Delay Scheduling

A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling

Matei Zaharia, Dhruba Borthakur[*], Joydeep Sen Sarma[*], Khaled Elmeleegy[+], Scott Shenker, Ion Stoica

UC Berkeley, [*]Facebook Inc, [+]Yahoo! Research

RAD Lab

Motivation

- MapReduce / Hadoop originally designed for high throughput batch processing

- Today's workload is far more diverse:
  - *Many users* want to share a cluster
    - Engineering, marketing, business intelligence, etc
  - Vast majority of jobs are *short*
    - Ad-hoc queries, sampling, periodic reports
  - *Response time* is critical
    - Interactive queries, deadline-driven reports

➢ How can we efficiently share MapReduce clusters between users?

Example: Hadoop at Facebook

- 600-node, 2 PB data warehouse, growing at 15 TB/day
- Applications: data mining, spam detection, ads
- 200 users (half non-engineers)
- 7500 MapReduce jobs / day

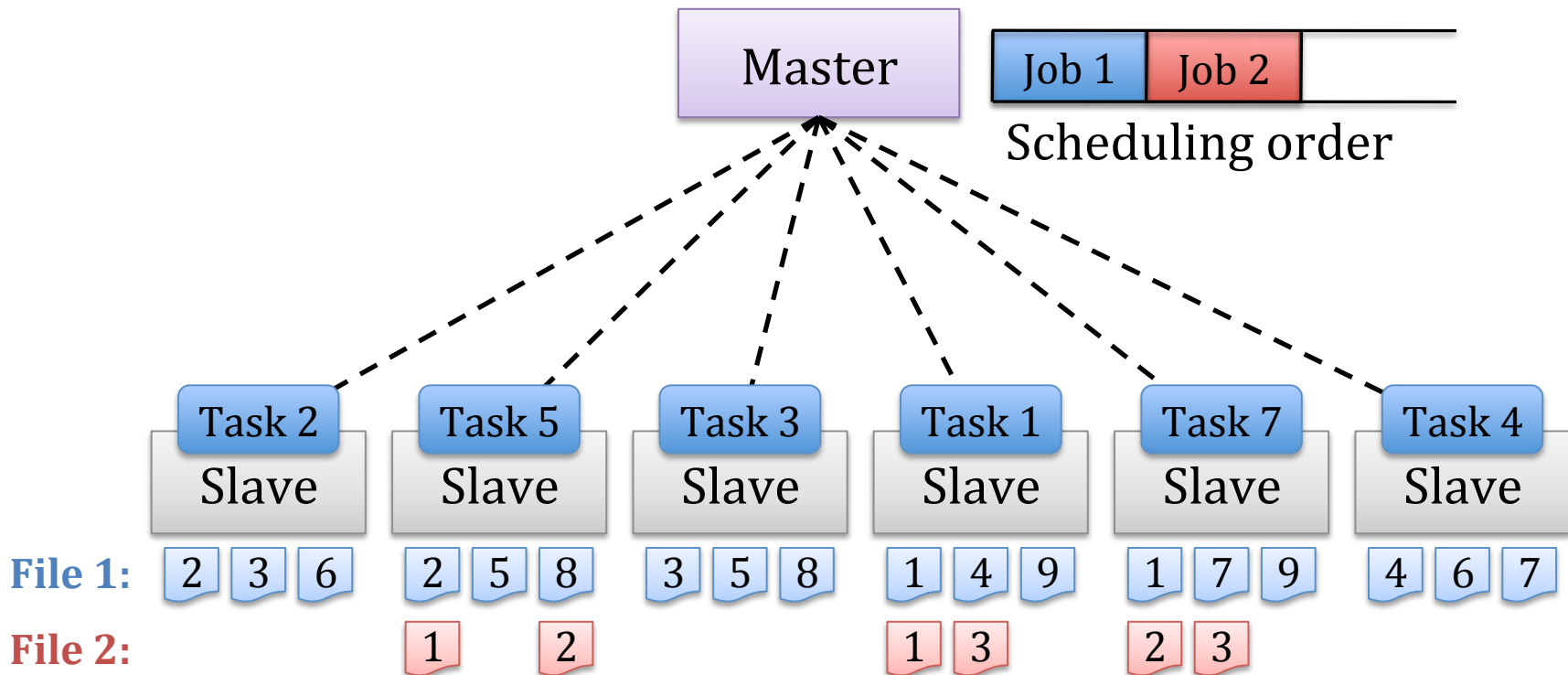CDF of Job Running Time (s) with Median: 84s
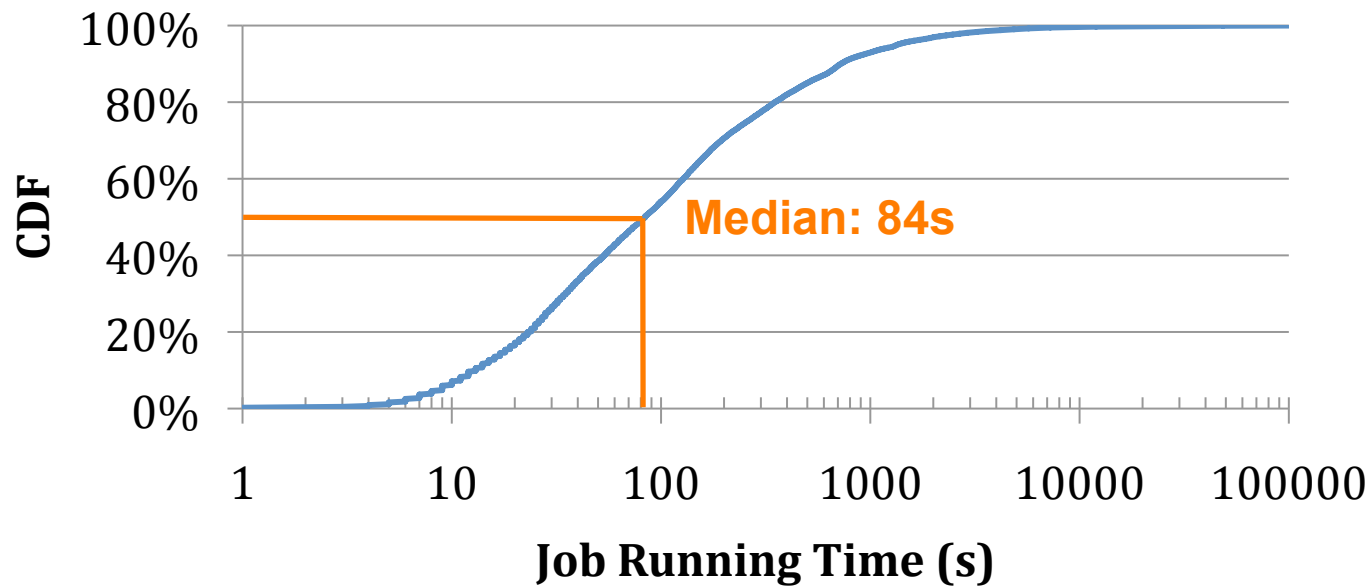
Approaches to Sharing

