香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

SENG 工學院 SCHOOL OF ENGINEERING

LRC: Dependency-Aware Cache Management for Data Analytics Clusters

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

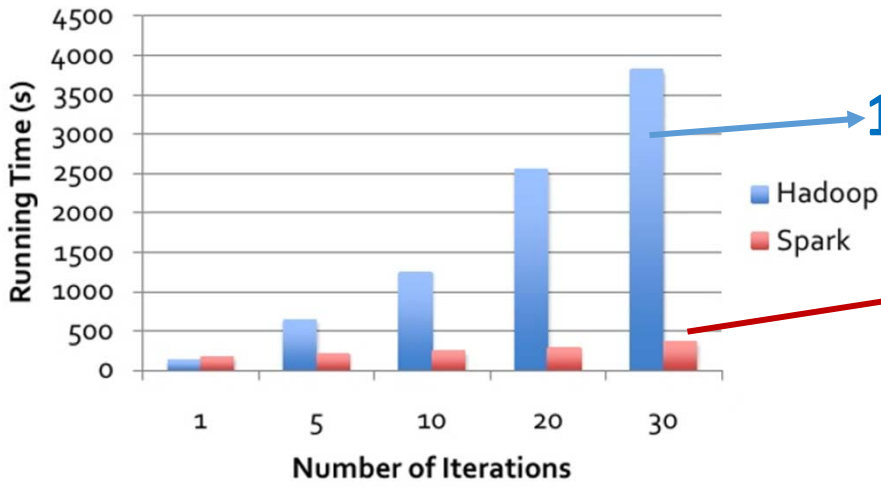
IEEE INFOCOM 2017

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.



[M. Zaharia, 2012]

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

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).

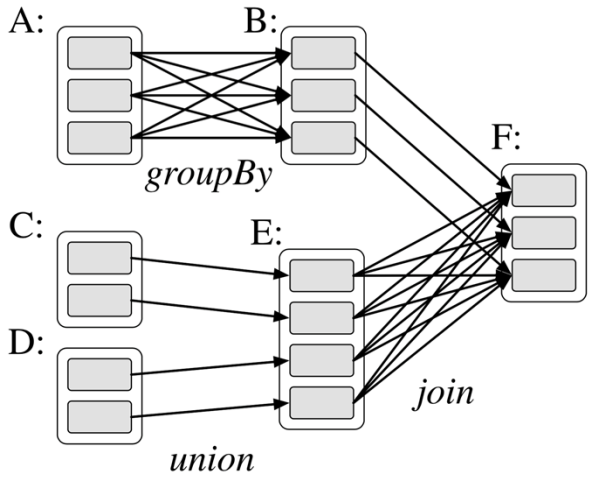
What's New for Data Analytics Clusters?

- 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 Polices**
- LRC: Dependency-Aware Cache Management
- Evaluations
- Conclusions

