PACMan

Coordinated Memory Caching for Parallel Jobs

Ganesh Ananthanarayanan, Ali Ghodsi, Andrew Wang,
Dhruba Borthakur, Srikanth Kandula,
Scott Shenker, Ion Stoica

amplab

Berkeley University of California

facebook

Microsoft Research

Data Analytics Clusters

- Data analytics frameworks are an important driver for modern Internet services
  - E.g., MapReduce, Dryad, Hadoop
  - Jobs are parallel and data-intensive

- Jobs sizes follow the power-law [HotOS'11]
  - 92% of jobs at FB's Hadoop cluster can fit all their data in memory

Cache the data to speed up jobs

- Falling memory prices
  - 64GB/machine at FB in Aug 2011, 192GB/machine not uncommon now

- Memory utilization often low
  - Analyzed Hadoop jobs in Facebook’s production cluster
  - 19% median memory utilization (95th-tile 42%)

We built a memory cache…

- File cache in memory on top of HDFS
  - Cache input data of jobs (accessed by map tasks)

- Schedule map tasks for memory locality

- Simple cache replacement policies
  - Least Recently Used (LRU) and Least Frequently Used (LFU)

We built a memory cache…

- Replayed the Facebook trace of Hadoop jobs
- Jobs sped up by only 10%, hit-ratio of 47% (for LFU) ☹
- Optimal hit-ratio (Belady's MIN Oracle)
  - Hit-ratio 63%
  - Completion time speedup 13%

How can we make caching significantly speedup jobs?

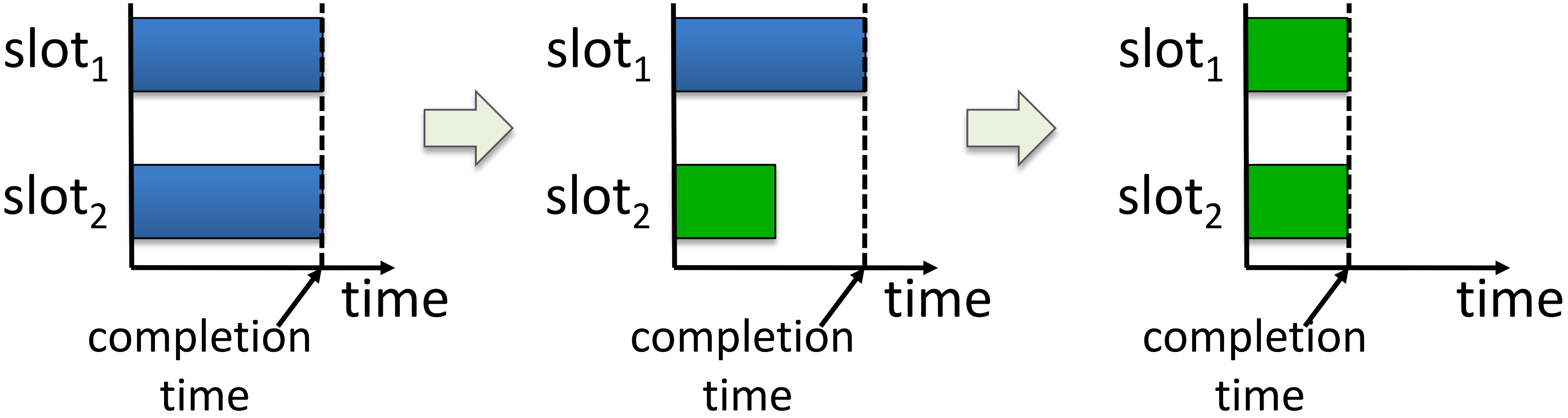
Parallel jobs require a new class of caching algorithms

Parallel Jobs

- Tasks of small jobs run simultaneously in a ***wave***

🟦 Task duration (**uncached** input)

