Knockoff: Cheap versions in the cloud

Xianzheng Dou, Peter M. Chen, Jason Flinn
Cloud-based storage

Pros:
- Ease-of-management
- Reliability
Cloud-based storage

Challenges:
Storage costs
Communication costs

Google Drive
Dropbox
Microsoft OneDrive
Versioning increases costs

Pros:
- Recovery of lost data
- Auditing
- Troubleshooting

Google Drive
Dropbox
Microsoft OneDrive
Reducing costs: a new direction

• Established methods exploit similarities in data
  – Chunk-based deduplication
  – Delta compression
  – Greater work for incremental gains

• Our goal: explore an orthogonal new dimension
  – Deterministically recompute data in lieu of communication, storage
File: data or computation?

Computation

File data
File: data or computation?
File: data or computation?

Computation → File data
File: data or computation?

Computation → File data
File: data or computation?
File: data or computation?

Computation  File data
File: data or computation?

How can we address non-determinism?
File: data or computation?

• Deterministic record and replay

Logs of nondeterminism
File: data or computation?

- Deterministic record and replay

Record

Logs of nondeterminism

Xianzheng Dou
File: data or computation?

- Deterministic record and replay

Record

Logs of nondeterminism
File: data or computation?

- Deterministic record and replay

![Diagram showing record and replay processes]

- Logs of nondeterminism

Xianzheng Dou
Knockoff

• Selectively substitutes computation for data
• Benefits
  – Reduction compared to chunk-based deduplication
    • Communication costs: 21%
    • Storage costs: 19%
  – Benefits increases as we retain versions more frequently
  – A new fined-grained versioning policy
Outline

• Introduction
• Writing files
• Storing files
• Evaluation
Knockoff

- Knockoff selectively represents a file as:
  - Normal file data (by value)
  - Logs of the nondeterminism needed to recompute the file (by operation)
Knockoff

- Knockoff selectively represents a file as:
  - Normal file data (by value)
  - Logs of the nondeterminism needed to recompute the file (by operation)
Knockoff

• Knockoff selectively represents a file as:
  - Normal file data (by value)
  - Logs of the nondeterminism needed to recompute the file (by operation)
Knockoff

- Knockoff selectively represents a file as:
  - Normal file data (by value)
  - Logs of the nondeterminism needed to recompute the file (by operation)
An example log for compilation

<table>
<thead>
<tr>
<th>Log entry</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 open</td>
<td>rc=3</td>
</tr>
<tr>
<td>2 mmap</td>
<td>rc=&lt;addr&gt;, file=&lt;id, version&gt;</td>
</tr>
<tr>
<td>3 gettimeofday</td>
<td>rc=0, time=&lt;time&gt;</td>
</tr>
<tr>
<td>4 pthread_lock</td>
<td>rc=0</td>
</tr>
<tr>
<td>5 SIGCHILD</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
An example log for compilation

<table>
<thead>
<tr>
<th>Log entry</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 open</td>
<td>rc=3</td>
</tr>
<tr>
<td>2 mmap</td>
<td>rc=&lt;addr&gt;, file=&lt;id, version&gt;</td>
</tr>
<tr>
<td>3 gettimeofday</td>
<td>rc=0, time=&lt;time&gt;</td>
</tr>
<tr>
<td>4 pthread_lock</td>
<td>rc=0</td>
</tr>
<tr>
<td>5 SIGCHILD</td>
<td></td>
</tr>
</tbody>
</table>

Return values from syscalls
Ordering of thread synchronization
Signals
An example log for compilation

<table>
<thead>
<tr>
<th>Log entry</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 \textit{open}</td>
<td>rc=3</td>
</tr>
<tr>
<td>2 \textit{mmap}</td>
<td>rc=&lt;addr&gt;, file=&lt;id, version&gt;</td>
</tr>
<tr>
<td>3 \textit{gettimeofday}</td>
<td>rc=0, time=&lt;time&gt;</td>
</tr>
<tr>
<td>4 \textit{pthread_lock}</td>
<td>rc=0</td>
</tr>
<tr>
<td>5 \textit{SIGCHILD}</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### An example log for compilation

<table>
<thead>
<tr>
<th>Log entry</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 open</td>
<td>rc=3</td>
</tr>
<tr>
<td>2 mmap</td>
<td>rc=&lt;addr&gt;, file=&lt;id,version&gt;</td>
</tr>
<tr>
<td>3 gettimeofday</td>
<td>rc=0, time=&lt;time&gt;</td>
</tr>
<tr>
<td>4 pthread_lock</td>
<td>rc=0</td>
</tr>
<tr>
<td>5 SIGCHILD</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

---

Xianzheng Dou
Writing files

By value

By operation

Xianzheng Dou
Writing files

By value

By operation

Xianzheng Dou
Writing files

By value

By operation

Xianzheng Dou
Writing files

By value

By operation

photo editing

Xianzheng Dou
Writing files

By value

cryptographic key generation

By operation
• Introduction
• Writing files
• Storing files
• Evaluation
Storing files

• Store files by value or by operation?

