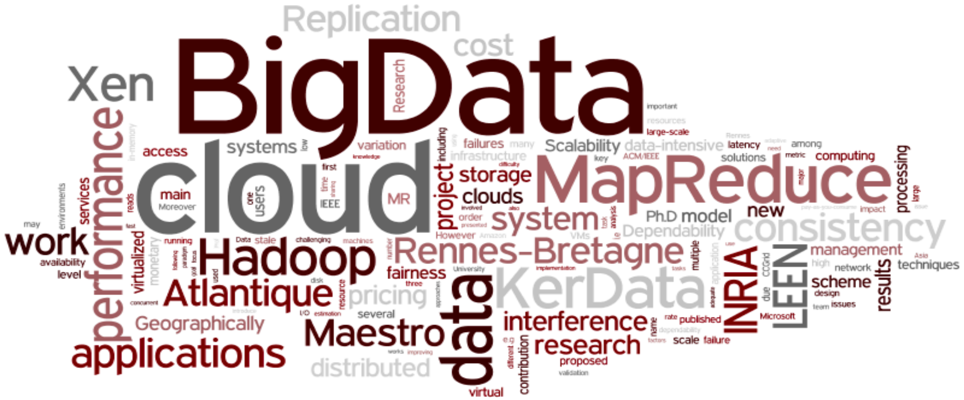
Hadoop System



Shadi Ibrahim

Inria, Rennes - Bretagne Atlantique Research Center

Batch Big data Processing

Adapted from Presentations from:
http://wiki.apache.org/hadoop/HadoopPresentations

HDFS Architecture: NameNode (1)

Master-Slave Architecture
HDFS Master "NameNode"

- Manages all file system metadata in memory
    - List of files
    - For each file name, a set of blocks
    - For each block, a set of DataNodes
    - File attributes (creation time, replication factor)
- Controls read/write access to files
- Manages block replication
- Transaction log: register file creation, deletion, etc.

http://wiki.apache.org/hadoop/HadoopPresentations

HDFS Architecture: DataNodes (2)

HDFS Slaves "DataNodes"

A DataNode is a block server

- Stores data in the local file system (e.g. ext3)
- Stores meta-data of a block (e.g. CRC)
- Serves data and meta-data to Clients

Block Report

- Periodically sends a report of all existing blocks to the NameNode

Pipelining of Data

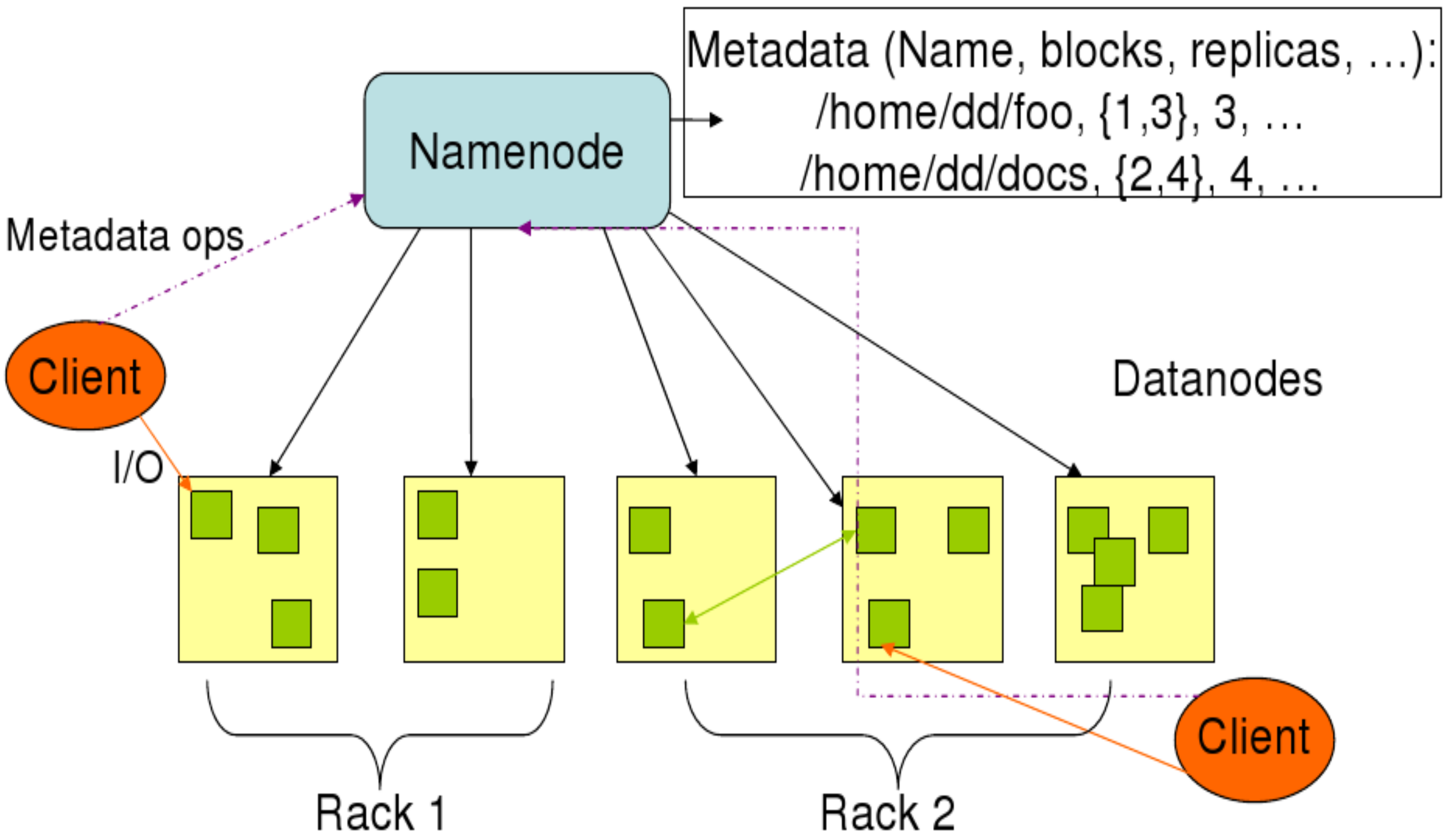
- Forwards data to other specified DataNodes

Perform replication tasks upon instruction by NameNode

Rack-aware

http://wiki.apache.org/hadoop/HadoopPresentations

HDFS Architecture (3)



http://wiki.apache.org/hadoop/HadoopPresentations

Fault Tolerance in HDFS

DataNodes send heartbeats to the NameNode

- Once every 3 seconds

NameNode uses heartbeats to detect DataNode failures

- Chooses new DataNodes for new replicas
- Balances disk usage
- Balances communication traffic to DataNodes

http://wiki.apache.org/hadoop/HadoopPresentations

Data Pipelining

- Client retrieves a list of DataNodes on which to place replicas of a block
  - Client writes block to the first DataNode
  - The first DataNode forwards the data to the next
  - The second DataNode forwards the data to the next
- DataNode in the Pipeline
  - When all replicas are written, the client moves on to write the next block in file

http://wiki.apache.org/hadoop/HadoopPresentations

Hadoop MapReduce

Master-Slave architecture

- Map-Reduce Master "JobTracker"
  - Accepts MR jobs submitted by users
  - Assigns Map and Reduce tasks to TaskTrackers
  - Monitors task and TaskTracker status, re-executes tasks upon failure

