HPDedup: A Hybrid Prioritized Data Deduplication Mechanism for Primary Storage in the Cloud

Huijun Wu¹,⁴, Chen Wang², Yinjin Fu³, Sherif Sakr¹, Liming Zhu¹,² and Kai Lu⁴

The University of New South Wales¹

Data61, CSIRO²

PLA University of Science and Technology³

National University of Defence Technology⁴
Outline

● Background
● Motivations
● Hybrid Prioritized Deduplication
● Experiment Results
● Conclusion
Background

- **Primary Storage Deduplication**
  - Save the storage capacity
  - Improve the I/O efficiency

- **The state-of-the-art**
  - Post-processing deduplication
    - Perform during off-peak time
  - Inline deduplication
    - Perform on the write path

18/05/2017
Post-processing Deduplication

- The commodity product uses post-processing deduplication [TOS’16]
  - Windows Server 2012 [ATC’12]

- Challenges remain for real-world systems
  - Off-peak periods may not be enough
  - More storage capacity is required
  - Duplicate writes shorten the lifespan of storage devices (e.g., SSD)
  - Does not help improving the I/O performance, but wastes I/O bandwidth

- Inline deduplication can help
Inline Deduplication

- Fingerprint look-up is the bottleneck
  - On-disk fingerprint table introduces high latency
  - Fingerprint table is large and hard to fit in memory
  - Cache efficiency is critical

- The state-of-the-art solutions and challenges
  - Exploit the temporal locality of workloads [FAST’12][IPDPS’14]
    - But temporal locality may not exist [TPDS’17]
  - For cloud scenario,
    - Locality for workloads of different VMs may be quite different
  - Workloads may interfere with each other and reduce the cache efficiency
Outline

- Background
- **Motivations**
- Hybrid Prioritized Deduplication
- Experiment Results
- Conclusion
Motivation

- Workloads with different temporal locality interfere with each other
  - A toy example.

![Diagram showing workloads A, B, and C with a fingerprint cache and # of Deduplicated Blocks: 0]

18/05/2017
**Motivation**

- Workloads with different temporal locality interfere with each other
  - A toy example.

---

### A toy example.

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>3</td>
<td>4</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# of Deduplicated Blocks: 1
Motivation

- Workloads with different temporal locality interfere with each other
  - A toy example.

# of Deduplicated Blocks: 2
Motivation

- Workloads with different temporal locality interfere with each other
  - A toy example.

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>3</td>
<td>4</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# of Deduplicated Blocks: 4
Motivation

- Workloads with different temporal locality interfere with each other
  - A toy example.

```
Time 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
A  1 2 3 4 5 6 7 8 9 10 11 12 13 3 4 14 15 16 17
B  1 2 3 1 2 3 1 1 4 5 6 6 6 7 8 9 10 7
C  1 1 1 2 3 3 4 4 5 5 6 6 7 8 8 9

# of Deduplicated Blocks: 5
```
Motivation

- Workloads with different temporal locality interfere with each other
  - A toy example.
    - 18 duplicate blocks in total, only 6 are identified.

# of Deduplicated Blocks: 6
Motivation

- Temporal locality may be weak for workloads
  - Histogram for the distribution of distance between duplicate blocks
Motivation

- Workloads with different temporal locality interfere with each other
  - Using real-world I/O trace. (LRU)

<table>
<thead>
<tr>
<th>Trace</th>
<th>Request number</th>
<th>Write request ratio</th>
<th>Duplicate writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud-FTP</td>
<td>2293424</td>
<td>84.15%</td>
<td>387140</td>
</tr>
<tr>
<td>FIU-mail</td>
<td>1961588</td>
<td>98.58%</td>
<td>1633424</td>
</tr>
<tr>
<td>FIU-web</td>
<td>116940</td>
<td>49.36%</td>
<td>30534</td>
</tr>
<tr>
<td>FIU-home</td>
<td>293605</td>
<td>91.03%</td>
<td>32688</td>
</tr>
</tbody>
</table>

# of duplicate blocks: FIU-mail > 4*Cloud-FTP
Occupied cache size: FIU-mail < 0.8*Cloud-FTP

Cache resource allocation is unreasonable!
Outline

- Background
- Motivations
- Hybrid Prioritized Deduplication
- Experiment Results
- Conclusion
Hybrid Prioritized Deduplication

- **Hybrid** inline & post-processing deduplication
  - Either post-processing or inline deduplication works well
  - Solution: Combine inline and post-processing deduplication together
  - Identifying more duplicates by inline caching
  - Using post-processing to achieve exact deduplication

- Challenges: Interference compromises the temporal locality of workload, thus reducing the efficiency of fingerprint caching

- We differentiate workloads (data streams) to improve it
Hybrid Prioritized Deduplication

- **Prioritize** the cache allocation for inline deduplication
  - Data stream that contributes more deduplication ratio should get more cache resources
  - For inline phase, deduplication ratio comes from better temporal locality

- How to evaluate temporal locality?
  - Changes dynamically with time
  - Accurate estimation is critical to achieve good cache allocation
  - Use # of duplicate blocks in $N$ consecutive data blocks (*estimation interval*) as an indicator for temporal locality
System architecture

Estimate the temporal locality for streams and allocate cache according to this.