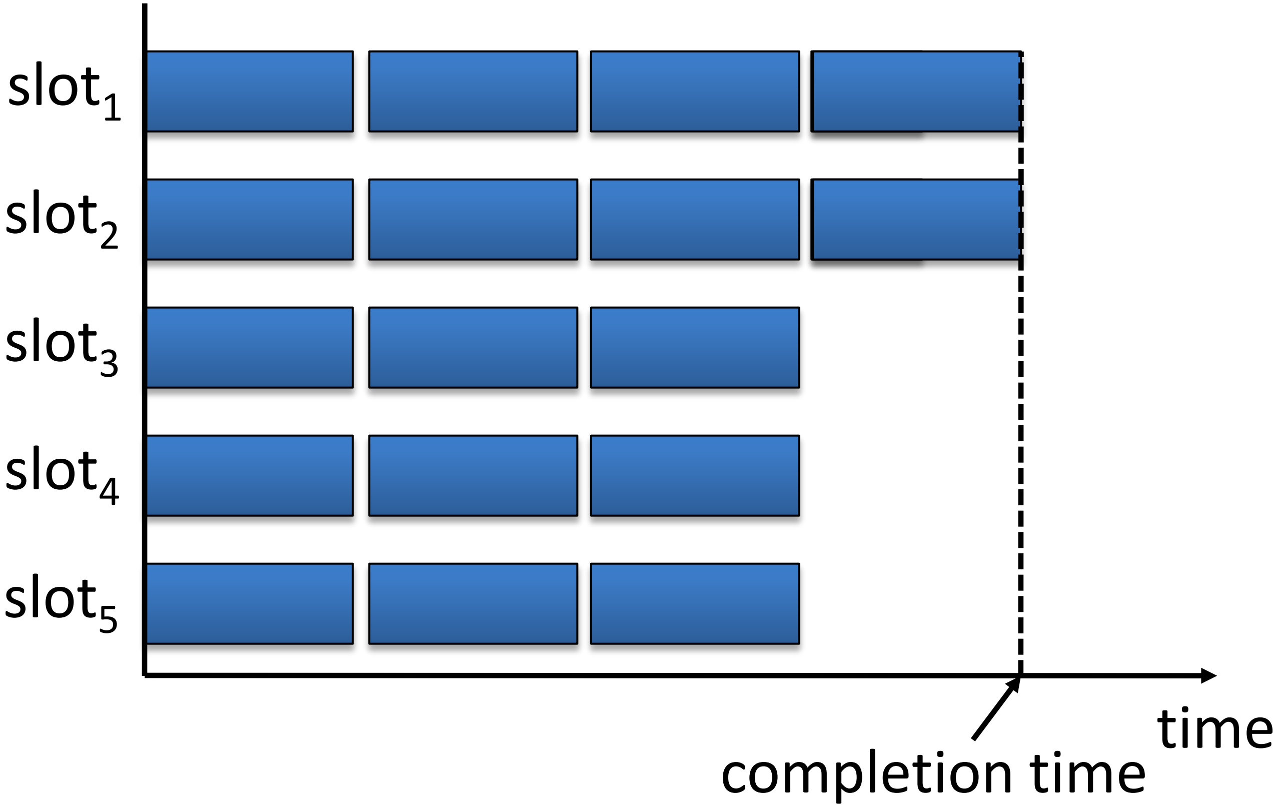
🟩 Task duration (**cached** input)