http://wiki.apache.org/hadoop/HadoopPresentations

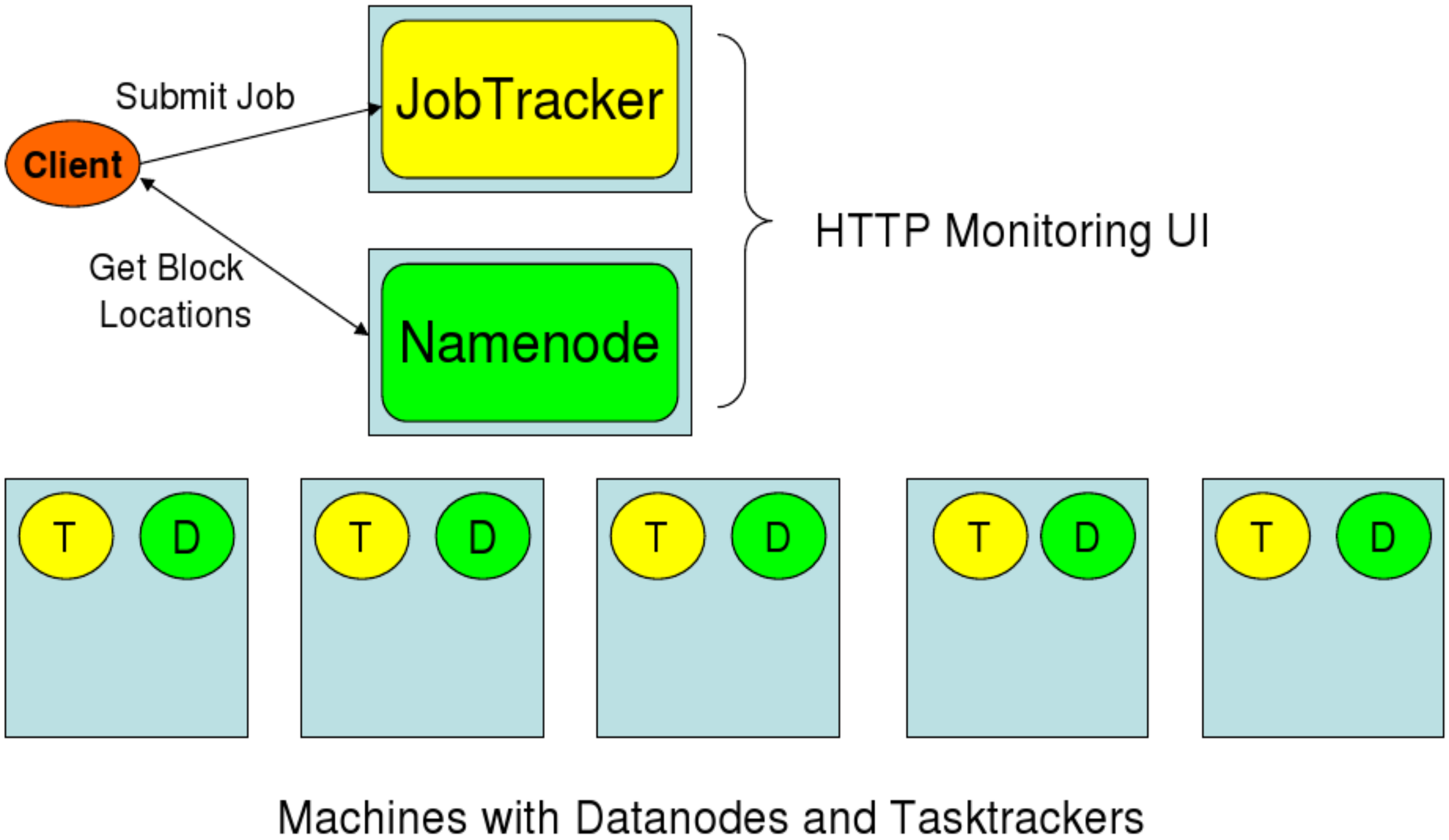
Hadoop MapReduce

Master-Slave architecture

- Map-Reduce Slaves “TaskTrackers”
  - Run Map and Reduce tasks upon instruction from the JobTracker
- Manage storage and transmission of intermediate output

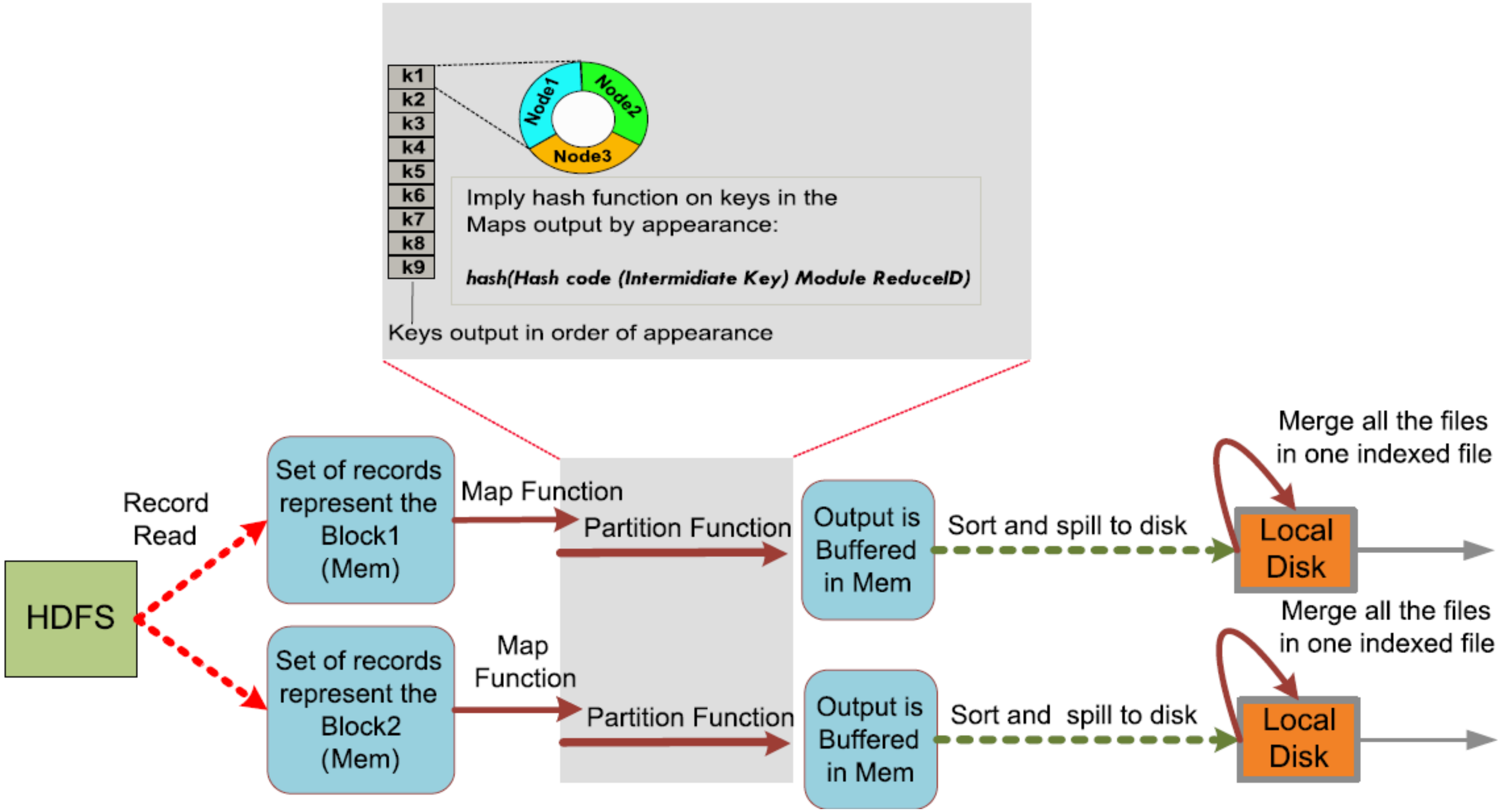
http://wiki.apache.org/hadoop/HadoopPresentations

