memory and $2x$ BW overhead needed for additional reads

Significant degradation in tail latency without additional reads ($\Delta = 0$)
Additional evaluations in the paper

• With background network imbalance
• With server failures
• Sensitivity analysis for all parameters
• Write performance
Summary

• EC-Cache
  - Erasure coding highly effective for load balancing and reducing read latency in cluster caches
  - New application and new goals: Erasure coding previously used in disk-based storage systems primarily for fault-tolerance

• Implementation on Alluxio

• Evaluation
  - Median latency: > 5x improvement
  - Tail latency: > 3x improvement
  - Load balancing: > 3x improvement
END - rest are backup (rough) slides
Design considerations

Coding within vs. across objects

Storage systems

• Some systems code across (e.g., HDFS) and some within (e.g., Ceph)
  - Typically served from data units
Design considerations

### Coding within vs. across objects

<table>
<thead>
<tr>
<th>Storage systems</th>
<th>EC-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some systems code across (e.g., HDFS) and some within (e.g., Ceph)</td>
<td>Coding across objects not suitable</td>
</tr>
<tr>
<td>- Typically served from data units</td>
<td>- Too high BW overhead for serving object using parities</td>
</tr>
</tbody>
</table>
Design considerations

Choice of erasure code

†Rashmi et al. SIGCOMM 2014, Sathiamoorthy et al. VLDB 2013, Huang et al. ATC 2012
Design considerations

Choice of erasure code

Storage systems

- Reconstruction operations are frequent
- Systems employ codes that optimize resource usage during reconstruction†
  - some do not have “any k out of n” property

†Rashmi et al. SIGCOMM 2014, Sathiamoorthy et al. VLDB 2013, Huang et al. ATC 2012
## Design considerations

### Choice of erasure code

<table>
<thead>
<tr>
<th>Storage systems</th>
<th>EC-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>- <strong>Reconstruction</strong> operations are frequent</td>
<td>- No reconstruction operations</td>
</tr>
<tr>
<td>- Systems employ codes that optimize resource usage during reconstruction†</td>
<td>- Data persisted in underlying storage system</td>
</tr>
<tr>
<td>- some do not have “any k out of n” property</td>
<td>- “Any k out of n” property provides maximum flexibility in serving objects</td>
</tr>
</tbody>
</table>

†Rashmi et al. SIGCOMM 2014, Sathiamoorthy et al. VLDB 2013, Huang et al. ATC 2012
Design considerations
Design considerations

Goals

Storage systems

• Fault tolerance
Design considerations

Goals

Storage systems

• Fault tolerance

EC-Cache

• Latency
• Load balancing
Design Goals
Design Goals

Storage systems

• Fault tolerance
Design Goals

Storage systems

• Fault tolerance

EC-Cache

• Latency
• Load balancing
data units

parity units

\[
\begin{align*}
&d_1 & & d_2 & & d_3 & & d_4 & & d_5 \\
&d_1 & & d_2 & & d_3 & & d_4 & & d_5 \\
&d_1 & & d_2 & & d_3 & & d_4 & & d_5
\end{align*}
\]
Quick primer on erasure coding

• Takes in $k$ data units and creates $r$ “parity” units
Quick primer on erasure coding

- Takes in \( k \) data units and creates \( r \) “parity” units
- Any \( k \) of the \( (k+r) \) units are sufficient to decode the original \( k \) data units
Quick primer on erasure coding

- Takes in \( k \) data units and creates \( r \) “parity” units

- *Any* \( k \) of the \((k+r)\) units are sufficient to decode the original \( k \) data units

\[
\begin{array}{c}
data units \\
\hline
\text{d1} & \text{d2} & \text{d3} & \text{d4} & \text{d5} \\
\hline
\text{parity units} \\
\hline
\text{P1} & \text{P2} & \text{P3} & \text{P4} \\
\end{array}
\]

- \( k = 5 \)
- \( r = 4 \)
Quick primer on erasure coding

- Takes in $k$ data units and creates $r$ “parity” units

- *Any* $k$ of the $(k+r)$ units are sufficient to decode the original $k$ data units

```plaintext

<table>
<thead>
<tr>
<th>Data units</th>
<th>Parity units</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1 d2 d3 d4 d5</td>
<td>P1 P2 P3 P4</td>
</tr>
</tbody>
</table>

- $k = 5$
- $r = 4$

```
Quick primer on erasure coding

• Takes in $k$ data units and creates $r$ “parity” units

• Any $k$ of the $(k+r)$ units are sufficient to decode the original $k$ data units

- $k = 5$
- $r = 4$
Quick primer on erasure coding

- Takes in $k$ data units and creates $r$ “parity” units.

- *Any* $k$ of the $(k+r)$ units are sufficient to decode the original $k$ data units.

```
data units  parity units
d1  d2  d3  d4  d5  P1  P2  P3  P4
```

- $k = 5$
- $r = 4$
Quick primer on erasure coding

• Takes in $k$ data units and creates $r$ “parity” units

• Any $k$ of the $(k+r)$ units are sufficient to decode the original $k$ data units

\[ \text{data units} \quad \begin{array}{cccc} d1 & d2 & d3 & d4 & d5 \\ \end{array} \quad \begin{array}{cccc} P1 & P2 & P3 & P4 \\ \end{array} \quad \text{parity units} \]

Read

Decode

\[ \begin{array}{cccc} d1 & d2 & d3 & d4 & d5 \\ \end{array} \]
Erasure coding: Why and How?
1. **Finer control over memory overhead**
   - Selective replication allows *integer* overheads
   - Erasure coding allows *fractional*
   - E.g., $k = 10$ allows increments of 0.1
Erasure coding: Why and How?

1. Finer control over memory overhead
   - Selective replication allows integer overheads
   - Erasure coding allows fractional
   - E.g., $k = 10$ allows increments of 0.1

2. Load spreading and read parallelism through object splitting
   - Helps load balancing and reducing median latency
   - Hurts tail latency (without additional reads, i.e., $\Delta=0$)
Erasure coding: Why and How?

1. Finer control over memory overhead
   - Selective replication allows integer overheads
   - Erasure coding allows fractional
   - E.g., $k = 10$ allows increments of 0.1

2. Load spreading and read parallelism through object splitting
   - Helps load balancing and reducing median latency
   - Hurts tail latency (without additional reads, i.e., $\Delta=0$)

3. “Any-k-out-of-n” property for tail latency
   - Read from $(k + \Delta)$ and use the first $k$ that arrive
   - Helps reduce tail latency
Suitable workloads

- **Not too small** objects
  - **overhead** of reading from multiple servers
  - ensure number of requests is not the bottleneck
  - current implementation for objects **larger than 1MB**
Suitable workloads

• **Immutable** data
  - updating data requires updating parities as well
  - common model in object stores and cluster FS

• **Not too small** objects
  - *overhead* of reading from multiple servers
  - ensure number of requests is not the bottleneck
  - current implementation for objects *larger than 1MB*