LRC: Dependency-Aware Cache Management for Data Analytics Clusters

Yinghao Yu, Wei Wang, Jun Zhang, and Khaled B. Letaief

IEEE INFOCOM 2017
Outline

- Cache Management for Data Analytics Clusters
- Inefficiency of Existing Cache Polices
- LRC: Dependency-Aware Cache Management
- Evaluations
- Conclusions
Memory Caches in Data Analytics Clusters

- Caching input data speeds up tasks by orders of magnitude.

![Graph showing running time vs number of iterations]

- Cache space is limited.
- Efficient cache management is desired.

127 sec / iteration
6 sec / iteration

[M. Zaharia, 2012]
Cache Management: A Classic Problem

- Well studied in conventional systems: databases, web servers, etc.

- Objective: optimize the **cache hit ratio**
  - Maximize the chance of in-memory data access.
  - Cache the data that will likely be reused again.

- Optimal cache replacement
  - The MIN policy [L. A. Belady, 1966]
    - When the cache is full, evicts the data whose **next** usage is the **farthest** away from now.
  - Not implementable: MIN requires the exact sequence of **future** data access.
Existing Solutions

- **Least Recently Used (LRU) policy** [R. L. Mattson, 1970]
  - Evicts the data block that has not been used for the longest period.
  - Short-term popularity
  - Widely employed in prevalent systems, e.g., Spark, Tez and Alluxio.

- **Least Frequently Used (LFU) policy** [M. Stonebraker, 1971]
  - Evicts the data block that has been used the least times.
  - Long-term popularity

- Combining LRU and LFU: ARC, LRFC, K-LRC, etc.

- Summary: “guessing” the future data access patterns based on historical information (access recency or frequency).
What’s New for Data Analytics Clusters?

Question: In data analytics systems, is the future data access completely random and unpredictable?

No.
Data Access Pattern Revealed in the Application Semantics

- **Application Semantics**
  - Data dependency structured as a Directed Acyclic Graph (DAG)

Available to the cluster scheduler **before** the job starts
- Data access follows the dependency DAG.

- The future is not totally unpredictable.
Outline

- Cache Management for Data Analytics Clusters
- Inefficiency of Existing Cache Policies
- LRC: Dependency-Aware Cache Management
- Evaluations
- Conclusions
Inefficiency of Existing Cache Polices

- Existing cache polices (LRU / LFU) are oblivious to the readily available data dependency information.

- How bad can the result be?
  - Inactive data (no future access) cannot be evicted timely.
  - In our measurement studies, inactive data accounts for $>77\%$ of the cache space for $>50\%$ of time.
LRU (or LFU) is Unable to Evict Inactive Data Timely.

- Assume a 3-entry cache keeping blocks A, B and C initially.

To keep block D, one in-memory block should be evicted.
- Block B becomes inactive after block D is computed but will be retained in memory by LRU (or LFU).
Inefficiency of Existing Cache Policies

- Inactive data takes up a large portion of the cache space
- Memory footprint of 15 SparkBench applications [M. Li, 2015] in a 10-node EC2 cluster

<table>
<thead>
<tr>
<th>Application Type</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Learning</td>
<td>Logistic Regression</td>
</tr>
<tr>
<td></td>
<td>Support Vector Machine (SVM)</td>
</tr>
<tr>
<td></td>
<td>Matrix Factorization</td>
</tr>
<tr>
<td></td>
<td>Page Rank</td>
</tr>
<tr>
<td></td>
<td>SVD Plus Plus</td>
</tr>
<tr>
<td></td>
<td>Triangle Count</td>
</tr>
<tr>
<td></td>
<td>Hive</td>
</tr>
<tr>
<td></td>
<td>RDD Relation</td>
</tr>
<tr>
<td></td>
<td>Twitter Tag</td>
</tr>
<tr>
<td>SQL Queries</td>
<td>Page View</td>
</tr>
<tr>
<td>Streaming Workloads</td>
<td>Connected Component</td>
</tr>
<tr>
<td></td>
<td>Strongly Connected Component</td>
</tr>
<tr>
<td></td>
<td>Shortest Paths</td>
</tr>
<tr>
<td>Other Workloads</td>
<td>Label Propagation</td>
</tr>
<tr>
<td></td>
<td>Pregel Operation</td>
</tr>
</tbody>
</table>

Median: 77%
A New Cache Policy is Desired

- A new cache policy for data analytics clusters is highly desired.

- Challenge: How to take use of the data dependency information (DAGs) to clear the inactive data efficiently?
The MIN Policy Cannot be Implemented, Still

- The MIN policy requires exact future data reference sequence.
- Unavailable due to parallel processing

Which one of block A and C is accessed first?
Outline

- Cache Management for Data Analytics Clusters
- Inefficiency of Existing Cache Policies
- LRC: Dependency-Aware Cache Management
- Evaluations
- Conclusions
LRC: Dependency-Aware Cache Management

- **Reference count**: defined for each data block as the number of downstream tasks depending on it.
  - Dynamically changing over time:
  - Least Reference Count (LRC) policy: when the cache is full, always evict the data with the least reference count.
    - Inactive data (w/ zero reference count) is evicted first, e.g., block B.
    - Easy to implement
Intuition of LRC

- Reference count
  Number of downstream tasks accessing it
  Chance to be used in the near future
- Empirical study: reference count is a more accurate indicator to predict future data access than recency/frequency.