Deployment: HDFS + MR



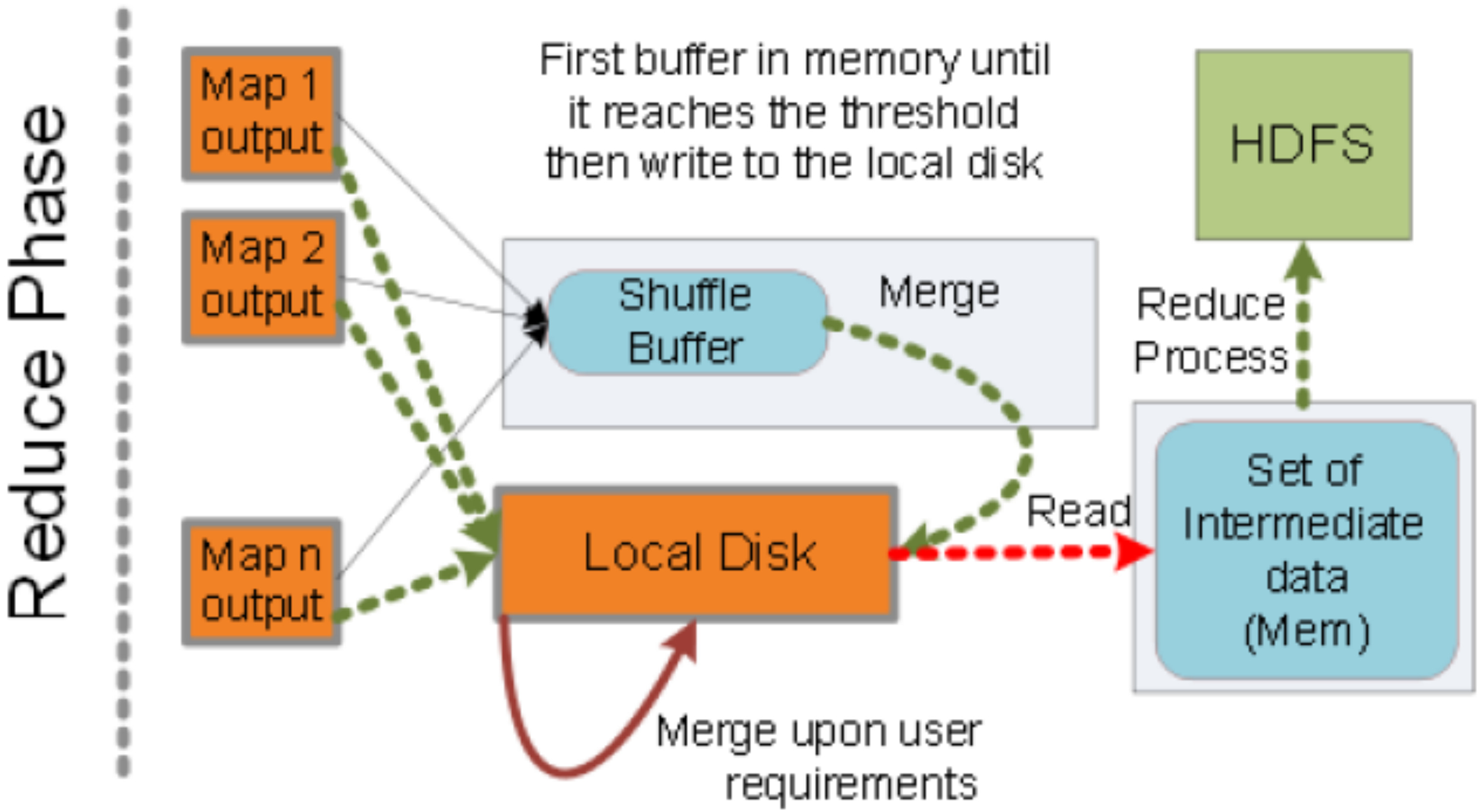
http://wiki.apache.org/hadoop/HadoopPresentations

Zoom on Map Phase



"Handling partitioning skew in MapReduce using LEEN" S Ibrahim, H Jin, L Lu, B He, G Antoniu, S Wu - Peer-to-Peer Networking and Applications, 2013

Zoom on Reduce Phase



Data Locality

Data Locality is exposed in the Map Task scheduling

Data are Replicated:

- Fault tolerance
- Performance : divide the work among nodes

Job Tracker schedules map tasks considering:

- Node-aware
- Rack-aware
- non-local map Tasks

Fault-tolerance

TaskTrackers send heartbeats to the Job Tracker

» Once every 3 seconds

TaskTracker uses heartbeats to detect

» Node is labled as failed If no heartbeat is recieved for a defined expiry time (Defualt : 10 Minutes)

Re-execute all the ongoing and complete tasks

Need to develop a more efficient policy to prevent re-executing completed tasks (storing this data in HDFS)

Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

  Other jobs consuming resources on machine

  Bad disks with soft errors transfer data very slowly

  Weird things: processor caches disabled (!!)

Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

  Other jobs consuming resources on machine

  Bad disks with soft errors transfer data very slowly

  Weird things: processor caches disabled (!!)

Node 1

Node 2

Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

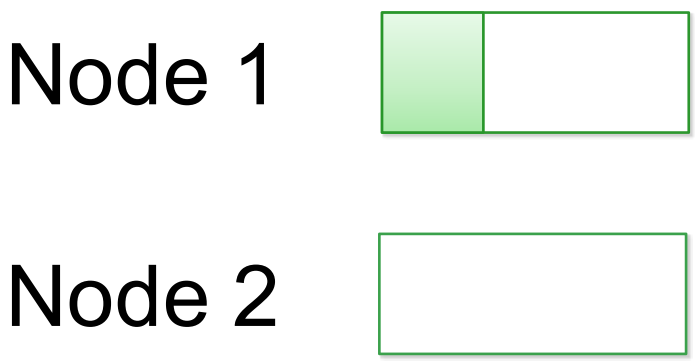
Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

  Other jobs consuming resources on machine

  Bad disks with soft errors transfer data very slowly

  Weird things: processor caches disabled (!!)



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

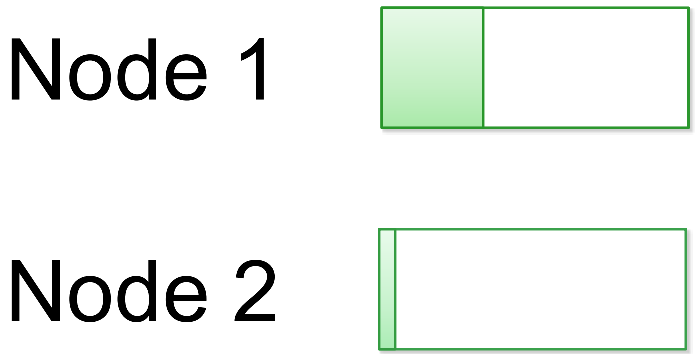
Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

  Other jobs consuming resources on machine

  Bad disks with soft errors transfer data very slowly

  Weird things: processor caches disabled (!!)



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

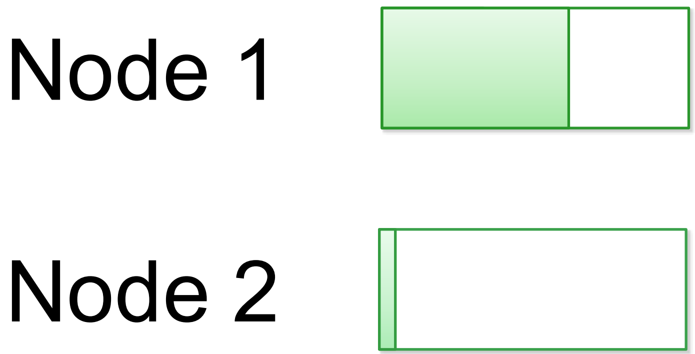
Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

Other jobs consuming resources on machine

Bad disks with soft errors transfer data very slowly

Weird things: processor caches disabled (!!)



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

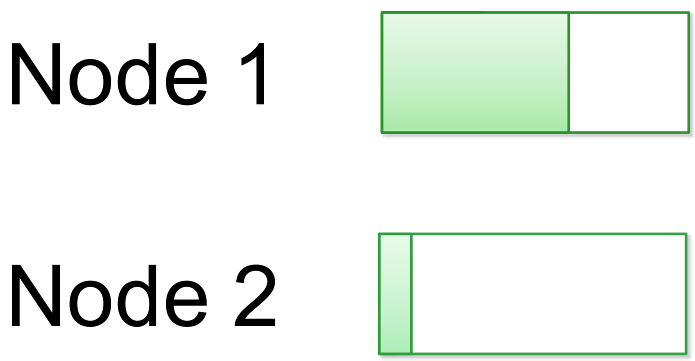
Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

  Other jobs consuming resources on machine

  Bad disks with soft errors transfer data very slowly

  Weird things: processor caches disabled (!!)



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

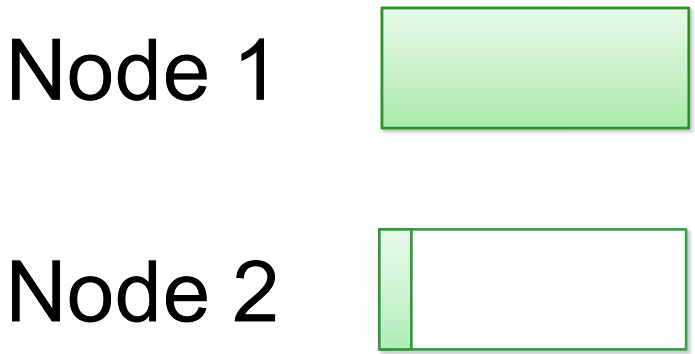
Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

  Other jobs consuming resources on machine

  Bad disks with soft errors transfer data very slowly

  Weird things: processor caches disabled (!!)



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

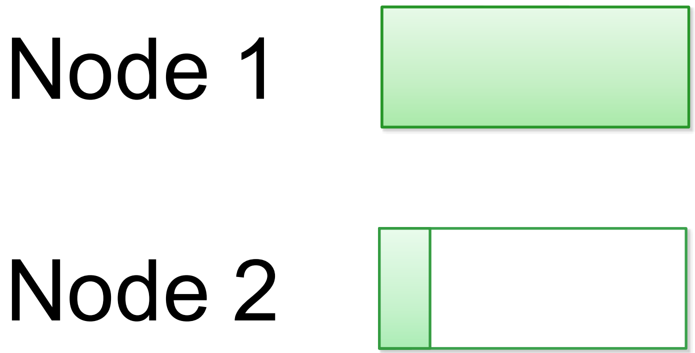
Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

  Other jobs consuming resources on machine

  Bad disks with soft errors transfer data very slowly

  Weird things: processor caches disabled (!!)