• A tradeoff between latency and costs
  – Current versions: by value
  – Past versions: by value or by operation
Storing past versions

• Maximum materialization delay
  – Time bound for reconstructing any version

Regeneration time = \(20\text{s}\)  <  Materialization delay = \(60\text{s}\)
Storing past versions

- Maximum materialization delay
  - Time bound for reconstructing any version

Regeneration time = 20s  <  Materialization delay = 60s
Storing past versions

- Maximum materialization delay
  - Time bound for reconstructing any version

Regeneration time = \(100\text{s}\)  \(>\) Materialization delay = \(60\text{s}\)
Storing past versions

- Maximum materialization delay
  - Time bound for reconstructing any version

Regeneration time = 100s > Materialization delay = 60s
Storing past versions

• Maximum materialization delay
  – Time bound for reconstructing any version
• Longest path > materialization delay

Total regeneration time = 20s

<

Materialization delay = 60s
Storing past versions

- Maximum materialization delay
  - Time bound for reconstructing any version
- Longest path > materialization delay

Total regeneration time = 50s
<
Materialization delay = 60s
Storing past versions

- Maximum materialization delay
  - Time bound for reconstructing any version
- Longest path > materialization delay

Total regeneration time = \( 80s \)

Materialization delay = \( 60s \)
Storing past versions

- Maximum materialization delay
  - Time bound for reconstructing any version
- Longest path > materialization delay

Total regeneration time = 80s

Materialization delay = 60s
Storing past versions

- Maximum materialization delay
  - Time bound for reconstructing any version
- Longest path > materialization delay

Total regeneration time = 80s

Materialization delay = 60s
Storing past versions

- Maximum materialization delay
  - Time bound for reconstructing any version
- Longest path > materialization delay
  - A greedy algorithm

Materialization delay = 60s
Storing past versions: versioning policies

• Frequency of versioning
Storing past versions: versioning policies

• Frequency of versioning

- No versioning
- Version on close
- Version on write
- Eidetic versioning
Storing past versions: versioning policies

- Frequency of versioning

  - No versioning
  - Version on close
  - Version on write
  - Eidetic versioning

Any past transient state in memory? Memory-mapped files
Optimization: log compression

- Chunk-based deduplication is effective for file data
  - Executions of the same application have similar patterns
  - Can it also be applied to computation (logs of nondeterminism)?

- Delta compression
Optimization: log compression

- Problem: a smattering of values differ in each log

<table>
<thead>
<tr>
<th>Log entry</th>
<th>Values</th>
<th>Log entry</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 open</td>
<td>rc=3</td>
<td>1 open</td>
<td>rc=3</td>
</tr>
<tr>
<td>2 pthread_lock</td>
<td>rc=0</td>
<td>2 pthread_lock</td>
<td>rc=0</td>
</tr>
<tr>
<td>3 mmap</td>
<td>rc=&lt;addr&gt;, file=&lt;id,version&gt;</td>
<td>3 mmap</td>
<td>rc=&lt;addr&gt;, file=&lt;id,version&gt;</td>
</tr>
<tr>
<td>4 read</td>
<td>rc=&lt;size&gt;, file=&lt;id,version&gt;</td>
<td>4 read</td>
<td>rc=&lt;size&gt;, file=&lt;id,version&gt;</td>
</tr>
<tr>
<td>5 gettimeofday</td>
<td>rc=0, time=9.90</td>
<td>5 gettimeofday</td>
<td>rc=0, time=1.10</td>
</tr>
<tr>
<td>6 write</td>
<td>rc=&lt;size&gt;</td>
<td>6 write</td>
<td>rc=&lt;size&gt;</td>
</tr>
<tr>
<td>7 pthread_unlock</td>
<td>rc=0</td>
<td>7 pthread_unlock</td>
<td>rc=0</td>
</tr>
</tbody>
</table>

...
Optimization: log compression

- Problem: a smattering of values differ in each log

<table>
<thead>
<tr>
<th>Log entry</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>rc=3</td>
</tr>
<tr>
<td>pthread_lock</td>
<td>rc=0</td>
</tr>
<tr>
<td>pthread_unlock</td>
<td>rc=0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Delta compression: 42% reduction
Outline

• Introduction
• Writing files
• Storing files
• Evaluation
Evaluation

- How much does Knockoff reduce bandwidth usage?
- How much does Knockoff reduce storage costs?
- What is Knockoff’s performance overhead?