- Hadoop default scheduler (FIFO)
  - **Problem:** short jobs get stuck behind long ones

- Separate clusters
  - **Problem 1:** poor utilization
  - **Problem 2:** costly data replication
    - Full replication across clusters nearly infeasible at Facebook/Yahoo! scale
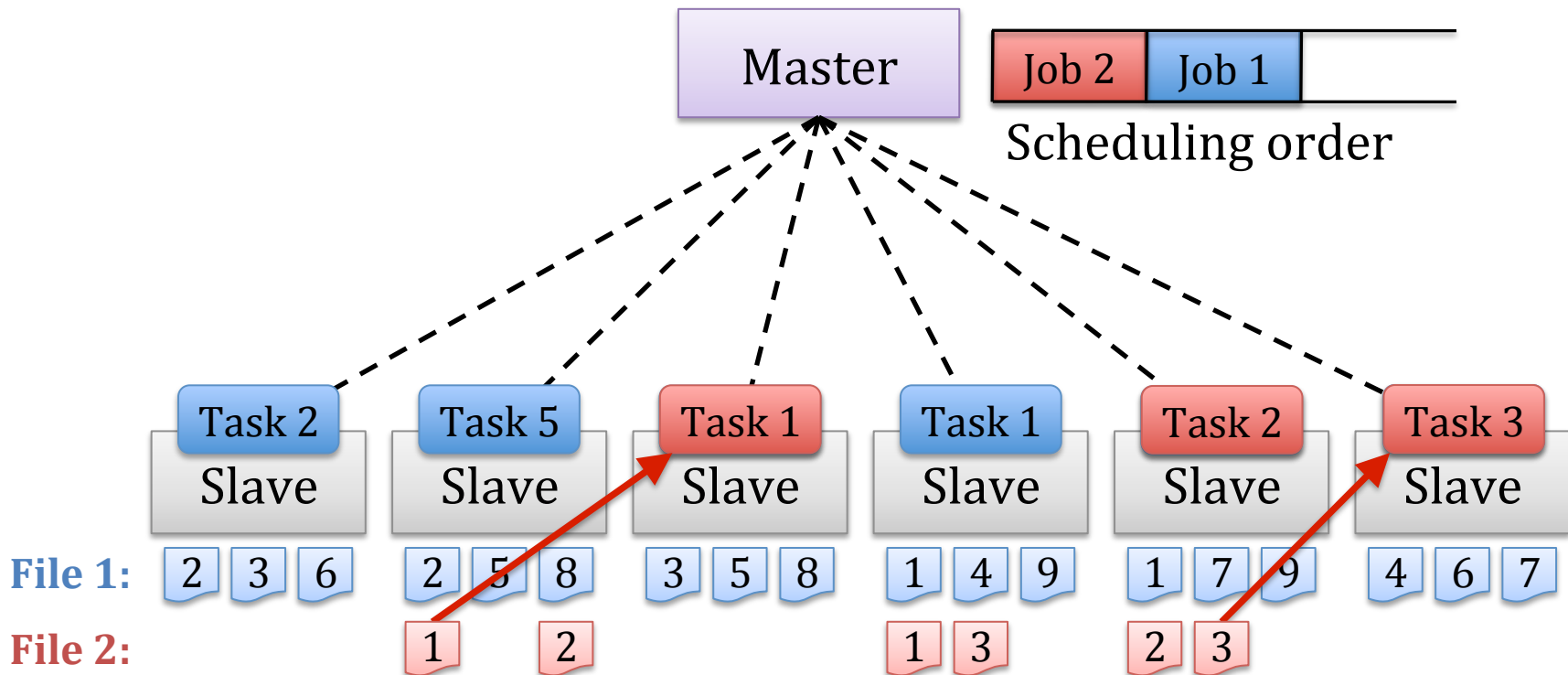   - Partial replication prevents cross-dataset queries