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

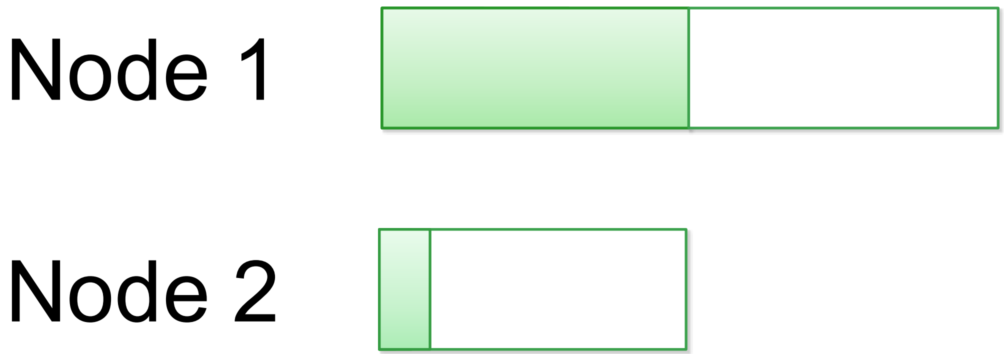
Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

Other jobs consuming resources on machine

Bad disks with soft errors transfer data very slowly

Weird things: processor caches disabled (!!)



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

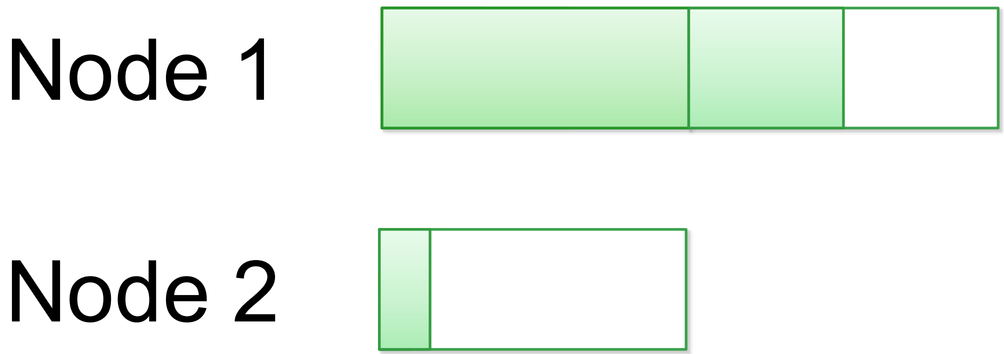
Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

Other jobs consuming resources on machine

Bad disks with soft errors transfer data very slowly

Weird things: processor caches disabled (!!)



Adopted from a presentation by Matei Zaharia “Improving MapReduce Performance in Heterogeneous Environments”, OSDI 2008, San Diego, CA, December 2008.

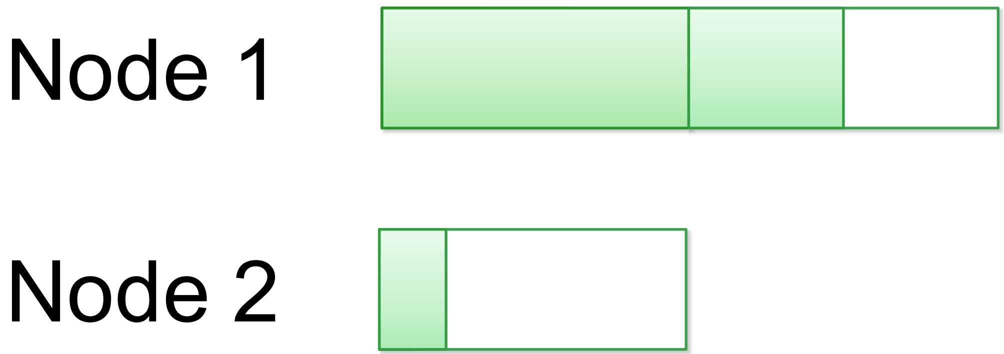
Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

Other jobs consuming resources on machine

Bad disks with soft errors transfer data very slowly

Weird things: processor caches disabled (!!)



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

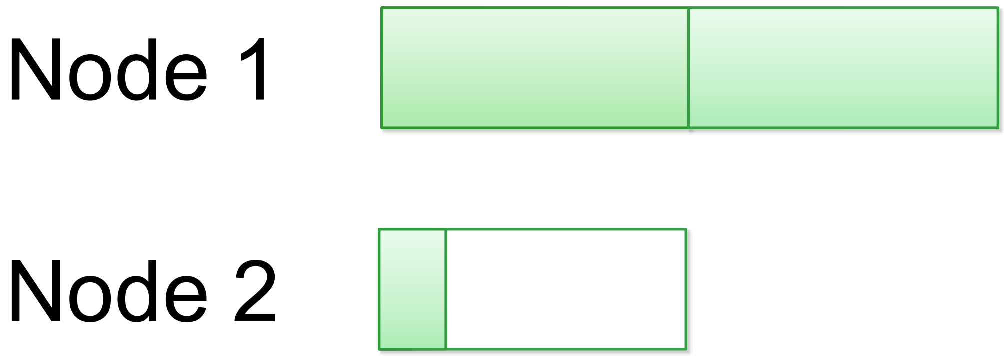
Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

  Other jobs consuming resources on machine

  Bad disks with soft errors transfer data very slowly

  Weird things: processor caches disabled (!!)



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

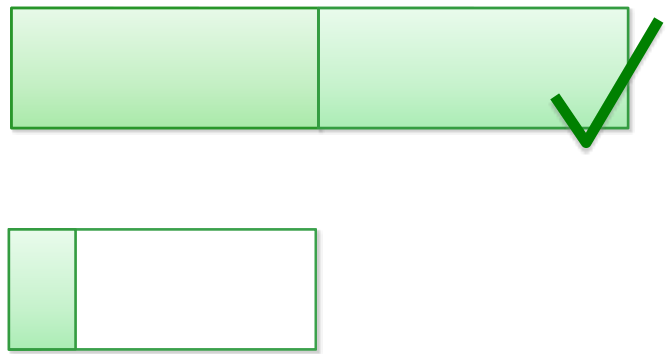
Other jobs consuming resources on machine

Bad disks with soft errors transfer data very slowly

Weird things: processor caches disabled (!!)

Node 1

Node 2



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

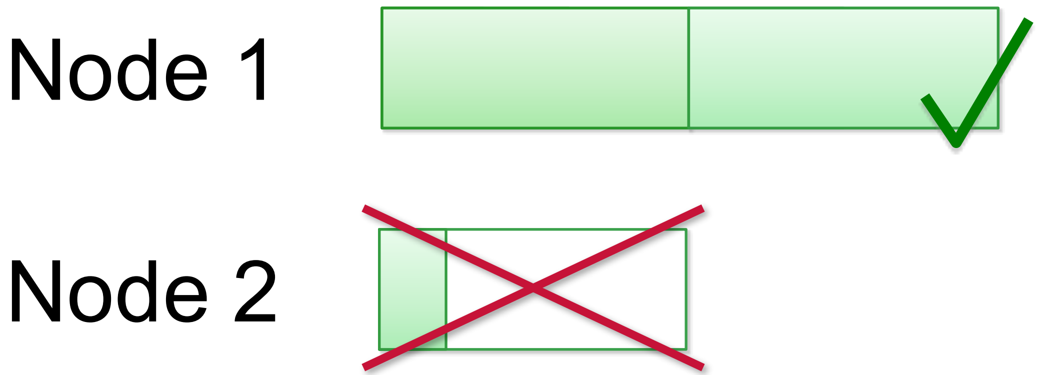
Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

Other jobs consuming resources on machine

Bad disks with soft errors transfer data very slowly

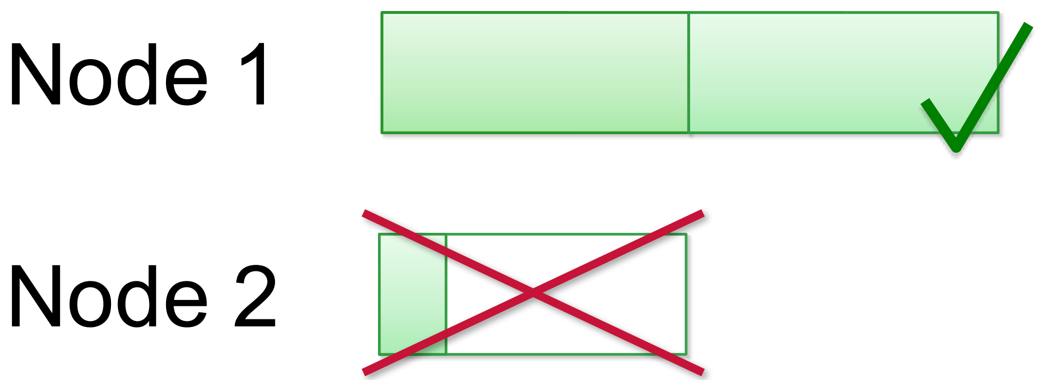
Weird things: processor caches disabled (!!)