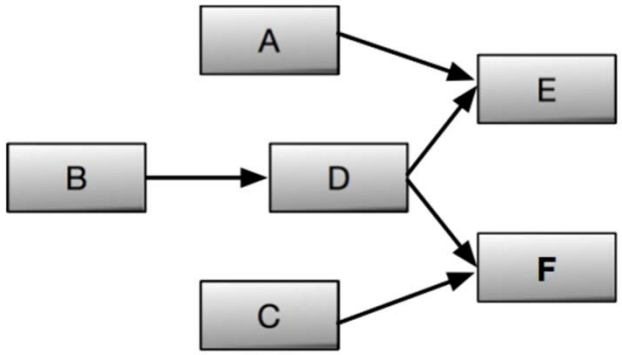
Inefficiency of Existing Cache Polices

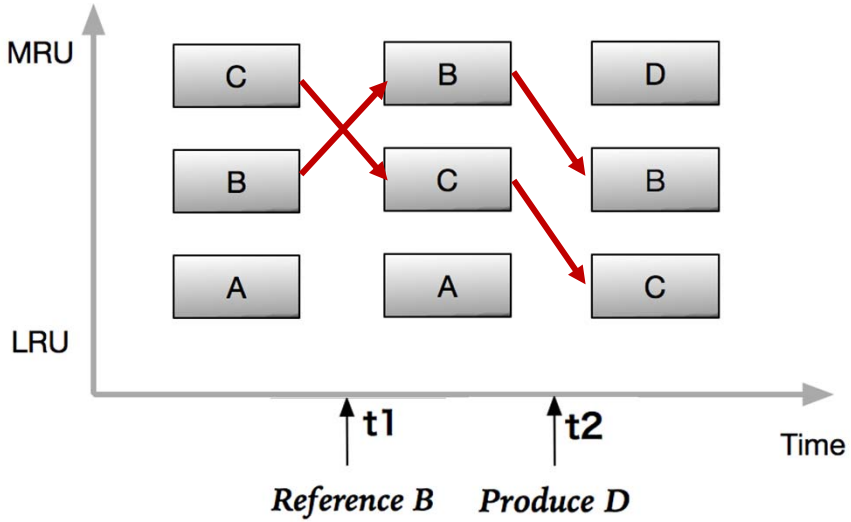
- Existing cache polices (LRU / LFU) are oblivious to the readily available data dependency information.

- 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 Polices

- 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

Application Type	Workload
Machine Learning	*Logistic Regression*
	Support Vector Machine (SVM)
	Matrix Factorization
Graph Computation	*Page Rank*
	SVD Plus Plus
	Triangle Count
SQL Queries	*Hive*
	RDD Relation
Streaming Workloads	*Twitter Tag*
	Page View
Other Workloads	*Connected Component*
	Strongly Connected Component
	Shortest Paths
	Label Propagation
	Pregel Operation

Median: 77%

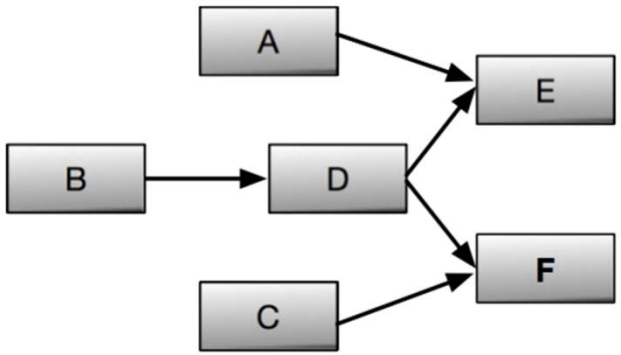
A New Cache Policy is Desired

- 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



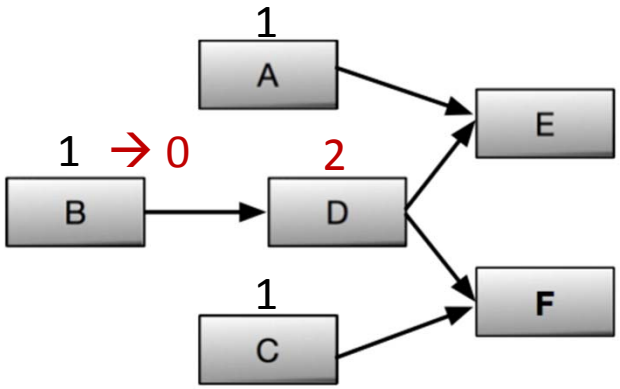
- Which one of block A and C is accessed first?

Outline

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

LRC: Dependency-Aware Cache Management

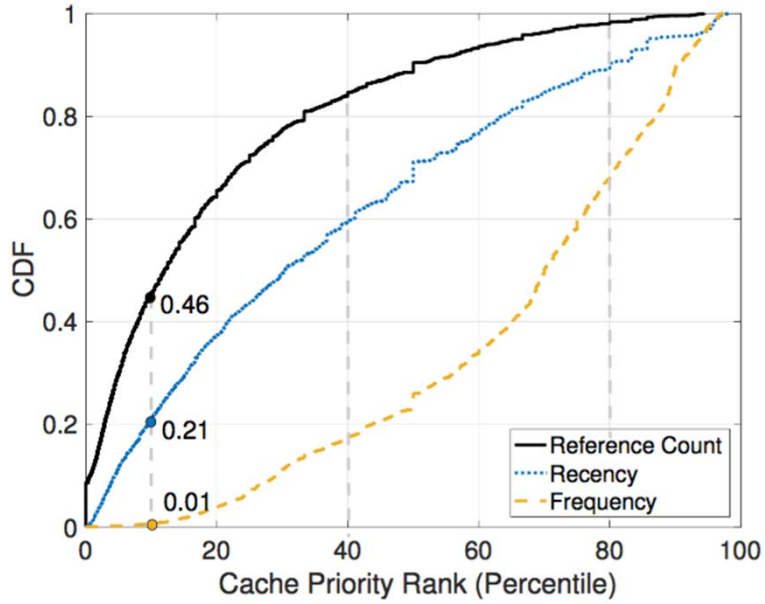
- **Reference count**: defined for each data block as the number of downstream tasks depending on it.
  - Dynamically changing over time:



- 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

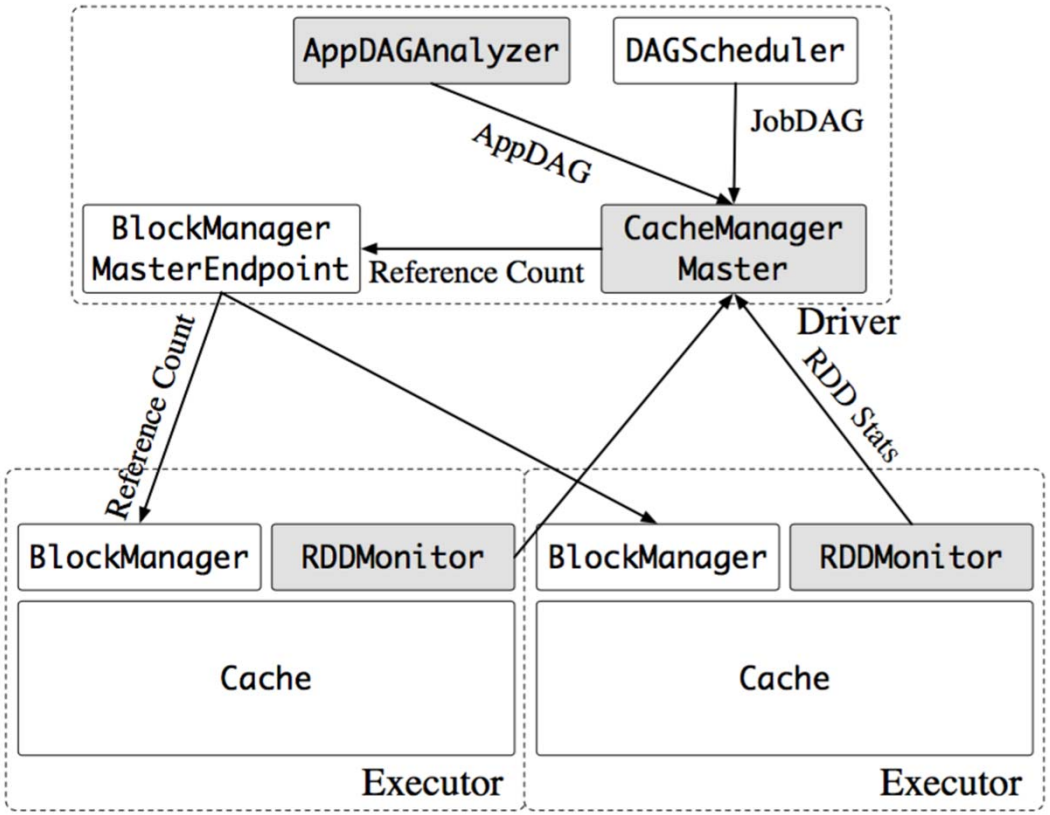
- Reference count ↑
  - Number of downstream tasks accessing it ↑
  - Chance to be used in the near future ↑
- Empirical study: reference count is a more accurate indicator to predict future data access than recency/frequency.



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 Polices
- 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

Workload Characterization

- Not all applications benefit from the improvement of cache management.