Caching the data blocks with the largest reference count is always better than caching the most recently (frequently) used ones.
Implementation in Spark

- **Architecture:** *Shaded boxes highlight our implementation*
Outline

- Cache Management for Data Analytics Clusters
- Inefficiency of Existing Cache Policies
- LRC: Dependency-Aware Cache Management
- Evaluations
- Conclusions
Evaluations

- Metrics
  - Cache hit ratio
  - Application runtime

- Cluster setup
  - 20 Amazon EC2 instances.
  - Instance type: m4.large. Dual-core 2.4 GHz Intel Xeon® E5-2676 v3 (Haswell) processors and 8 GB memory.

- Workloads.
  - Typical applications in SparkBench
Not all applications benefit from the improvement of cache management.

<table>
<thead>
<tr>
<th>Workload</th>
<th>Cache All</th>
<th>Cache None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Rank</td>
<td>56 s</td>
<td>552 s</td>
</tr>
<tr>
<td>Connected Component</td>
<td>34 s</td>
<td>72 s</td>
</tr>
<tr>
<td>Shortest Paths</td>
<td>36 s</td>
<td>78 s</td>
</tr>
<tr>
<td>K-Means</td>
<td>26 s</td>
<td>30 s</td>
</tr>
<tr>
<td>Pregel Operation</td>
<td>42 s</td>
<td>156 s</td>
</tr>
<tr>
<td>Strongly Connected Component</td>
<td>126 s</td>
<td>216 s</td>
</tr>
<tr>
<td>Label Propagation</td>
<td>34 s</td>
<td>37 s</td>
</tr>
<tr>
<td>SVD Plus Plus</td>
<td>55 s</td>
<td>120 s</td>
</tr>
<tr>
<td>Triangle Count</td>
<td>84 s</td>
<td>99 s</td>
</tr>
<tr>
<td>Support Vector Machine (SVM)</td>
<td>72 s</td>
<td>138 s</td>
</tr>
</tbody>
</table>
Evaluation Summary

➢ Summary: maximum saving of application runtime

<table>
<thead>
<tr>
<th>Workload</th>
<th>Cache Size</th>
<th>LRU</th>
<th>LRC</th>
<th>Speedup by LRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Rank</td>
<td>6.6 GB</td>
<td>169.3 s</td>
<td>68.4 s</td>
<td>59.58%</td>
</tr>
<tr>
<td>Pregel Operation</td>
<td>0.22 GB</td>
<td>121.9 s</td>
<td>66.3 s</td>
<td>45.64%</td>
</tr>
<tr>
<td>Connected Component</td>
<td>2.2 GB</td>
<td>50.6 s</td>
<td>27.6 s</td>
<td>45.47%</td>
</tr>
<tr>
<td>SVD Plus Plus</td>
<td>0.88 GB</td>
<td>254.3 s</td>
<td>177.6 s</td>
<td>30.17%</td>
</tr>
</tbody>
</table>
Cache Hit Ratio - 1

**PageRank**

- Hit Ratio vs Total Cache Size (GB)
  - 7.7 GB: 1.0
  - 6.6 GB: 0.9
  - 5.5 GB: 0.8
  - 4.4 GB: 0.7
  - 2.2 GB: 0.6

**PregelOperation**

- Hit Ratio vs Total Cache Size (GB)
  - 1.65 GB: 1.0
  - 1.1 GB: 0.9
  - 0.55 GB: 0.8
  - 0.22 GB: 0.7
  - 0.11 GB: 0.6

**Legend:**
- LRU
- LRC
Cache Hit Ratio - 2

- **ConnectedComponent**
  - Hit Ratio vs Total Cache Size (GB)
  - Comparison between LRU and LRC

- **SVDPlusPlus**
  - Hit Ratio vs Total Cache Size (GB)
  - Comparison between LRU and LRC
Application Runtime - 1

![Graphs showing runtime comparison between LRU and LRC for PageRank and PregelOperation across different total cache sizes.](image-url)
The advantage of LRC becomes more prominent when the cache size decreases.
Outline

- Cache Management for Data Analytics Clusters
- Inefficiency of Existing Cache Policies
- LRC: Dependency-Aware Cache Management
- Evaluations
- Conclusions
Conclusions

In this work, we have

- Investigated the data access pattern in data analytic systems with empirical data.
- Motivated the need to leverage the dependency DAGs to optimize the cache management.
- Designed and implemented LRC, a dependency-aware cache management policy.
  - Speed up typical workloads by up to 60%
  - LRC provides greater performance gain when the cache contention becomes more intense.
Thank you

Questions?