All-or-nothing: Unless all inputs are cached, there is no benefit

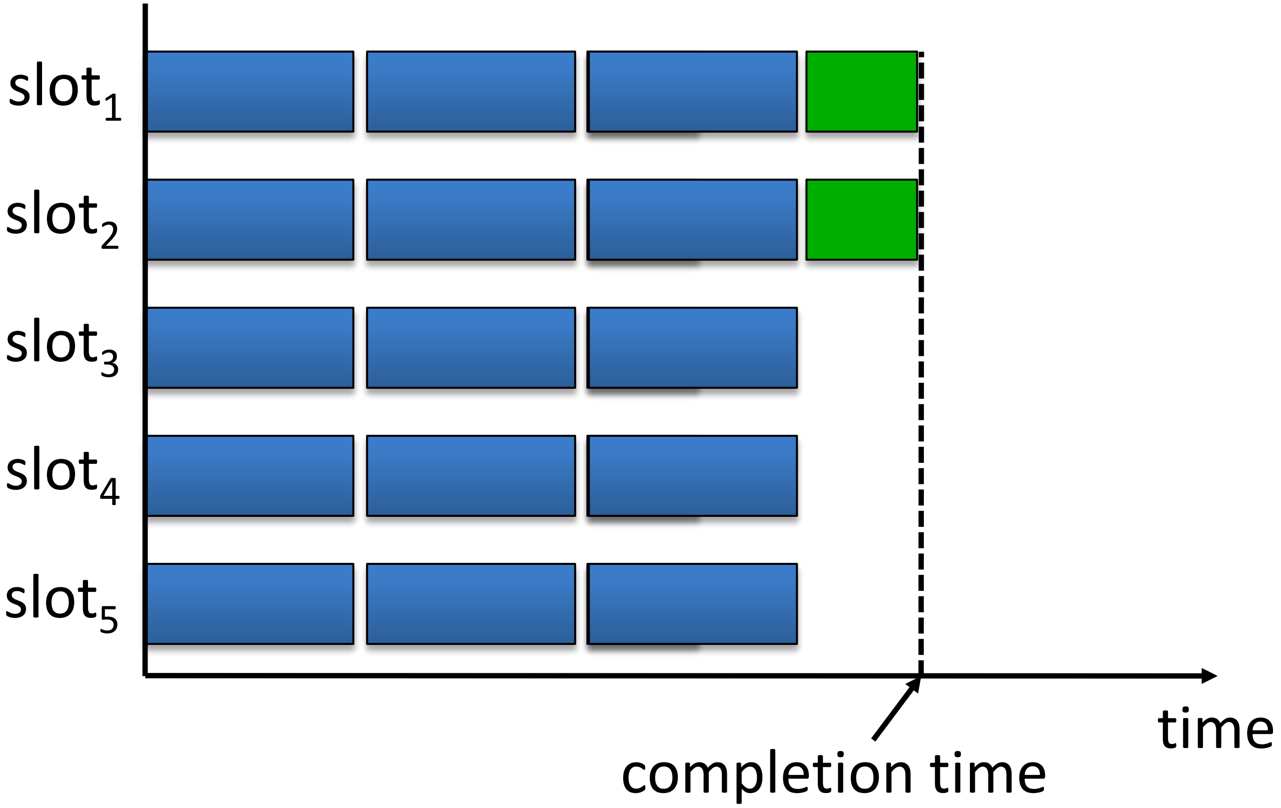
All-or-nothing for multi-waved jobs

- Large jobs run tasks in *multiple waves*
  - Number of tasks is larger than number of slots
  - **Wave-width**: Number of parallel tasks of a job



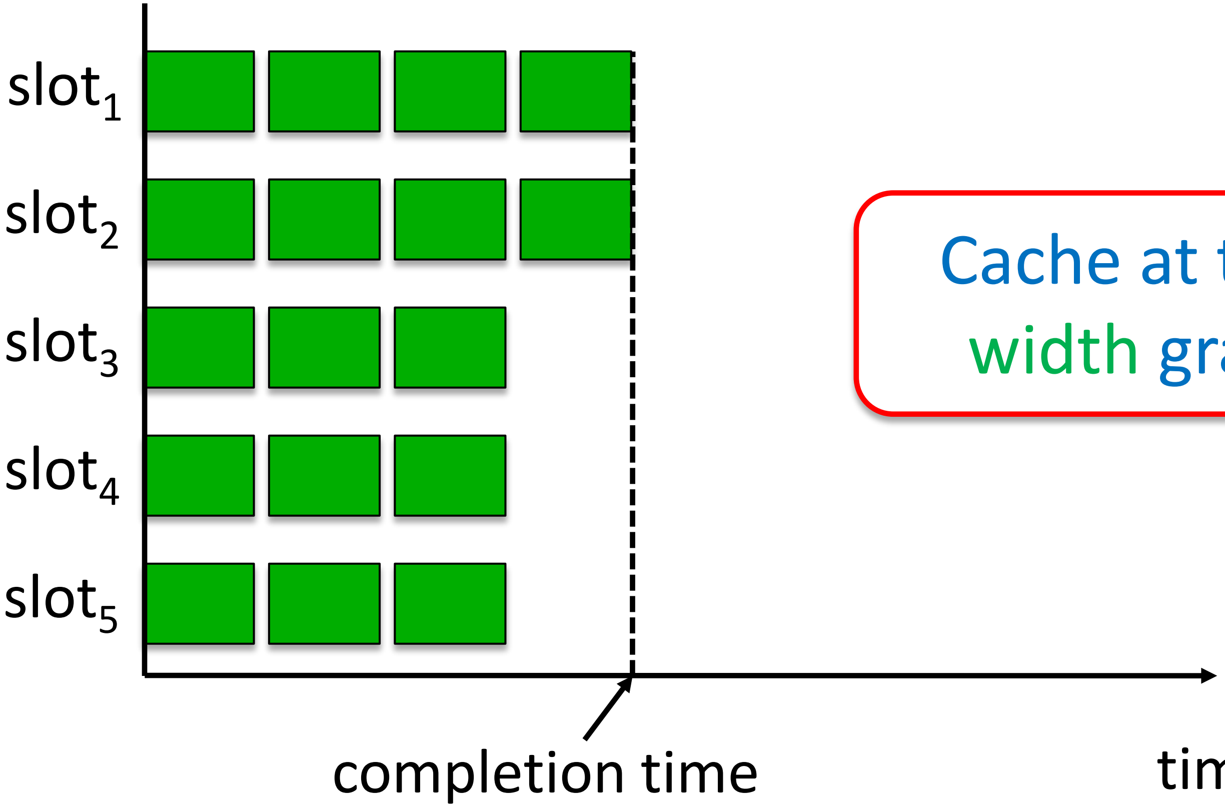
All-or-nothing for multi-waved jobs

- Large jobs run tasks in *multiple waves*
  - Number of tasks is larger than number of slots
  - **Wave-width**: Number of parallel tasks of a job



