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/ssd
  - Example: Alluxio (formerly Tachyon†)

†Li et al. SOCC 2014
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)

†Ananthanarayanan et al. NSDI 2012
Imbalances prevalent in clusters

Sources of imbalance:

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

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/unavailability

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 effects:
  - load imbalance
  - high read latency
Imbalances prevalent in cluster

Sources of imbalance:

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

→ Adverse effects:
  - load imbalance
  - high read latency

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

- Uses some memory overhead to cache replicas of objects based on their popularity
  - more replicas for more popular objects
Popular approach: Selective Replication

- Uses some memory overhead to cache replicas of objects based on their popularity
  - more replicas for more popular objects

Diagram:

- Server 1 with 2x GET A
- Server 2 with 1x GET B
- Server 3 with ellipses
Popular approach: Selective Replication

- Uses some memory overhead to cache replicas of objects based on their popularity
  - more replicas for more popular objects

![Diagram showing three servers with replicas of objects A and B.](image)
Popular approach: Selective Replication

• Uses some memory overhead to cache replicas of objects based on their popularity
  - more replicas for more popular objects

  ![Diagram](image)

• 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

Memory Overhead
Read performance & Load balance

Single copy in memory
Memory Overhead

Read performance & Load balance

Single copy in memory

Selective replication

Memory Overhead
Read performance & Load balance

Single copy in memory

Memory Overhead

Selective replication

EC-Cache
Read performance & Load balance

Memory Overhead

“Erasure Coding”

EC-Cache

Selective replication

Single copy in memory
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

\[ \begin{array}{c}
d1 & d2 & d3 & d4 & d5 & p1 & p2 & p3 & p4 \\
\end{array} \]

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

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

```
\begin{array}{c}
\text{data units} \\
\text{parity units} \\
\hline
\text{d1} & \text{d2} & \text{d3} & \text{d4} & \text{d5} \\
\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

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

$\begin{array}{c}
d1 \\
d2 \\
d3 \\
d4 \\
d5 \\
p1 \\
p2 \\
p3 \\
p4 \\
\end{array}$

Read

• $k = 5$
• $r = 4$
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

• $k = 5$
• $r = 4$
EC-Cache bird’s eye view: Writes
EC-Cache bird’s eye view: Writes

Put

Caching servers
EC-Cache bird’s eye view: Writes

- Object split into k data units

\[ X \]

\[ \text{Split} \]

\[ k = 2 \]

\[ \text{Put} \]

\[ \text{Caching servers} \]
EC-Cache bird’s eye view: Writes

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

```
   Caching servers

   ... 

   X

   Split
   d1  d2

   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 **uniformly** 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 uniformly 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 uniformly at random
  - “Additional reads”
- Use the first \(k\) units that arrive

Caching servers

\[
\begin{array}{c}
d1 \\
d2 \\
p1 \\
\vdots
\end{array}
\]

\[k = 2\]
\[r = 1\]

Get X
EC-Cache bird’s eye view: Reads

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

Caching servers

Get X
EC-Cache bird’s eye view: Reads

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

Caching servers

- **k = 2**
- **r = 1**
- **\(\Delta = 1\)**
- **k + \(\Delta\) = 3**

Get X
EC-Cache bird’s eye view: Reads

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

Caching servers

\(k = 2\)
\(r = 1\)
\(\Delta = 1\)
\(k + \Delta = 3\)

Get X
EC-Cache bird’s eye view: Reads

- Read from \((k + \Delta)\) units of the object chosen uniformly at random
  - “Additional reads”
- Use the first \(k\) units that arrive
- Decode the data units
- Combine the decoded units
Erasure coding: How does it help?
1. **Finer control over memory overhead**
   - Selective replication allows only **integer** control
   - Erasure coding allows **fractional** control
   - E.g., \( k = 10 \) allows control in multiples of 0.1
Erasure coding: How does it help?

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

2. Object splitting helps in load balancing
   - Smaller granularity reads help to smoothly spread load
   - Analysis on a certain simplified model:
     \[
     \frac{\text{Var}(L_{\text{EC-Cache}})}{\text{Var}(L_{\text{Selective Replication}})} = \frac{1}{k}
     \]
Erasure coding: How does it help?