Our Work

- Hadoop Fair Scheduler
  - Fine-grained sharing at level of map & reduce tasks
  - Predictable response times and user isolation

- Main challenge: data locality
  - For efficiency, must run tasks near their input data
  - Strictly following *any* job queuing policy hurts locality: job picked by policy may not have data on free nodes

- Solution: delay scheduling
  - Relax queuing policy for limited time to achieve locality
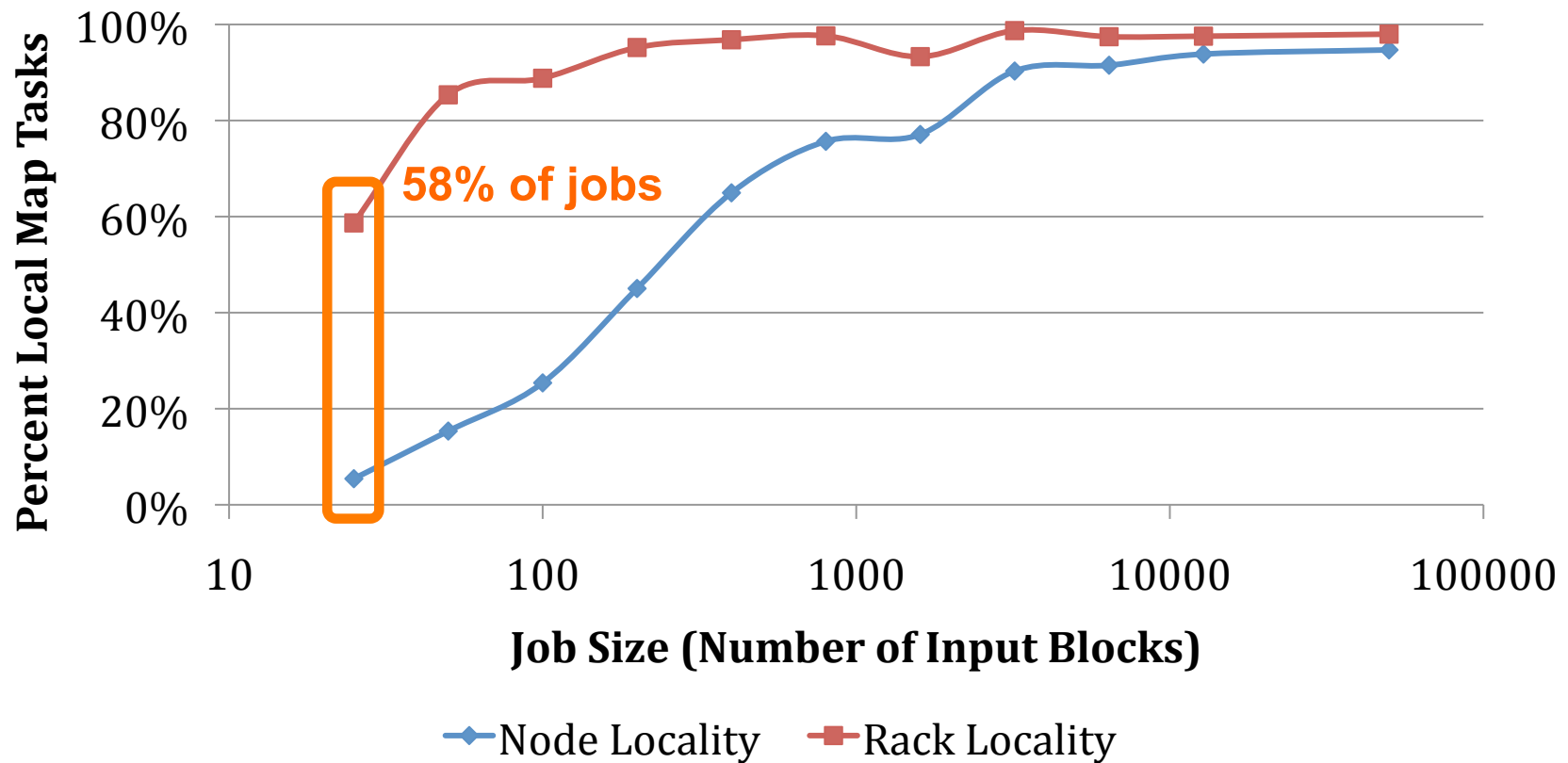
The Problem