On-disk fingerprint table for post-processing deduplication.
Evaluate the temporal locality

- Simple idea: Count distinct data block fingerprints for streams
  - Introduce high memory overhead
  - May be comparable to the cache capacity

- Estimate rather than count
  - Get the number of distinct fingerprints by small portion of samples
  - Essentially same as a classical problem ‘How many distinct elements exist in a set ?’ Origin – Estimate # of species of animal population from samples [Fisher, JSTOR’1940]
  - Sublinear estimator – Unseen Estimation Algorithm [NIPS’13]
Estimate the temporal locality

- Using unseen algorithm to estimate LDSS.

![Diagram](Diagram.png)
Key points to deploy the estimation

- Unseen algorithm requires uniform sampling
  - Each fingerprint should be sampled with the same probability
  - We use Reservoir Sampling [TOMS’04]

- Choose a proper estimation interval
  - More unique data blocks -> Larger interval
  - A good approximation
    - Historical inline deduplication ratio
  - Adaptive method
Differentiate the spatial locality

- Existing deduplication solutions exploit the spatial locality to reduce disk fragmentation
  - perform deduplication on block sequences longer than a fixed threshold.

- Workloads have different sensitivity to the increase of threshold
  - Differentiating the workloads achieves better deduplication ratio with less fragmentation

![Graph showing sensitivity to threshold change](image)
Outline

- Background
- Motivations
- Hybrid Prioritized Deduplication
- Experiment Results
- Conclusion
Evaluation Setup

- **Evaluated Systems**
  - compare with inline (iDedup), post-processing and hybrid (DIODE) deduplication schemas

- **Workloads**
  - FIU trace (FIU-home, FIU-web and FIU-mail)
  - Cloud-FTP (trace we collect from a FTP server by using NBD)

- **Mixing workloads as multiple VMs**
  - Different ratios between good locality (FIU, L) and bad locality (Cloud-FTP, NL) workloads
  - Workload A (L:NL = 3:1), workload B (L:NL = 1:1), workload C (L:NL = 1:3)
Evaluation

- Inline deduplication ratio
  - Cache size (20MB – 320MB)

  - HPDedup improves the inline deduplication ratio (8.04% - 37.75%)

18/05/2017
Evaluation

- Data written to disks (Comparing with post-processing deduplication)

  - HPDedup reduce the data written to disks by 12.78% - 45.08%
Evaluation

- Average # of hits for each cached fingerprint
  - DIODE [MASCOTS’16] skips files in inline deduplication based on file extensions
  - HPDedup classifies data at finer-grained (stream temporal locality level) so that the efficiency of inline deduplication can be further improved
Locality estimation allocates cache resources according to the temporal locality of streams and improves the inline deduplication ratio by 12.53%.
Evaluation

- Deduplication threshold
  - DIODE [MASCOTS’16] does not differentiate workloads.
  - HPDedup introduces less fragmentation while achieving higher dedup ratio
Overhead Analysis

- Computational Overhead
  - Mainly includes (1) generating the fingerprint histogram and (2) linear programming of estimation algorithm
  - (1) 7ms for 1M fingerprints
  - (2) 27ms regardless of the estimation interval size
  - More intuitive feeling – ms level overhead for GBs of data writing
  - Can be computed asynchronously

- Memory Overhead
  - Up to 2.81% of cache capacity for the three workloads
Outline

- Background
- Motivations
- Hybrid Prioritized Deduplication
- Experiment Results
- Conclusion
Conclusion

- New hybrid, prioritized data deduplication
  - Fusing inline and post-processing deduplication
  - Differentiate the temporal and spatial locality of data streams coming from VMs and applications

- More efficient compared with the state of the art
  - Improve the inline deduplication ratio by up to 37.75%
  - Reduce the disk capacity requirement by up to 45.08%
  - Low computational and memory overhead
HPDedup: A Hybrid Prioritized Data Deduplication Mechanism for Primary Storage in the Cloud

Huijun Wu
The University of New South Wales, Australia
Email: huijunw@cse.unsw.edu.au