All-or-nothing for multi-waved jobs

- Large jobs run tasks in *multiple waves*
  - Number of tasks is larger than number of slots
  - **Wave-width**: Number of parallel tasks of a job



Cache at the wave-width granularity

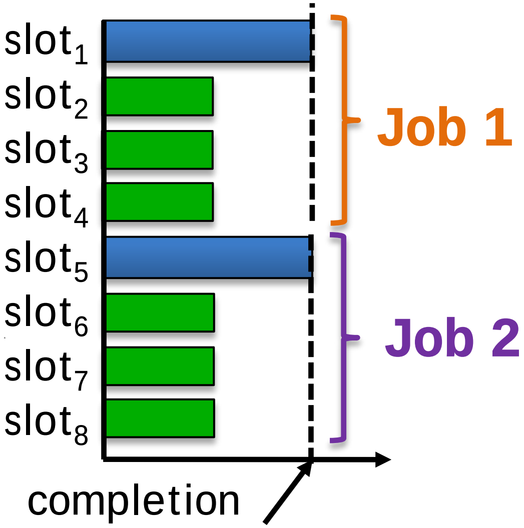
How to evict from cache?

- View at the granularity of a job's input (*file)*
- Focus evictions on incompletely cached waves– **Sticky Policy**

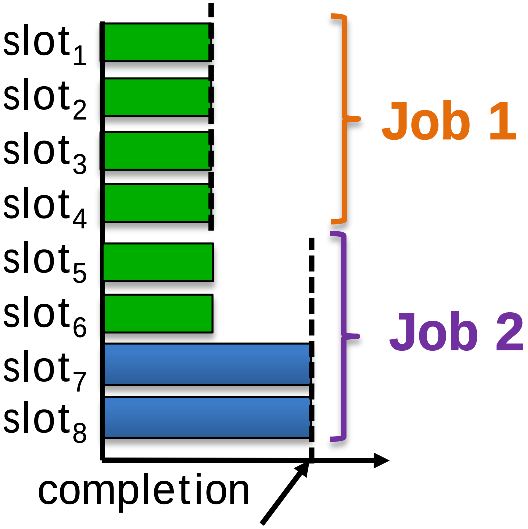
🟦 Task duration (**uncached** input)

🟩 Task duration (**cached** input)

Without Sticky Policy



Hit-ratio: 75%
No speed-up of jobs



With Sticky Policy

Hit-ratio: 75%
Job 1 speeds up

Which file should be evicted?

Depends on metric to optimize:

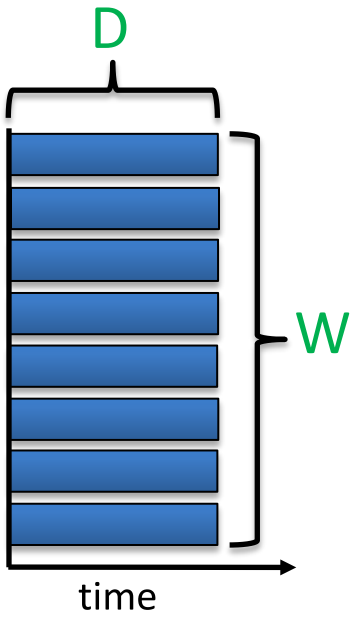
- User centric metric
  - **Completion time** of jobs

- System centric metric
  - **Utilization** of the cluster

What are the eviction policies for these metrics?