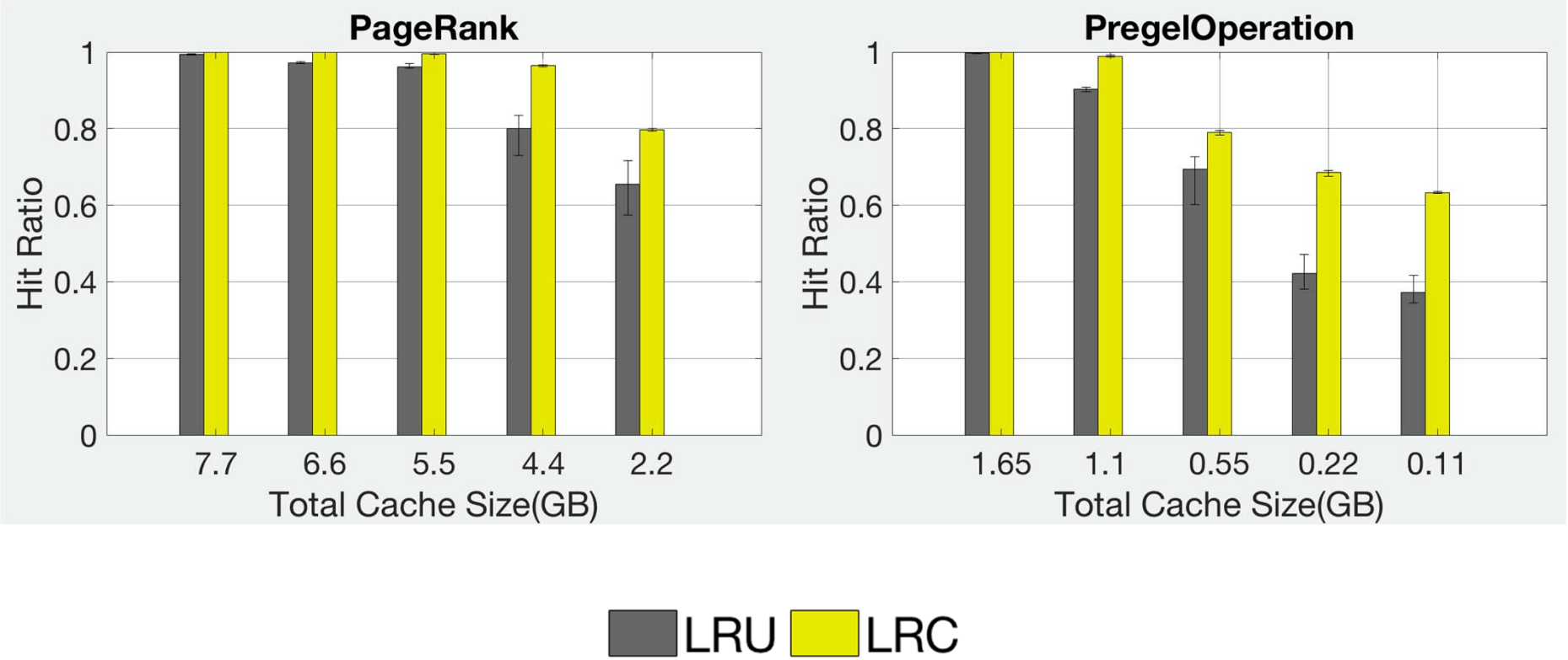
Workload	Cache All	Cache None
Page Rank	56 s	552 s
Connected Component	34 s	72 s
Shortest Paths	36 s	78 s
K-Means	26 s	30 s
Pregel Operation	42 s	156 s
Strongly Connected Component	126 s	216 s
Label Propagation	34 s	37 s
SVD Plus Plus	55 s	120 s
Triangle Count	84 s	99 s
Support Vector Machine (SVM)	72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