The Problem

Master

Job 2 | Job 1

Scheduling order

| Task 2 | Task 5 | Task 1 | Task 1 | Task 2 | Task 3 |
| Slave | Slave | Slave | Slave | Slave | Slave |

File 1: 2 3 6 | 2 5 8 | 3 5 8 | 1 4 9 | 1 7 9 | 4 6 7

File 2: 1 2 | | 1 3 | 2 3 |

Problem: Fair decision hurts locality

Especially bad for jobs with small input files

Locality vs. Job Size at Facebook

Data Locality in Production at Facebook

58% of jobs

Node Locality — Rack Locality

Percent Local Map Tasks vs. Job Size (Number of Input Blocks)

Special Instance: Sticky Slots

- Under fair sharing, locality can be poor even when all jobs have large input files
- Problem: jobs get "stuck" in the same set of task slots
  - When one a task in job j finishes, the slot it was running in is given back to j, because j is below its share
  - Bad because data files are spread out across *all* nodes

Job	Fair Share	Running Tasks
Job 1	2	**1**
Job 2	2	2
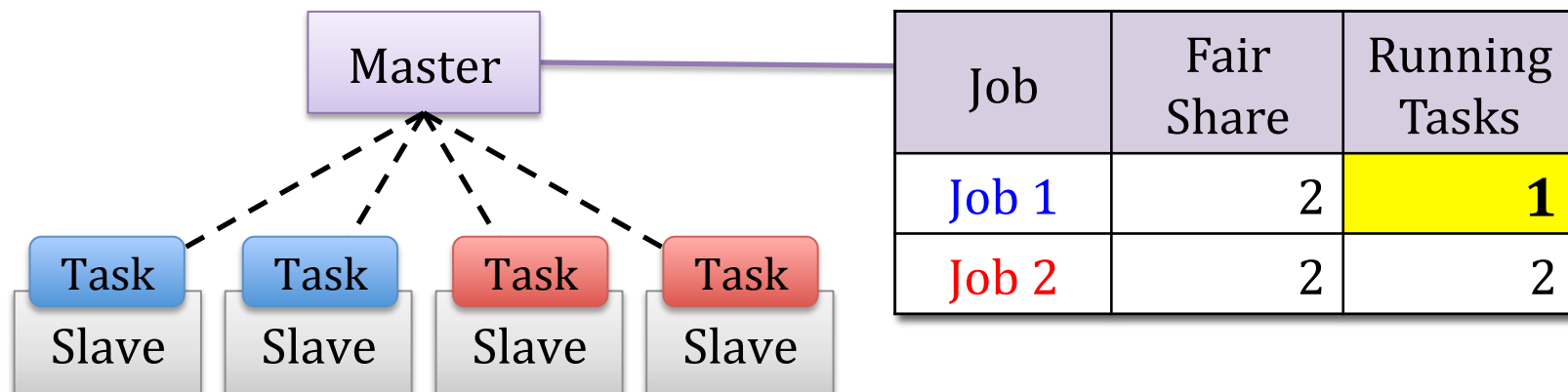
Special Instance: Sticky Slots