Reduction in Completion Time

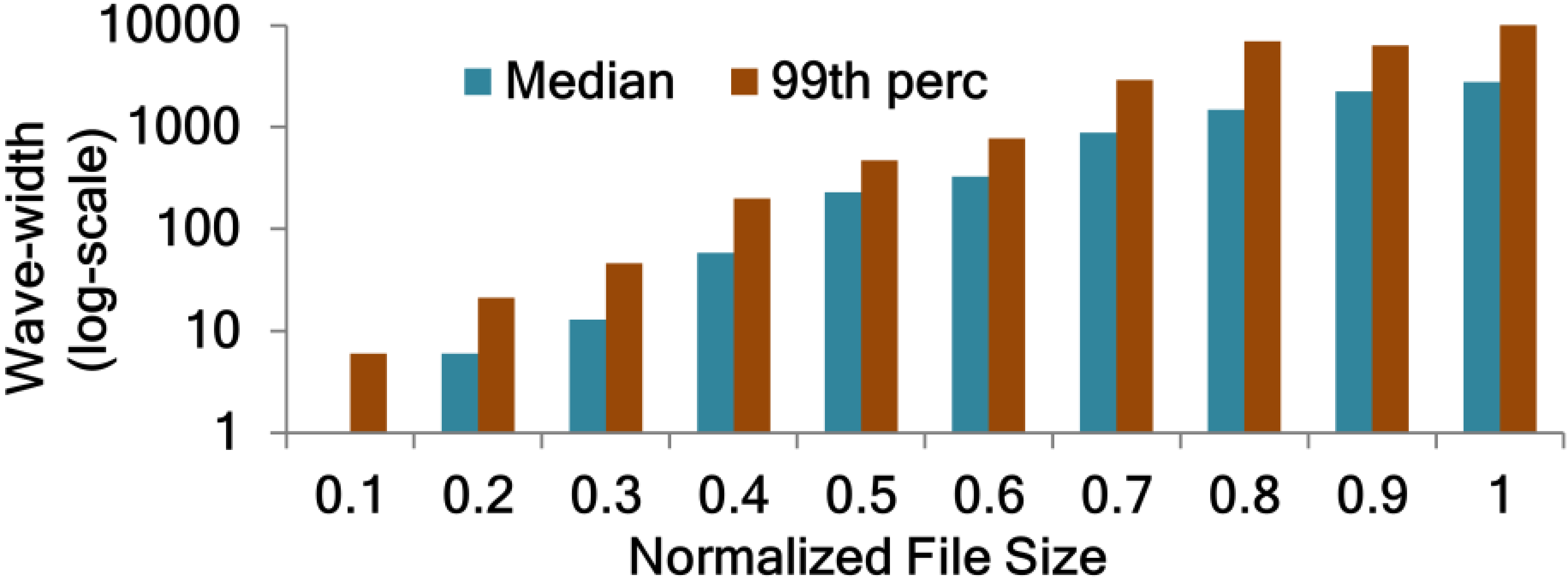
- Idealized model for job:
  - Wave-width for job: W
  - Frequency predicts future access: F
  - Data read is proportional to task length: D
  - Speedup factor for cached tasks: µ



- Cost of caching: W D
- Benefit of caching: µD F
- Benefit/cost: µF/W

Completion Time of Job: frequency/wave-width

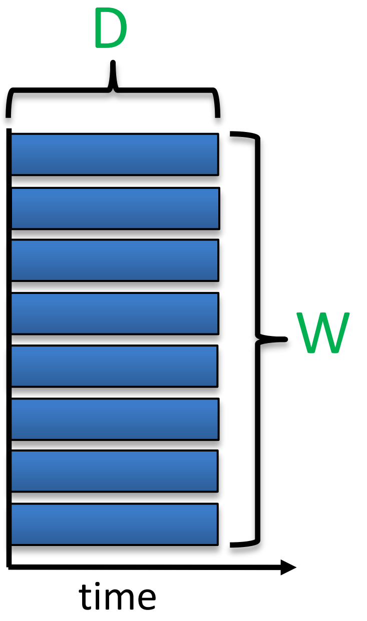
How to estimate W for a job?



- Use the size of a file as a proxy for wave-width
  - NSDI paper explains sophisticated approximation

Improvement in Utilization

- Idealized model for job:
  - Wave-width for job: W
  - Frequency predicts future access: F
  - Data read is proportional to task length: D
  - Speedup factor for cached tasks: µ