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Speculation in Hadoop

- Nodes slow (stragglers) → run backup tasks

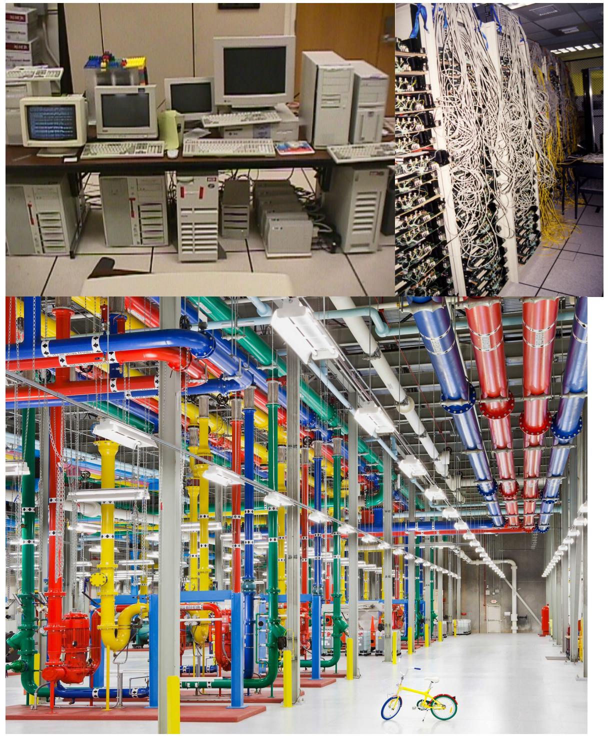


How to do this in heterogeneous environment?

Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Heterogeneity in Clouds

- Scale out – Heterogeneous Hardware
- Virtualizations
- Dynamic resource allocations



facebook

>2300 server

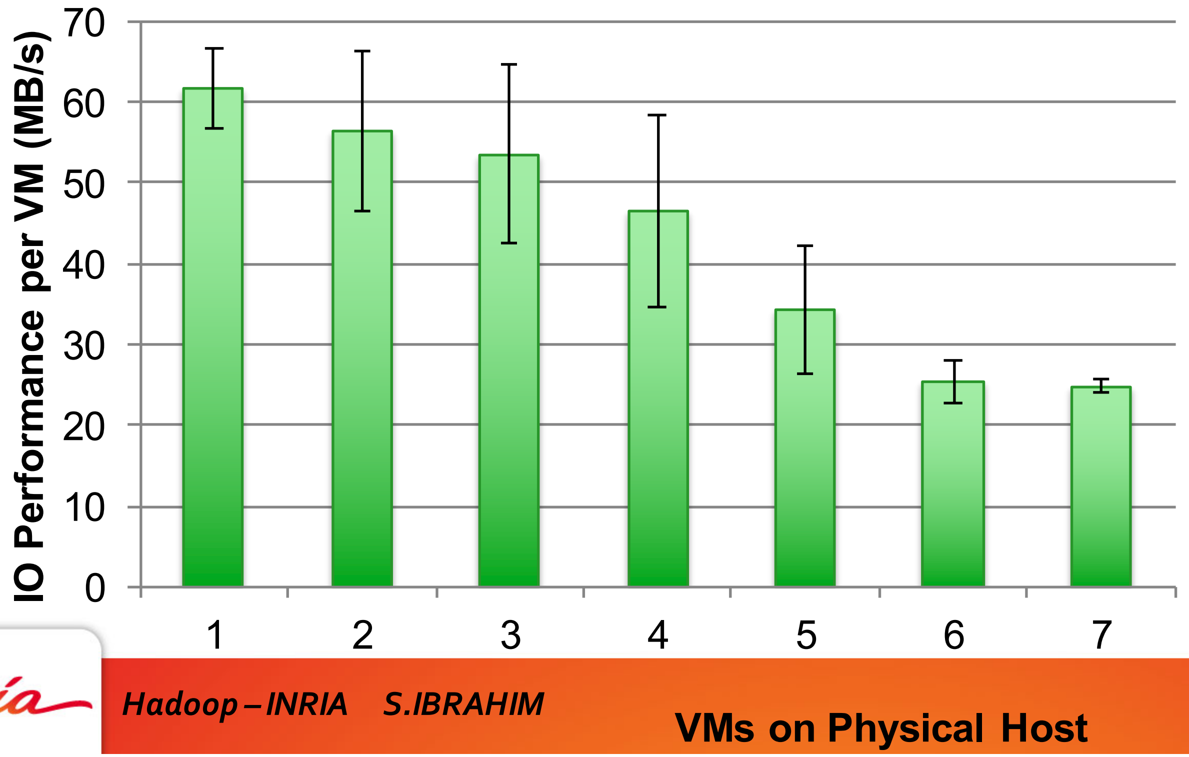
YAHOO!

>4300 server

[1] Lee et al., Heterogeneity-aware resource allocation and scheduling in the cloud, SoCC 2011

Heterogeneity in Virtualized Environments

- VM technology isolates CPU and memory, but disk and network are shared
  - Full bandwidth when no contention
  - Equal shares when there is contention
- **2.5x** performance difference



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Backup Tasks in Hadoop's Default Scheduler

- Start primary tasks, then look for backups to launch as nodes become free
- Tasks report "progress score" from 0 to 1
  - Launch backup if progress < avgProgress – 0.2

Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Problems in Heterogeneous Environment

Problems in Heterogeneous Environment

1. *Too many* backups, thrashing shared resources like network bandwidth

Problems in Heterogeneous Environment

1. *Too many* backups, thrashing shared resources like network bandwidth
2. *Wrong* tasks backed up

Problems in Heterogeneous Environment

1. *Too many* backups, thrashing shared resources like network bandwidth
2. *Wrong* tasks backed up
3. Backups may be placed on *slow nodes*

Problems in Heterogeneous Environment

1. *Too many* backups, thrashing shared resources like network bandwidth
2. *Wrong* tasks backed up
3. Backups may be placed on *slow nodes*
4. Breaks when tasks start at different times

Problems in Heterogeneous Environment

1. *Too many* backups, thrashing shared resources like network bandwidth
2. *Wrong* tasks backed up
3. Backups may be placed on *slow nodes*
4. Breaks when tasks start at different times

- Example: ~80% of reduces backed up, most losing to originals; network thrashed

Idea: Progress Rates

- Instead of using progress values, compute progress *rates*, and back up tasks that are "far enough" below the mean

Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

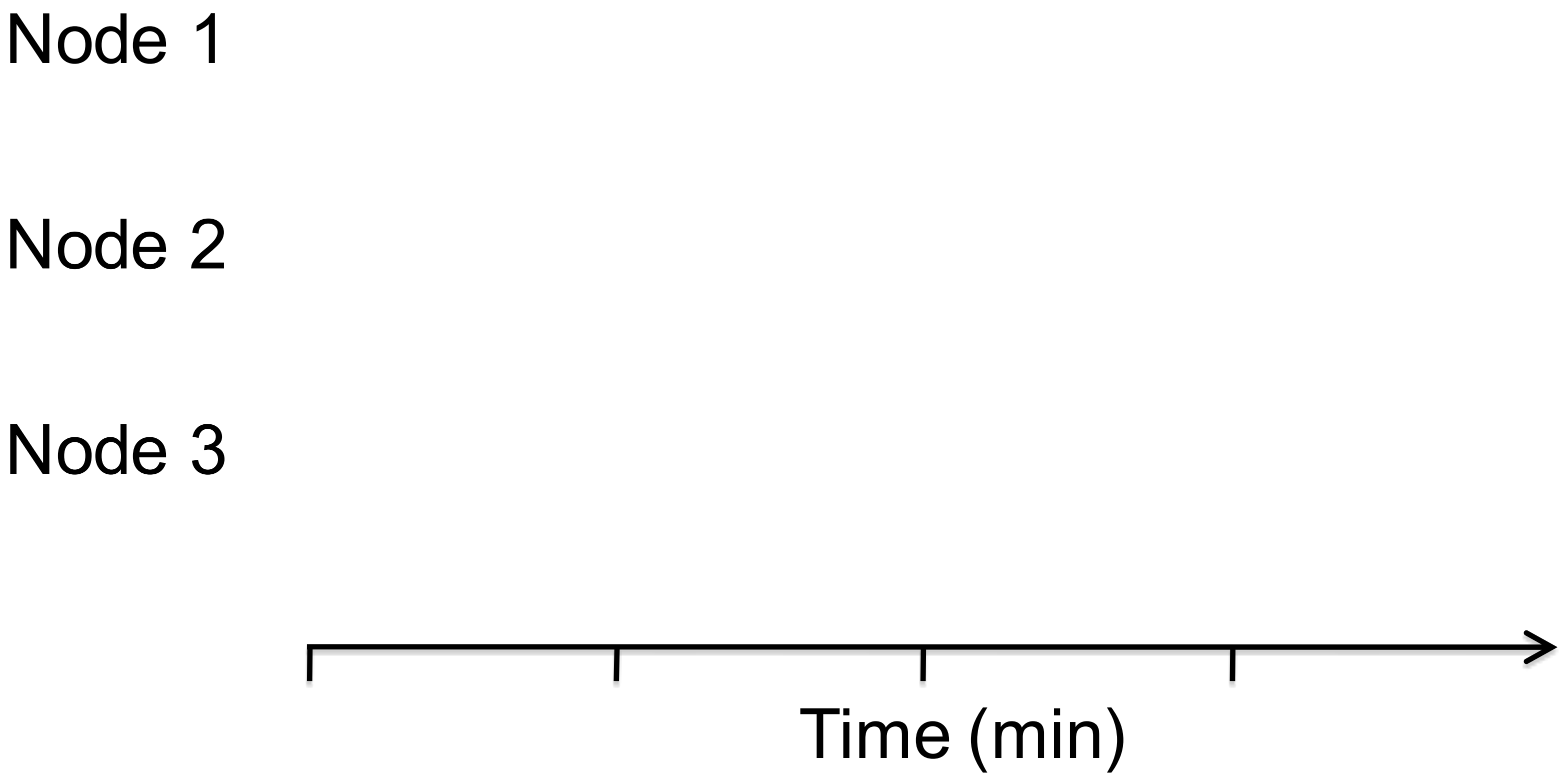
Idea: Progress Rates

- Instead of using progress values, compute progress *rates*, and back up tasks that are "far enough" below the mean

- **Problem:** can still select the wrong tasks

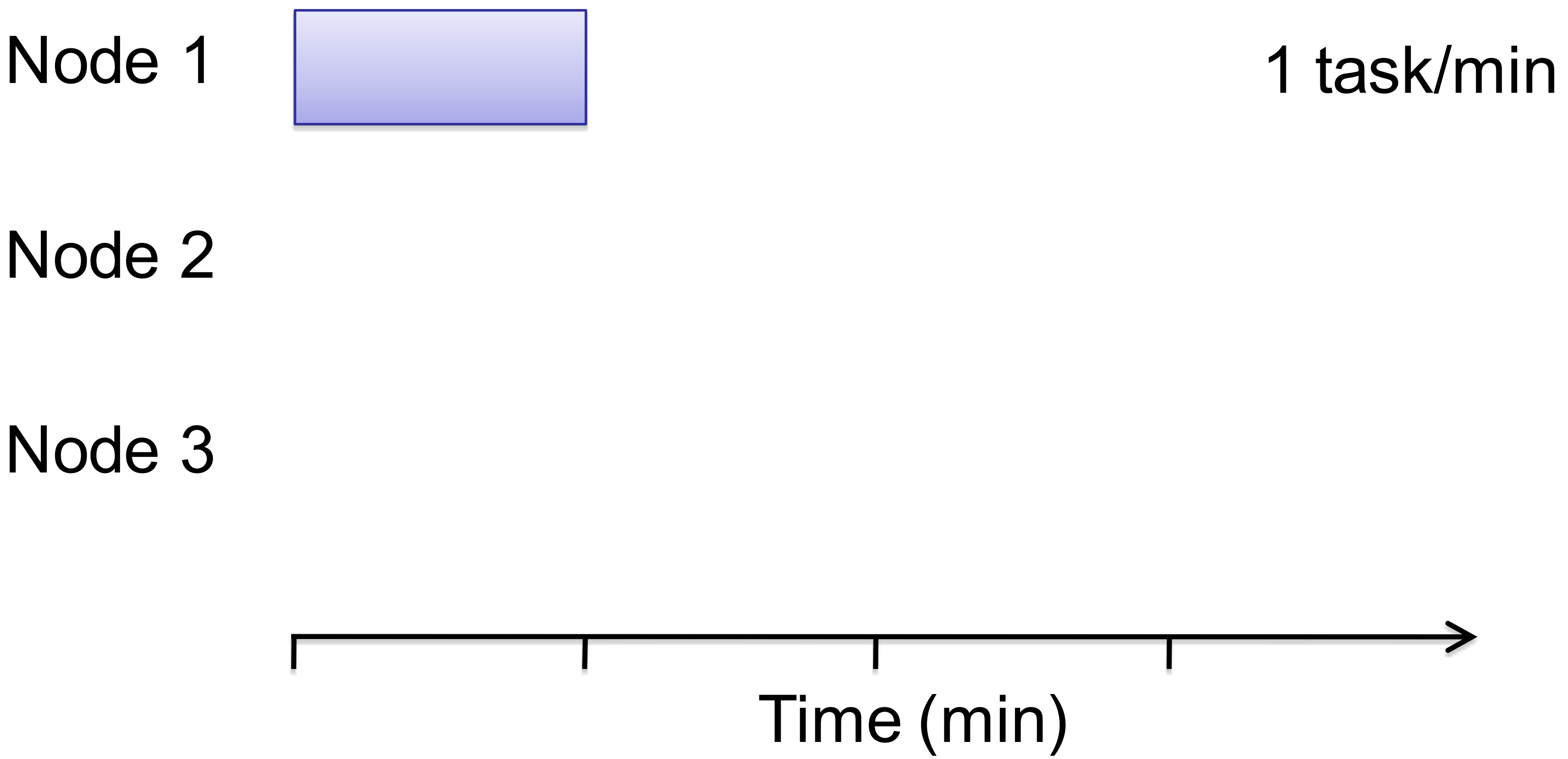
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example



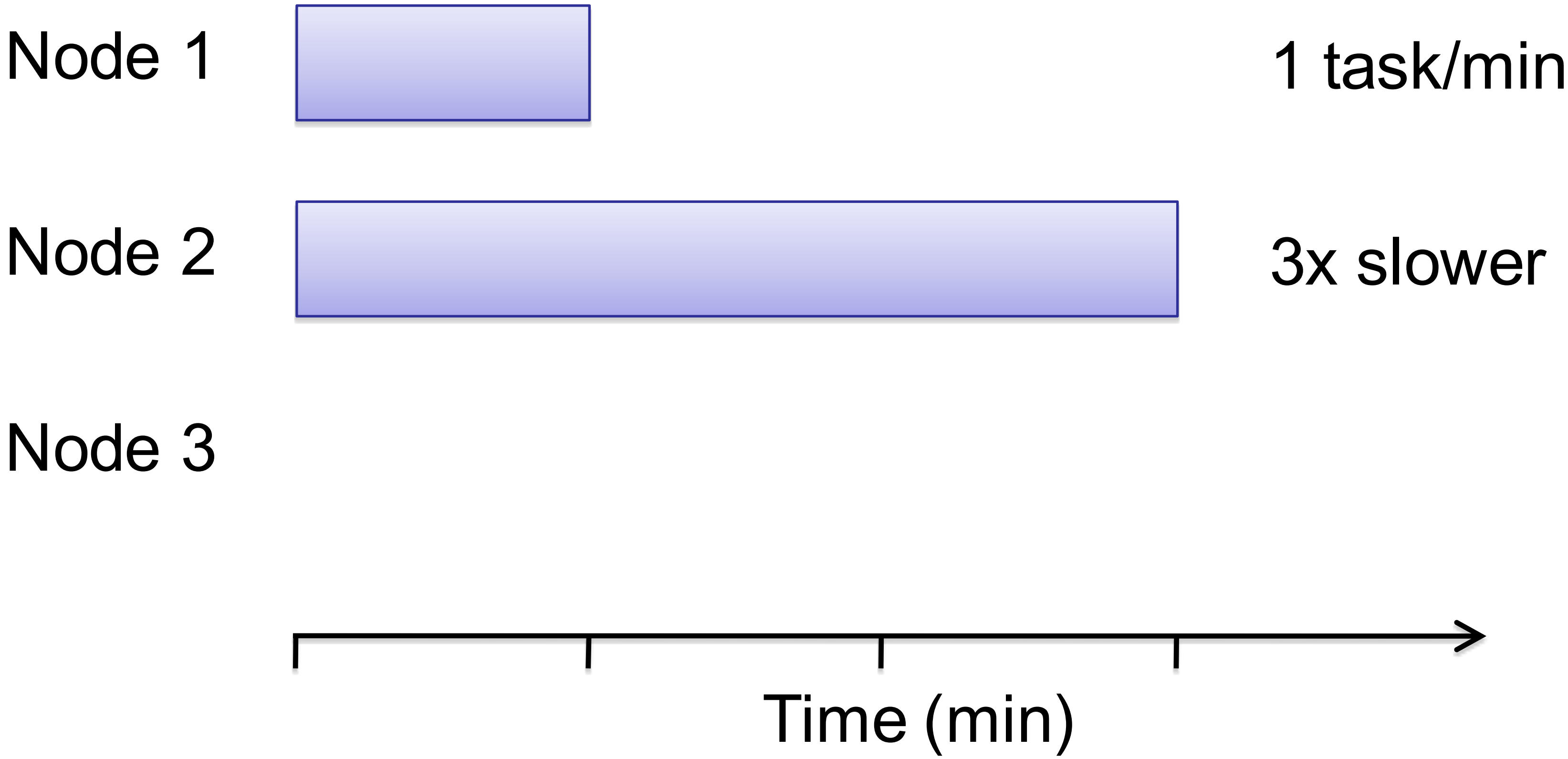
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example