- Under fair sharing, locality can be poor even when all jobs have large input files
- Problem: jobs get "stuck" in the same set of task slots
  - When one a task in job j finishes, the slot it was running in is given back to j, because j is below its share
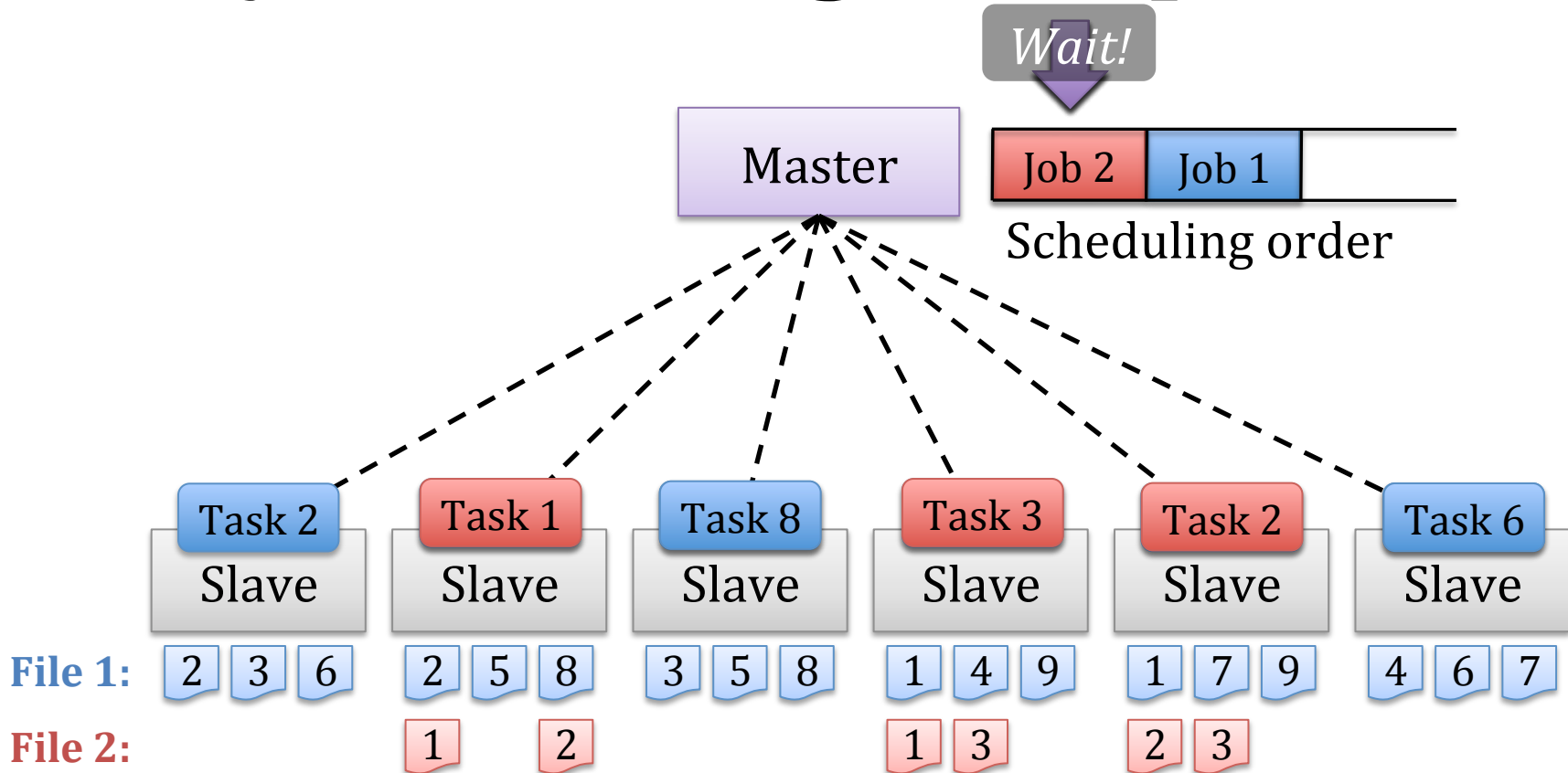 - Bad because data files are spread out across *all* nodes

Data Locality vs. Number of Concurrent Jobs
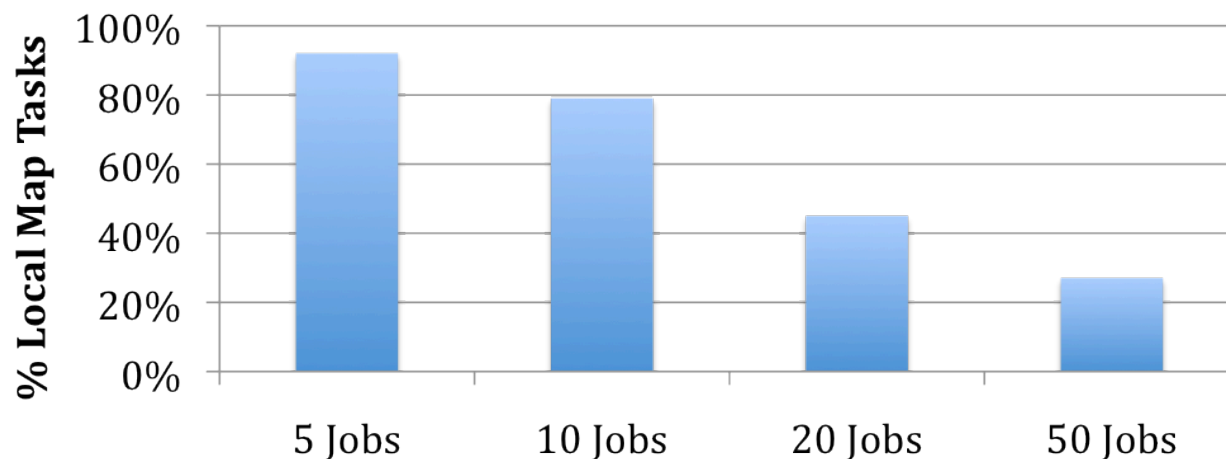
Solution: Delay Scheduling

- *Relax queuing policy* to make jobs wait for a limited time if they cannot launch local tasks

- Result: Very short wait time (1-5s) is enough to get nearly 100% locality

Delay Scheduling Example

Delay Scheduling Details

- Scan jobs in order given by queuing policy, picking first that is permitted to launch a task

- Jobs must wait before being permitted to launch non-local tasks
  - If wait < T_1, only allow node-local tasks
  - If T_1 < wait < T_2, also allow rack-local
  - If wait > T_2, also allow off-rack