Evaluation Summary

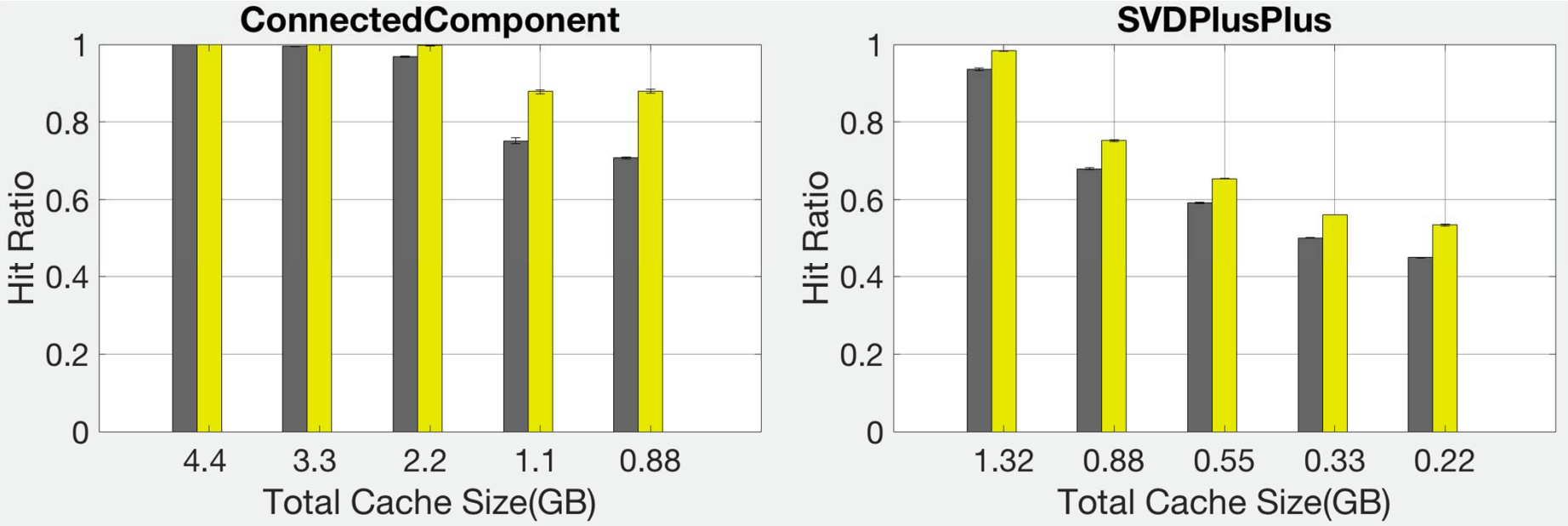
- Summary: maximum saving of application runtime

Workload	Cache Size	LRU	LRC	Speedup by LRC
Page Rank	6.6 GB	169.3 s	68.4 s	59.58%
Pregel Operation	0.22 GB	121.9 s	66.3 s	45.64%
Connected Component	2.2 GB	50.6 s	27.6 s	45.47%
SVD Plus Plus	0.88 GB	254.3 s	177.6 s	30.17%