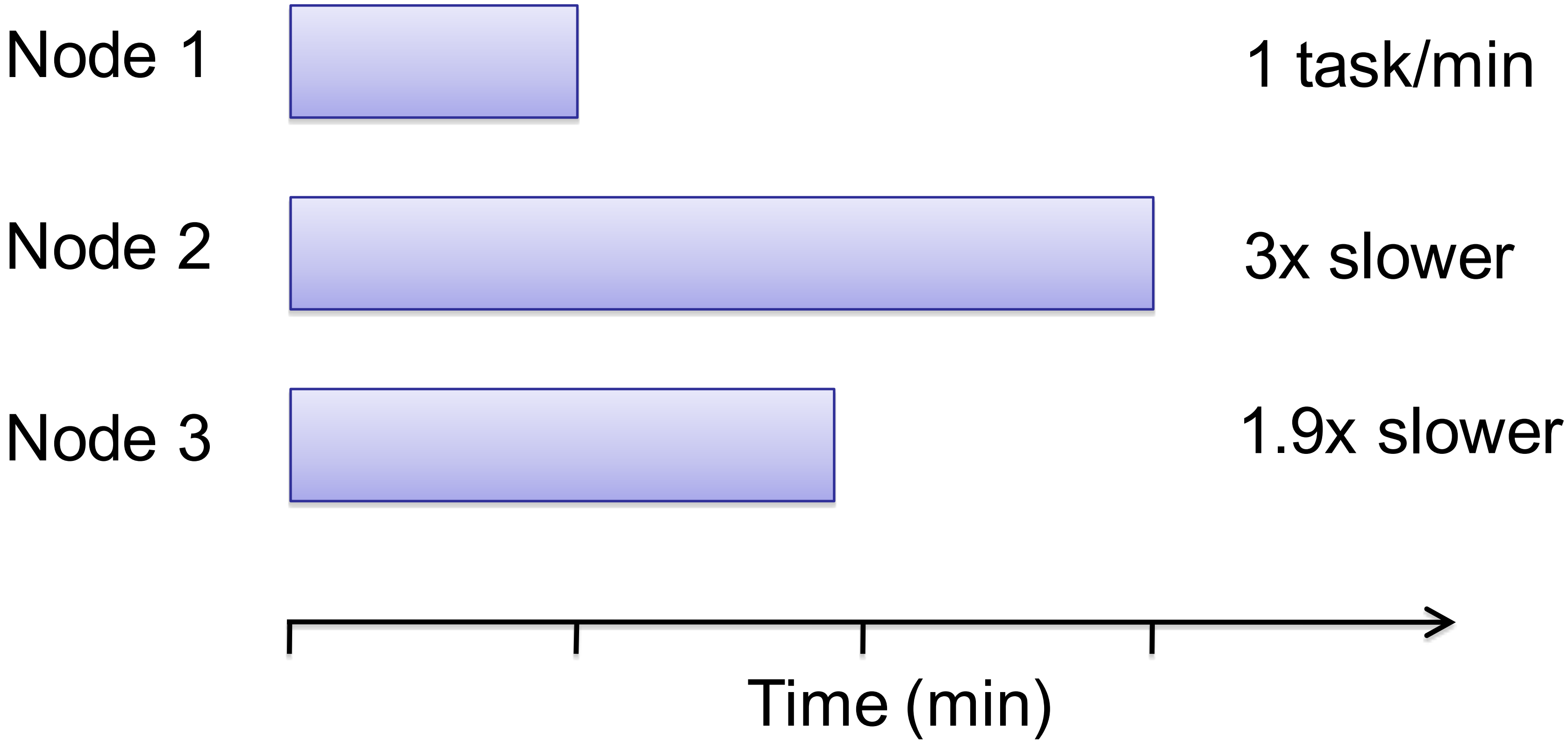
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example



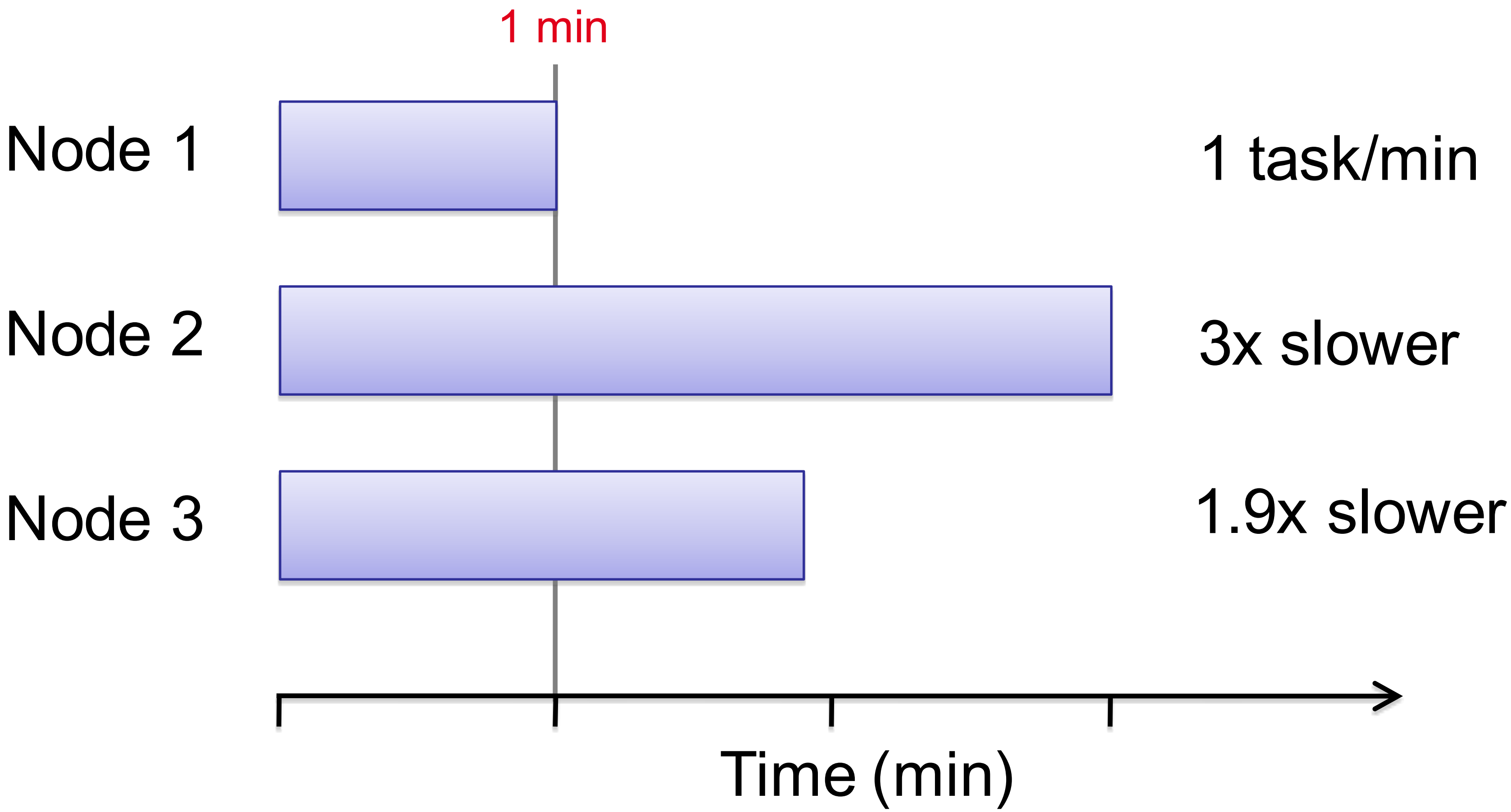
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example