- Increase a job's time waited when it is skipped

Analysis

- **When is it worth it to wait, and for how long?**

➤ *Waiting worth it if E(wait time) < cost to run non-locally*

- Simple model: Data-local scheduling opportunities arrive following Poisson process

- Expected response time gain from delay scheduling:

$$E(gain) = (1 - e^{-w/t})(d - t)$$

Wait amount

Expected time to get local scheduling opportunity

Delay from running non-locally

➤ *Optimal wait time is infinity*

Analysis Continued

$$E(\text{gain}) = (1 - e^{-w/t})(d - t)$$

Wait amount

Expected time to get local scheduling opportunity

Delay from running non-locally

- **Typical values of t and d:**

    t = (avg task length) / (file replication × tasks per node)

    d ≥ (avg task length)

    3

    6

More Analysis

- What if some nodes are occupied by long tasks?

- Suppose we have:
  - Fraction ϕ of active tasks are long
  - R replicas of each block
  - L task slots per node

- For a given data block,
    P(all replicas blocked by long tasks) = ϕ^{RL}

- With R = 3 and L = 6, this is less than 2% if $\phi < 0.8$

Evaluation
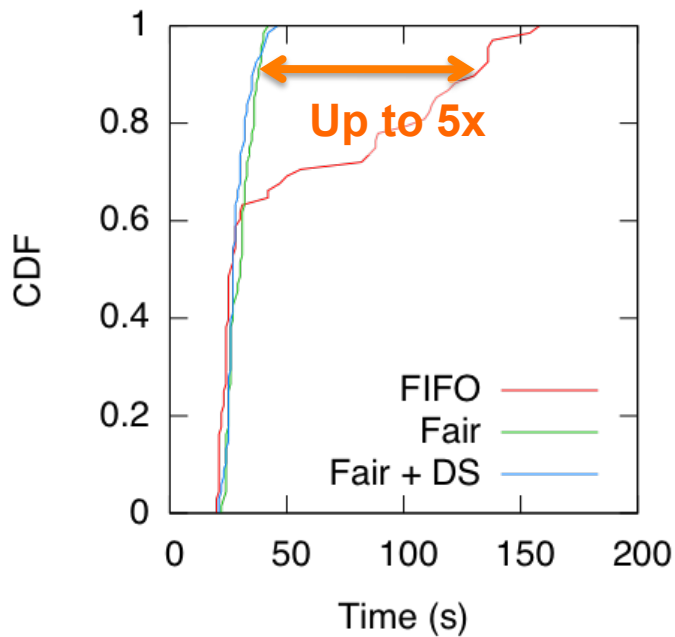
- Macrobenchmark
  - IO-heavy workload
  - CPU-heavy workload
  - Mixed workload
- Microbenchmarks
  - Sticky slots
  - Small jobs
  - Hierarchical fair scheduling
- Sensitivity analysis
- Scheduler overhead

Macrobenchmark

- 100-node EC2 cluster, 4 cores/node

- Job submission schedule based on job sizes and inter-arrival times at Facebook
  - 100 jobs grouped into 9 "bins" of sizes

- Three workloads:
  - IO-heavy, CPU-heavy, mixed

- Three schedulers:
  - FIFO
  - Fair sharing
  - Fair + delay scheduling (wait time = 5s)
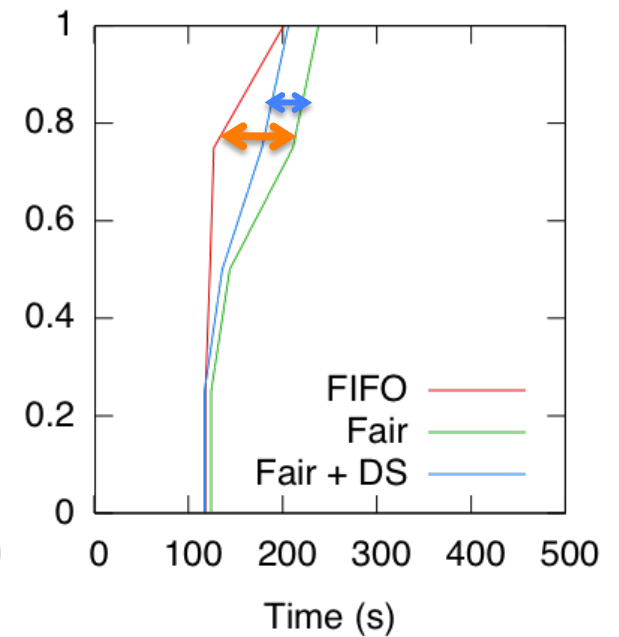
Results for IO-Heavy Workload