- For more experimental results, please refer to our paper
Experimental setup

• User study
  – 8 participants performed several simple tasks in one hour
• 20-day study
  – A single-user longitudinal study
• A variety of programs used
  – Various Linux utilities, text editors and programming languages
Bandwidth usage: user study
Bandwidth usage: user study

Already achieve 80%-85% reduction
Bandwidth usage: user study

Already achieve 80%-85% reduction
Bandwidth usage: user study

Data sent to the server (MB)

- No versioning
- Version on close
- Version on write
- Eidetic

- Chunk-based deduplication
- Knockoff

- 24%
Bandwidth usage: user study

Data sent to the server (MB)

- No versioning
- Version on close
- Version on write
- Eidetic

- Chunk-based deduplication
- Knockoff

- 24%
- 25%
- 47%
Bandwidth usage: user study

Data sent to the server (MB)

- **No versioning**: 24%
- **Version on close**: 25%
- **Version on write**: 47%
- **Eidetic**: 47%

**Legend**:
- Blue: Chunk-based deduplication
- Yellow: Knockoff

Xianzheng Dou
Variances across applications

Bandwidth savings (%)

<table>
<thead>
<tr>
<th>Application</th>
<th>Savings (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux utility</td>
<td>50</td>
</tr>
<tr>
<td>Graph editing</td>
<td>25</td>
</tr>
<tr>
<td>Libreoffice</td>
<td>20</td>
</tr>
<tr>
<td>Programming</td>
<td>20</td>
</tr>
<tr>
<td>Text editing</td>
<td>0</td>
</tr>
<tr>
<td>Web browsing</td>
<td>15</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>20</strong></td>
</tr>
</tbody>
</table>

Version on close

Xianzheng Dou
Variances across applications

Version on close

<table>
<thead>
<tr>
<th>Application</th>
<th>Bandwidth savings (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux utility</td>
<td>50</td>
</tr>
<tr>
<td>Graph editing</td>
<td>20</td>
</tr>
<tr>
<td>Libreoffice</td>
<td>10</td>
</tr>
<tr>
<td>Programming</td>
<td>20</td>
</tr>
<tr>
<td>Text editing</td>
<td>0</td>
</tr>
<tr>
<td>Web browsing</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>90</td>
</tr>
</tbody>
</table>
Variances across users

Bandwidth savings(%) vs. Version on close

- A
- B
- C
- D
- E
- F
- G

Knockoff
Variances across users

Bandwidth savings (%)

Version on close

Xianzheng Dou
Relative storage costs for past versions

- **Version on close**
  - Chunk-based deduplication
  - Knockoff

- **Version on write**
  - Chunk-based deduplication
  - Knockoff

- **Eidetic**
  - Knockoff
Relative storage costs for past versions

- **Version on close**: 19% (Chunk-based deduplication), 23% (Knockoff)
- **Version on write**: 23% (Knockoff)
- **Eidetic**: 23% (Knockoff)

Xianzheng Dou
Relative storage costs for past versions

- Version on close: 19%
- Version on write: 23%
- Eidetic: 23%

Xianzheng Dou
Performance overheads

- **7-8% performance overheads**
Conclusion

• A new dimension for reducing costs
• Selectively substitute computation for data
• A general-purpose system for deterministic recomputation
  – Reduces storage and communication costs for existing versioning policies
  – Enables eidetic versioning
Thank you!
Varying the materialization delay

![Graph showing the relationship between materialization delay and relative storage cost for different versions.]
Table 2: Relative cost savings from using Knockoff for different versioning policies. We show costs for a typical 4G cellular network, an expensive current ISP, a cheap current ISP, and a hypothetical ISP that is an order of magnitude cheaper than the cheap current ISP.
## Workload characteristics

<table>
<thead>
<tr>
<th></th>
<th>20-day study</th>
<th>User study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk read (MB)</td>
<td>5473</td>
<td>2583</td>
</tr>
<tr>
<td>Disk write (MB)</td>
<td>6706</td>
<td>4339</td>
</tr>
<tr>
<td>File open count</td>
<td>261523</td>
<td>418594</td>
</tr>
<tr>
<td>Number of executions</td>
<td>3803</td>
<td>1146</td>
</tr>
<tr>
<td>Number of programs</td>
<td>75</td>
<td>63</td>
</tr>
</tbody>
</table>

Table 1: Workload characteristics