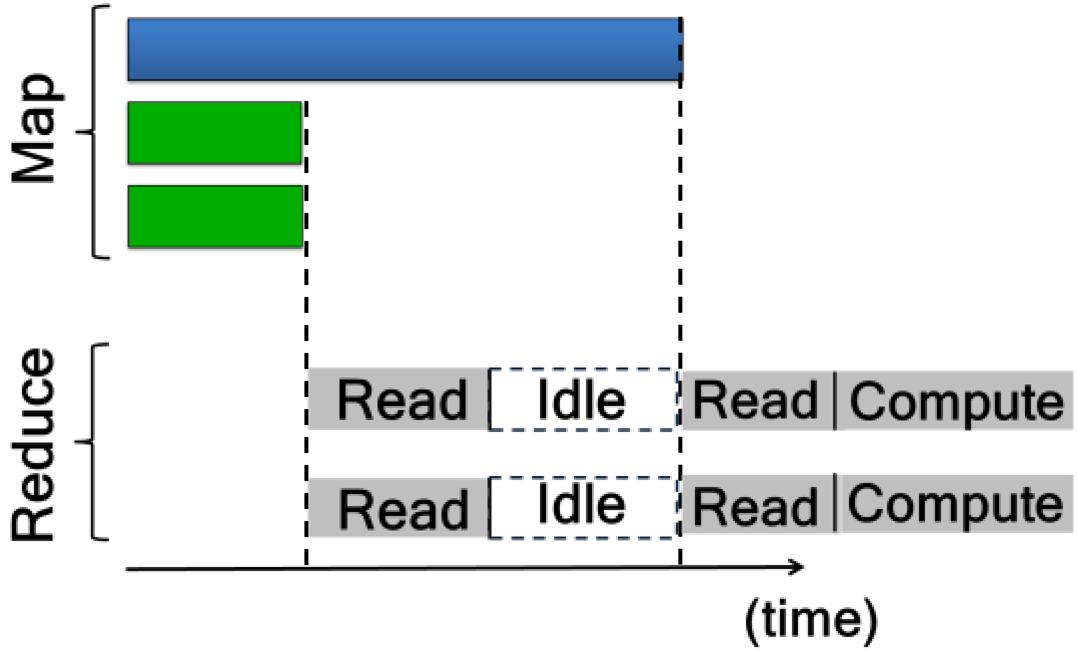


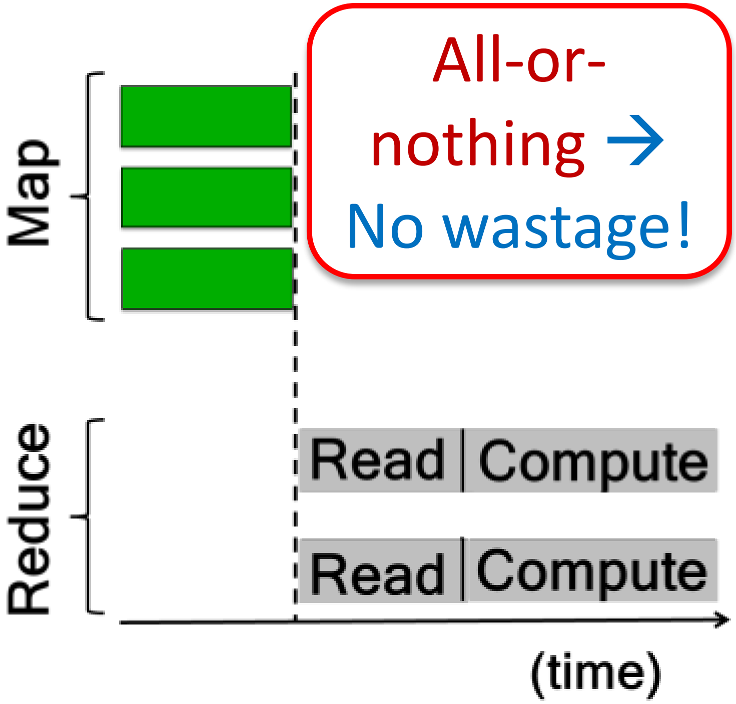
- Cost of caching: W D
- Benefit of caching: W µD F
- Benefit/cost: µF

Utilization of job: frequency

Isn’t this just Least Frequently Used?

- All-or-nothing property matters for utilization
- Tasks of different phases overlap
  - Reduce tasks start before all map tasks finish (to overlap communication)





All-or-nothing → No wastage!