Job Response Times

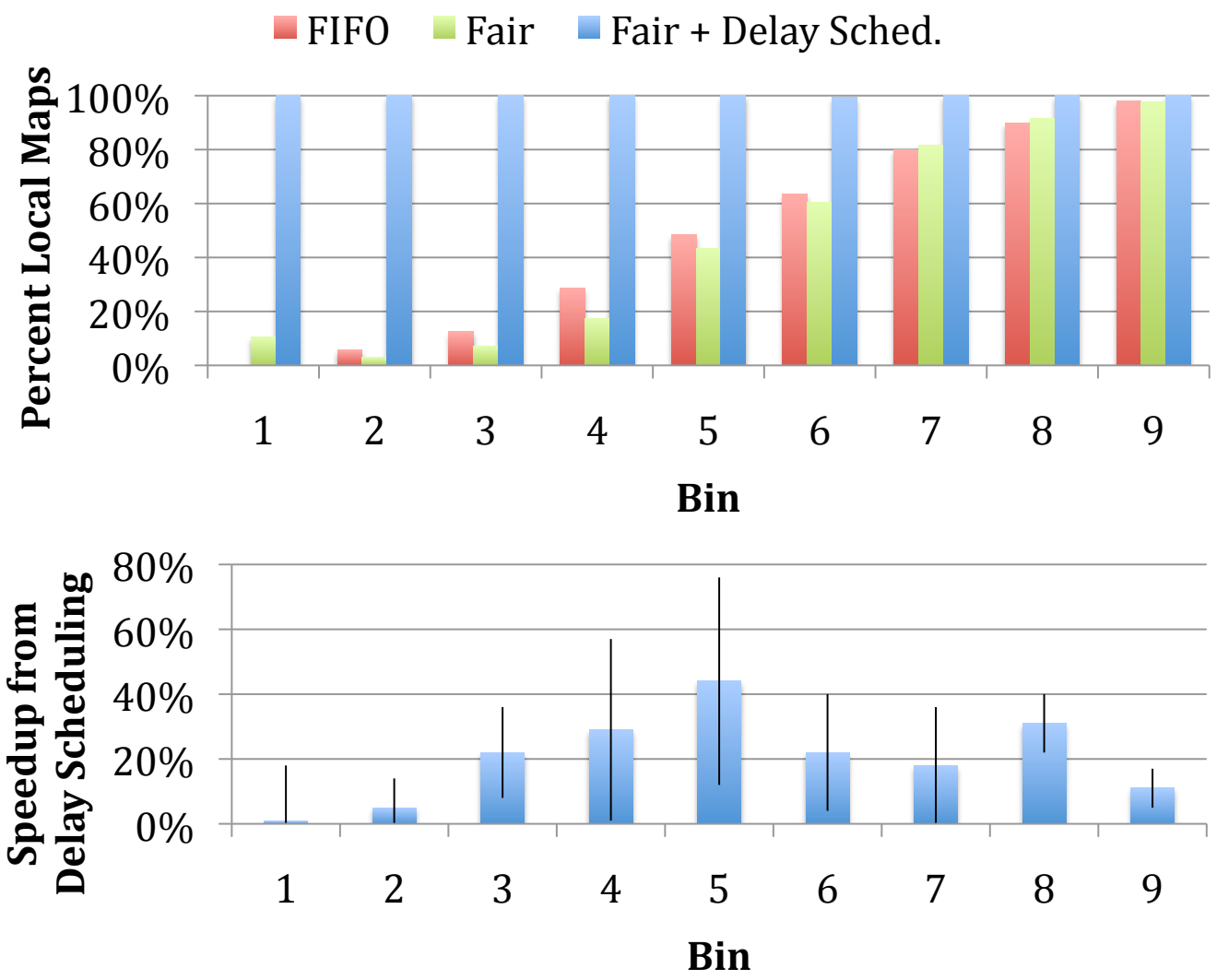Small Jobs
(1-10 input blocks)
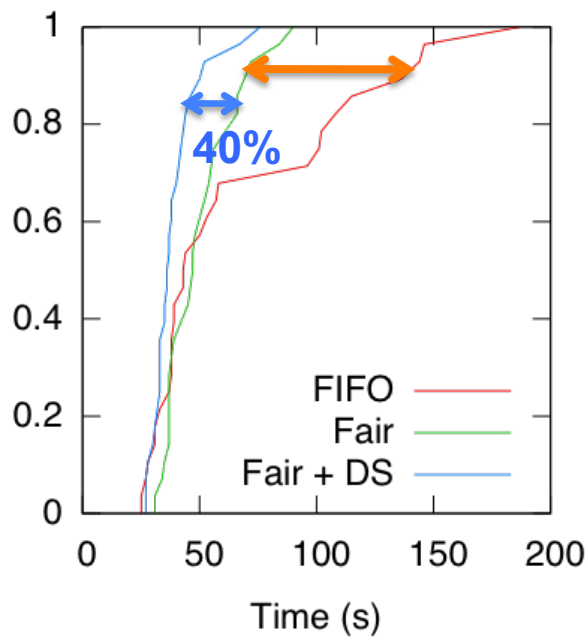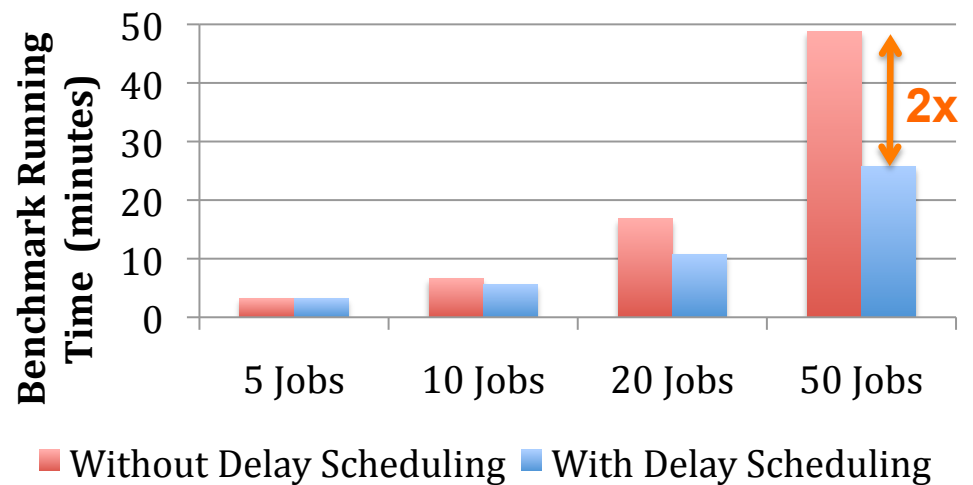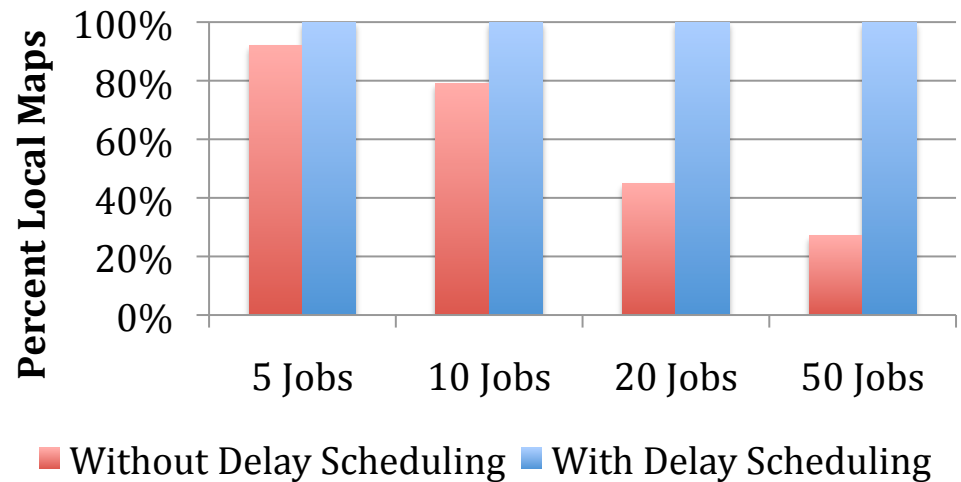
Medium Jobs
(50-800 input blocks)