3. Object splitting reduces median latency but hurts tail latency
   - Read parallelism helps reduce median latency
   - Straggler effect hurts tail latency (if no additional reads)
Erasure coding: How does it help?

3. Object splitting reduces median latency but hurts tail latency
   - Read parallelism helps reduce median latency
   - Straggler effect hurts tail latency (if no additional reads)

4. “Any k out of (k+r)” property helps to reduce tail latency
   - Read from (k + Δ) and use the first k that arrive
   - Δ = 1 often 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>• Space-efficient fault tolerance</td>
<td>• Reduce read 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$^\dagger$</td>
<td></td>
</tr>
<tr>
<td>• Some codes do not have “any k out of (k+r)” property</td>
<td></td>
</tr>
</tbody>
</table>

$^\dagger$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 (k+r)” property</td>
<td>• “Any k out of (k+r)” 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. How do we use erasure coding: 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>
3. How do we use erasure coding: 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>
<tr>
<td>• Does not affect fault tolerance</td>
<td></td>
</tr>
</tbody>
</table>
### Design considerations

3. **How do we use erasure coding: 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>• Need to encode within objects</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: Very 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** (formerly Tachyon)
  - **Backend caching servers**: cache data — unaware of erasure coding
  - **EC-Cache client library**: all read/write logic handled
Implementation

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

- Reed-Solomon code
  - Any k out of (k+r) property
Implementation

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

- Reed-Solomon code
  - Any k out of (k+r) property

- Intel ISA-L hardware acceleration library
  - Fast encoding and decoding
Evaluation set-up
Evaluation set-up

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

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

• Amazon EC2
• 25 backend caching servers and 30 client servers
• Object popularity: Zipf distribution with high skew
• EC-Cache uses $k = 10, \Delta = 1$
  - BW overhead = 10%
Evaluation set-up

- Amazon EC2
- **25 backend caching** servers and **30 client** servers
- Object popularity: Zipf distribution with high skew
- EC-Cache uses $k = 10, \Delta = 1$
  - BW overhead = 10%
- Varying object sizes
Load balancing

Selective Replication

EC-Cache
Load balancing

Selective Replication

- Percent imbalance metric:

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

EC-Cache
Load balancing

Selecting Replication

- Percent imbalance metric:

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

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

> 3x reduction in load imbalance metric
Read latency

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

- **Mean**: 242, 96 for Selective Replication, 238, 90 for EC-Cache
- **Median**: 283, 134 for Selective Replication, 283, 134 for EC-Cache
- **95th Percentile**: 340, 193 for Selective Replication, 340, 193 for EC-Cache
- **99th Percentile**: 481, 242 for Selective Replication, 481, 242 for EC-Cache
- **99.9th Percentile**: 81, 492 for Selective Replication, 81, 492 for EC-Cache
Read latency

- Median: $2.64x$ improvement
- 99th and 99.9th: $\sim 1.75x$ improvement
Varying object sizes

Median latency

Tail latency

5.5x improvement for 100MB

3.85x improvement for 100 MB

More improvement for larger object sizes
Role of additional reads ($\Delta$)
Role of additional reads ($\Delta$)

Significant degradation in tail latency without additional reads (i.e., $\Delta = 0$)

![Graph showing CDF of read latency for different replication strategies.](image)

- EC-Cache, $\Delta=0$
- EC-Cache, $\Delta=1$
- Selective Replication
Additional evaluations in the paper

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

- EC-Cache
  - Cluster cache employing erasure coding for load balancing and reducing read latencies
  - Demonstrates new application and new goals for which erasure coding is highly effective
Summary

• EC-Cache
  - Cluster cache employing erasure coding for load balancing and reducing read latencies
  - Demonstrates new application and new goals for which erasure coding is highly effective

• Implementation on Alluxio

• Evaluation
  - Load balancing: >3x improvement
  - Median latency: >5x improvement
  - Tail latency: >3x improvement
Summary

• EC-Cache
  - Cluster cache employing erasure coding for load balancing and reducing read latencies
  - Demonstrates new application and new goals for which erasure coding is highly effective

• Implementation on Alluxio

• Evaluation
  - Load balancing: > 3x improvement
  - Median latency: > 5x improvement
  - Tail latency: > 3x improvement

Thanks!