Cache Eviction Policies

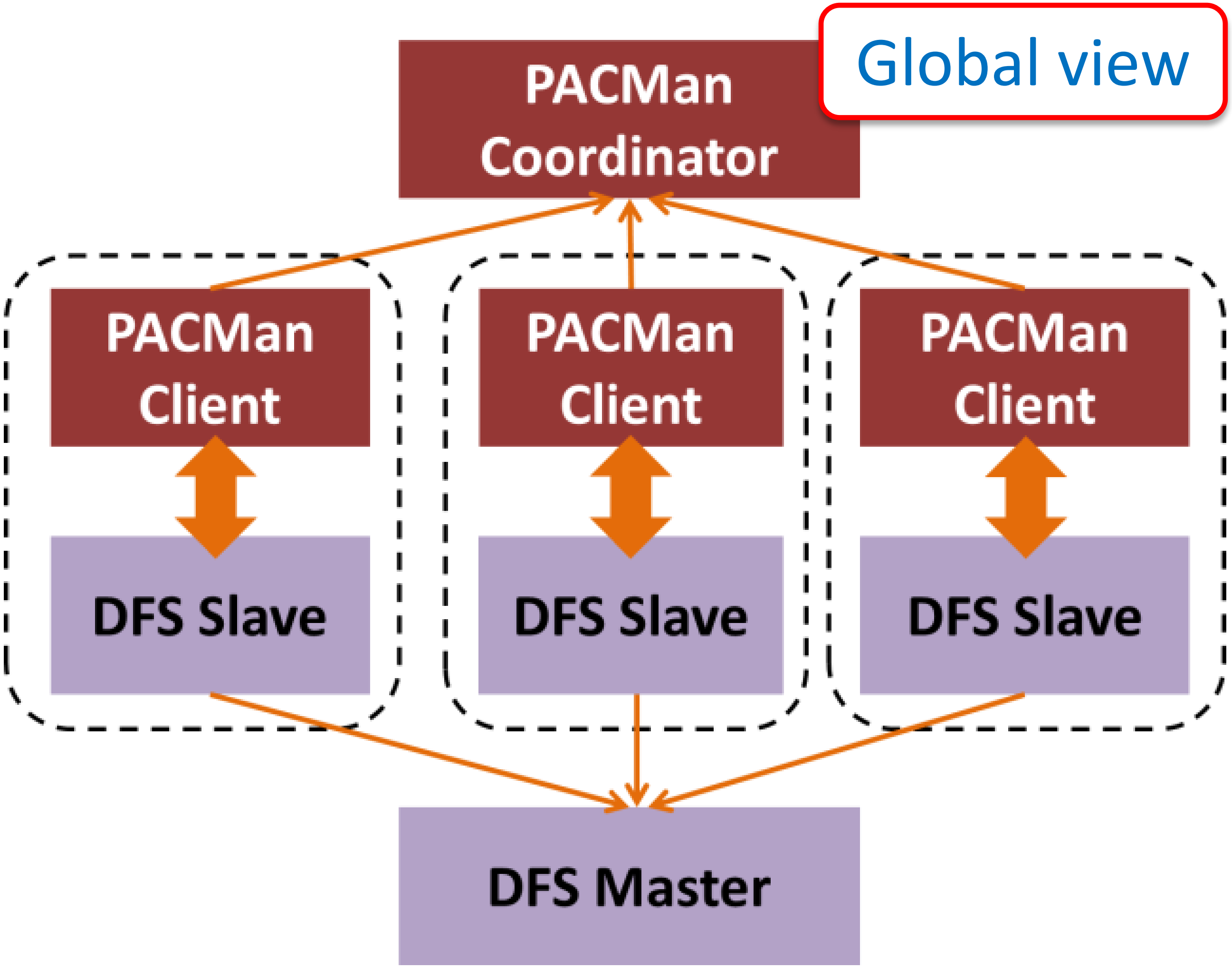
- Completion time policy: LIFE
  - Evict from file with *lowest* (*frequency/wave-width*)
  - Sticky: fully evict file before going to next (all-or-nothing)

- Utilization policy: LFU-F
  - Evict from file with the *lowest frequency*
  - Sticky: fully evict file before going to next (all-or-nothing)

How do we achieve the sticky policy?

- Caches are distributed
- Blocks of files are spread across different machines
- **Coordination**
  - Global view of all the caches
  - …which blocks to evict (sticky policy)
  - …where to schedule tasks (memory locality)

PACMan: Centralized Coordination



Evaluation Setup

- Workload derived from Facebook & Bing traces
  - FB: 3500 node Hadoop cluster, 375K jobs, 1 month
  - Bing: 1000’s of nodes Dryad cluster, 200K jobs, 6 weeks

- Prototype in conjunction with HDFS
- Experiments on 100-node EC2 cluster
  - Cache of 20GB per machine

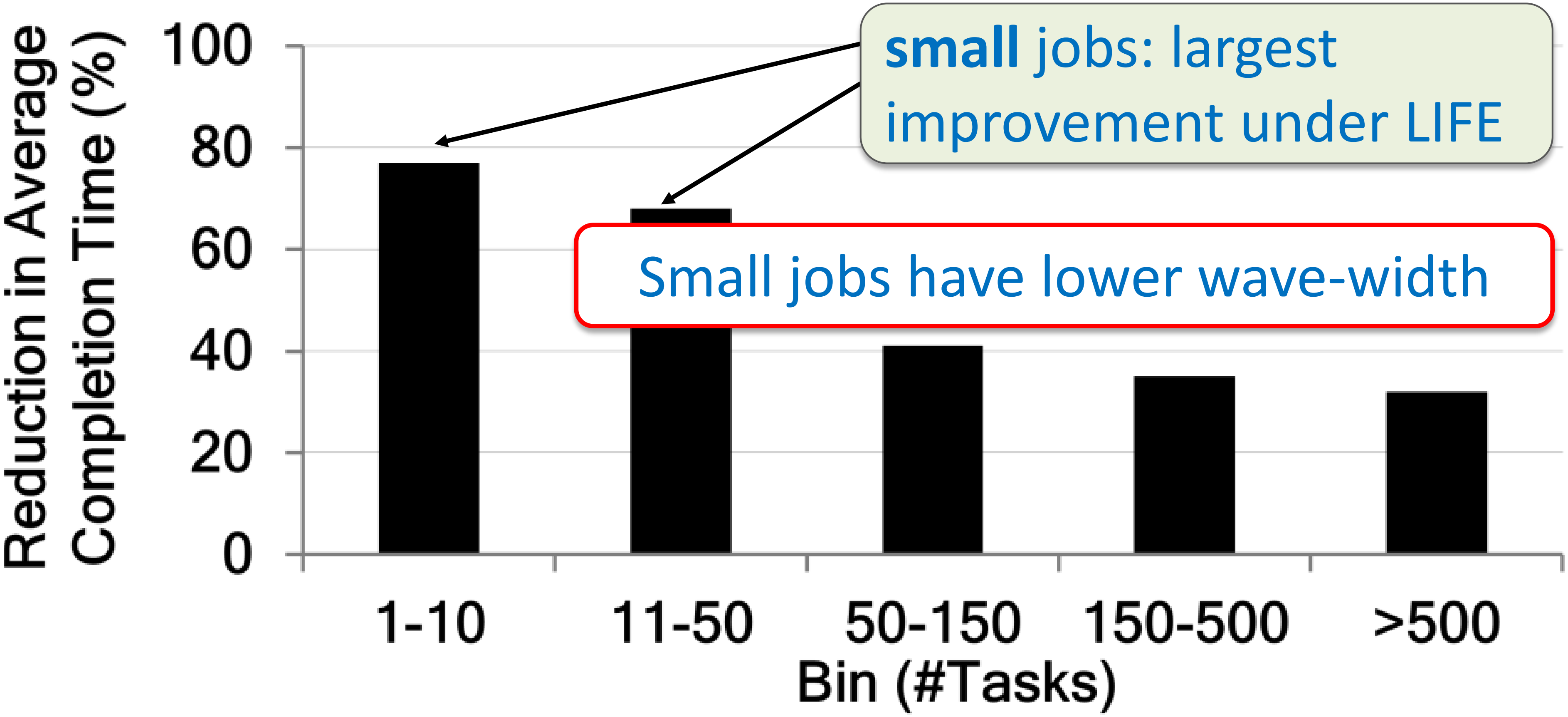
- Simulations
  - Replay of entire traces