Large Jobs
(4800 input blocks)

Results for IO-Heavy Workload

Sticky Slots Microbenchmark

- 5-50 jobs on EC2

- 100-node cluster

- 4 cores / node

- 5s delay scheduling

Conclusions

- Delay scheduling works well under two conditions:
  - Sufficient fraction of tasks are *short* relative to jobs
  - There are *many locations* where a task can run efficiently
    - Blocks replicated across nodes, multiple tasks/node

- Generalizes beyond MapReduce and fair sharing
  - Applies to any queuing policy that gives ordered job list
    - Used in Hadoop Fair Scheduler to implement hierarchical policy
  - Applies to placement needs other than data locality

Thanks!

- This work is open source, packaged with Hadoop as the Hadoop Fair Scheduler
  - Old version without delay scheduling in use at Facebook since fall 2008
  - Delay scheduling will appear in Hadoop 0.21

- My email: matei@eecs.berkeley.edu

Related Work

- Quincy (SOSP '09)
  - Locality aware fair scheduler for Dryad
  - Expresses scheduling problem as min-cost flow
  - Kills tasks to reassign nodes when jobs change
  - One task slot per machine

- HPC scheduling
  - Inelastic jobs (cannot scale up/down over time)
  - Data locality generally not considered

- Grid computing
  - Focused on common interfaces for accessing compute resources in multiple administrative domains

Delay Scheduling Details

when there is a free task slot on node n:

    sort *jobs* according to queuing policy

    for j in jobs:

        if j has node-local task t on n:

            j.level := 0; *j.wait* := 0; **return** t

        else if j has rack-local task t on n and (*j.level* $\geq$ 1 or *j.wait* $\geq T_1$)**:**

            j.level := 1; *j.wait* := 0; **return** t

        else if *j.level* = 2 or (*j.level* = 1 and *j.wait* $\geq T_2$)

                or (*j.level* = 0 and *j.wait* $\geq T_1 + T_2$)**:**

            j.level := 2; *j.wait* := 0; **return** t

        else:

            j.wait += time since last scheduling decision