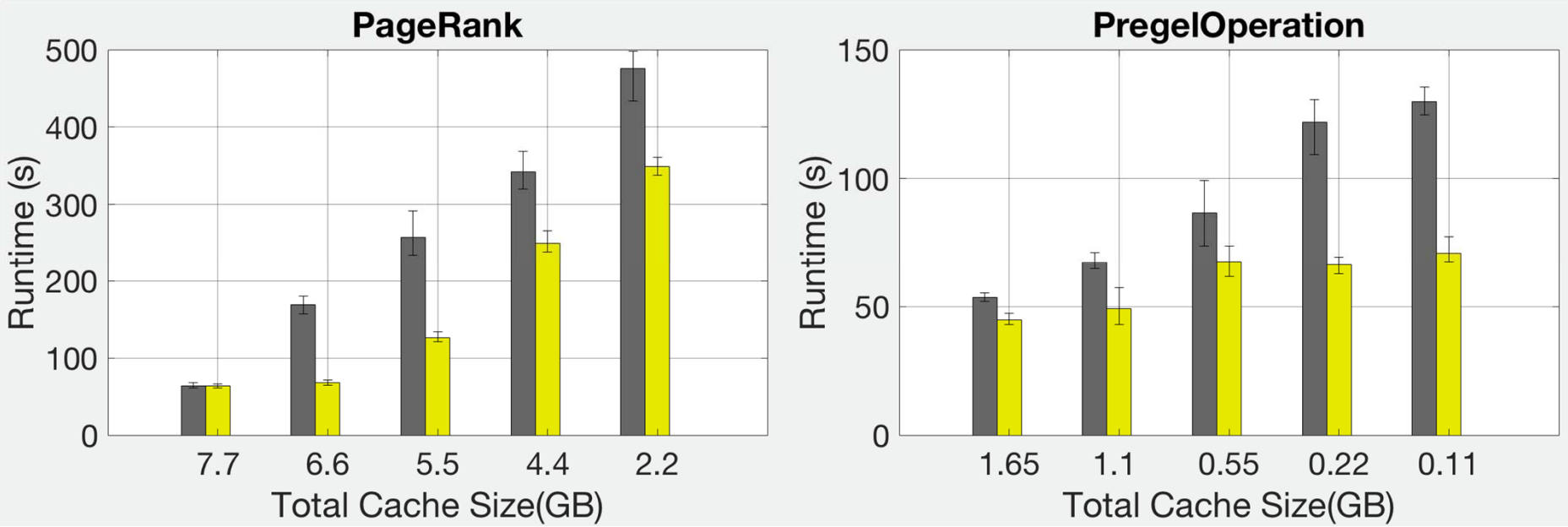
Cache Hit Ratio - 1



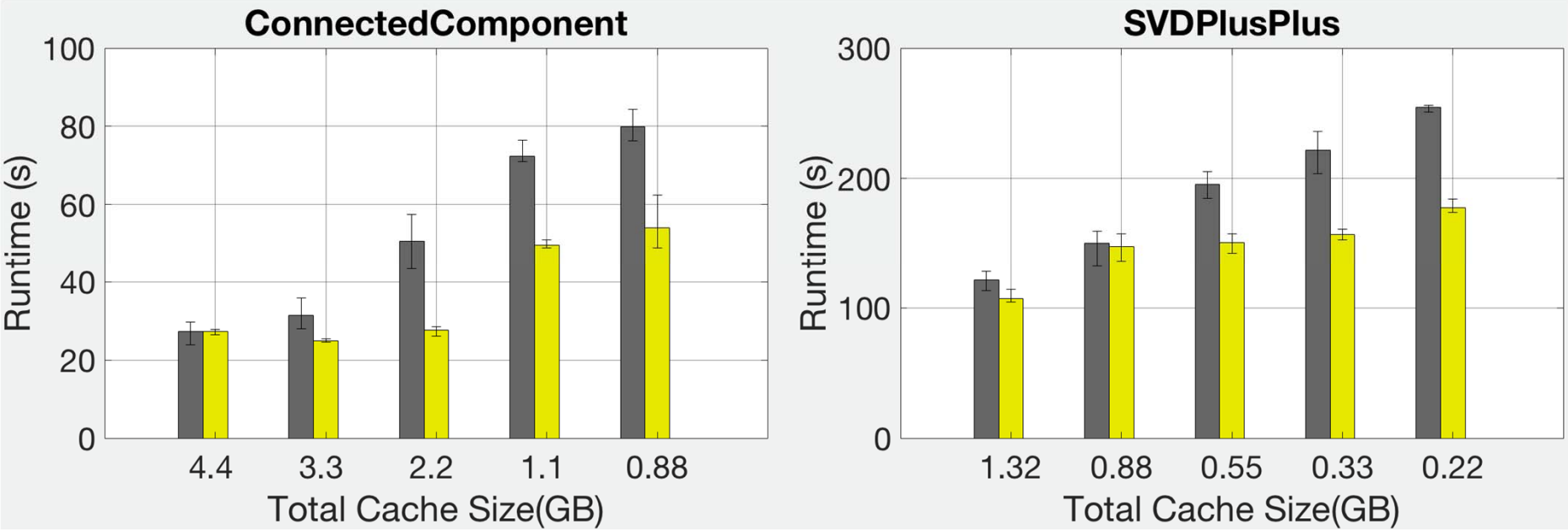
Cache Hit Ratio - 2



Application Runtime - 1



Application Runtime - 2



The advantage of LRC becomes more prominent when the cache size decreases.

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Outline

- Cache Management for Data Analytics Clusters
- Inefficiency of Existing Cache Polices
- 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?