Reduction in Completion Time

Replacement Policy	Reduction in average completion time (%)
LRU	9%
LFU	10%
MIN	13%
LIFE	53%

Sticky Policy

Which jobs are sped up?



small jobs: largest improvement under LIFE

Small jobs have lower wave-width

Power law in job sizes → there is space for large jobs too

Improvement in Utilization

Replacement Policy	Improvement in utilization (%)
LRU	13%
LFU	46%
MIN	51%
LFU-F	54%

Sticky Policy

What if we had an oracle?

- Optimal Cache Replacement
  - LP: Minimize average completion
  - 10% improvement in average completion time

- Cache prior to first access
  - One third of tasks read singly-accessed data
  - 27% improvement in average completion time

Pre-fetch & Pre-replace: Adds oracle capability to PACMan → 87% improvement

Related Work

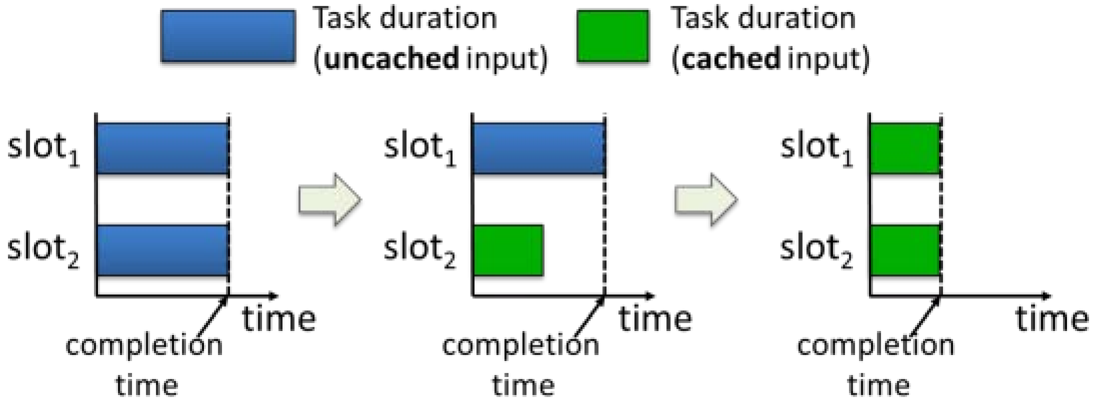
- In-memory computation frameworks [e.g., Spark, Piccolo, Twister]
  - Mediates cache access *across* jobs

- Memory Storage Systems [e.g., RAMCloud]
  - Data-intensive clusters cannot fit all data in memory; 200x more storage on disk than available memory

- Global Memory Systems [e.g., GMS, NOW]
  - Does not replace based on job-level granularity

Conclusions

- All-or-nothing property of parallel jobs
  - Cache all of the inputs of a job



- PACMan: Coordinated Cache Management
  - Sticky policy: Evict from incomplete inputs

- LIFE for completion time, LFU-F for utilization
- Jobs are 53% faster, cluster utilization improves by 54%