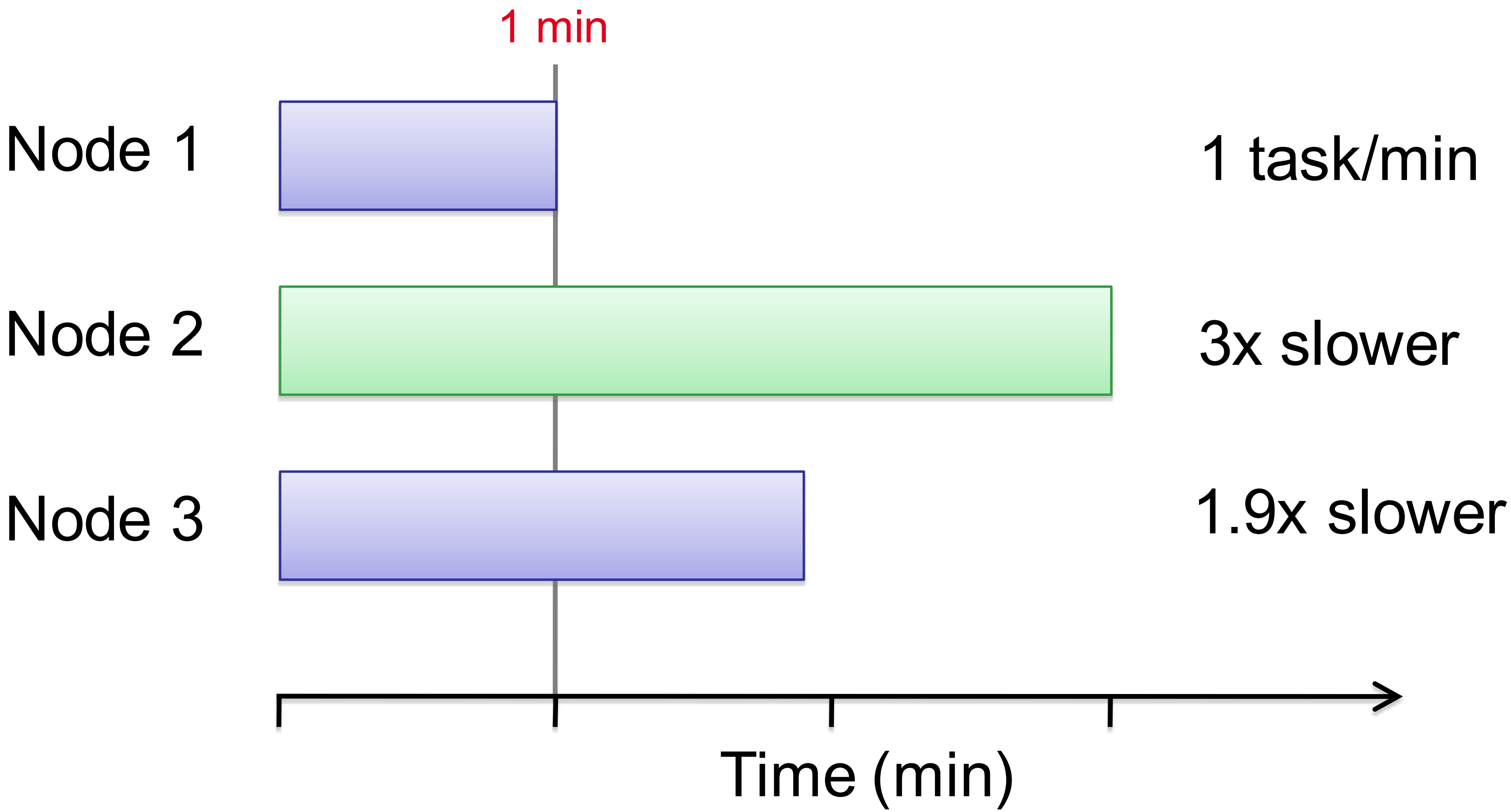
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example



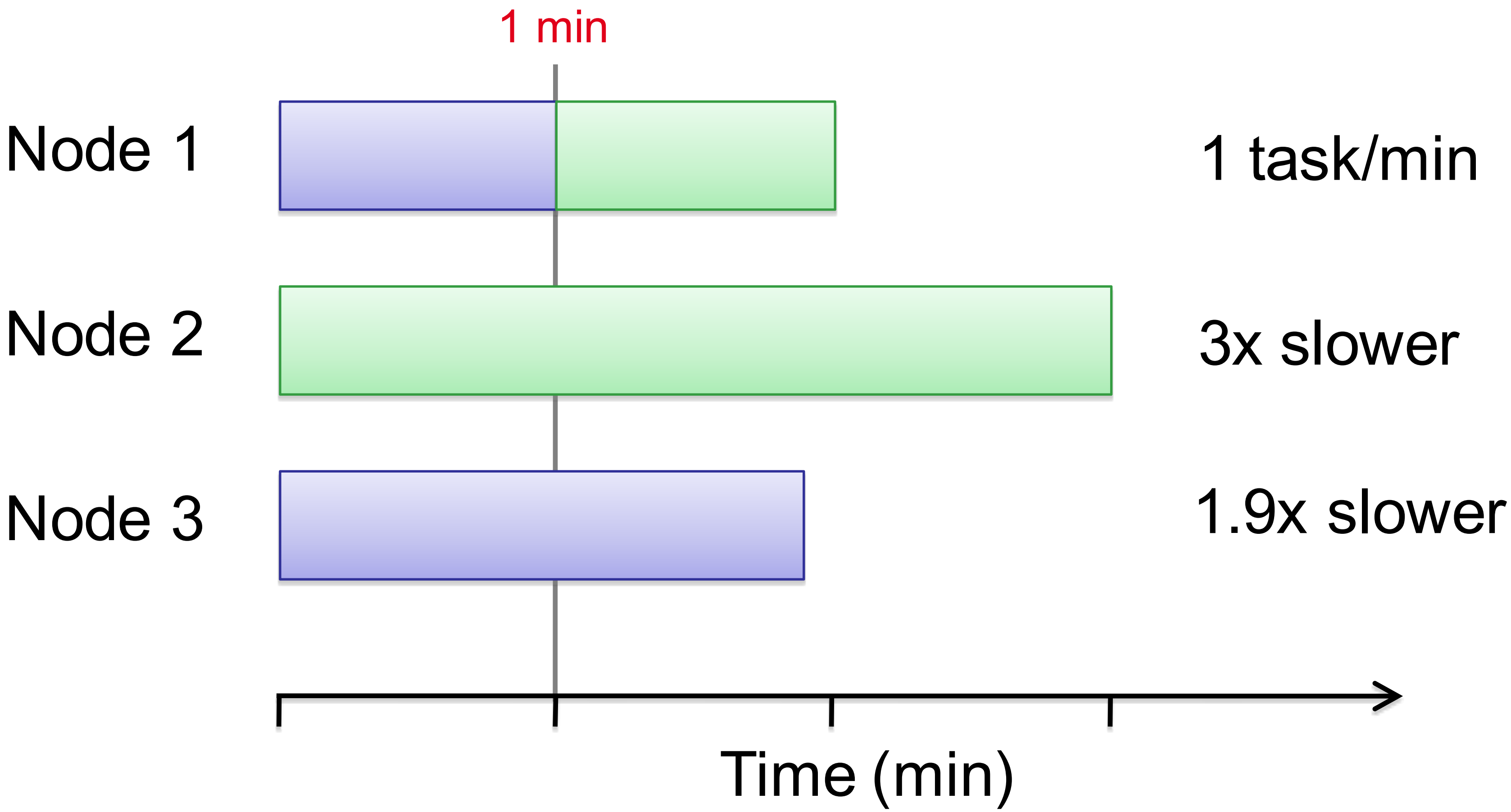
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example



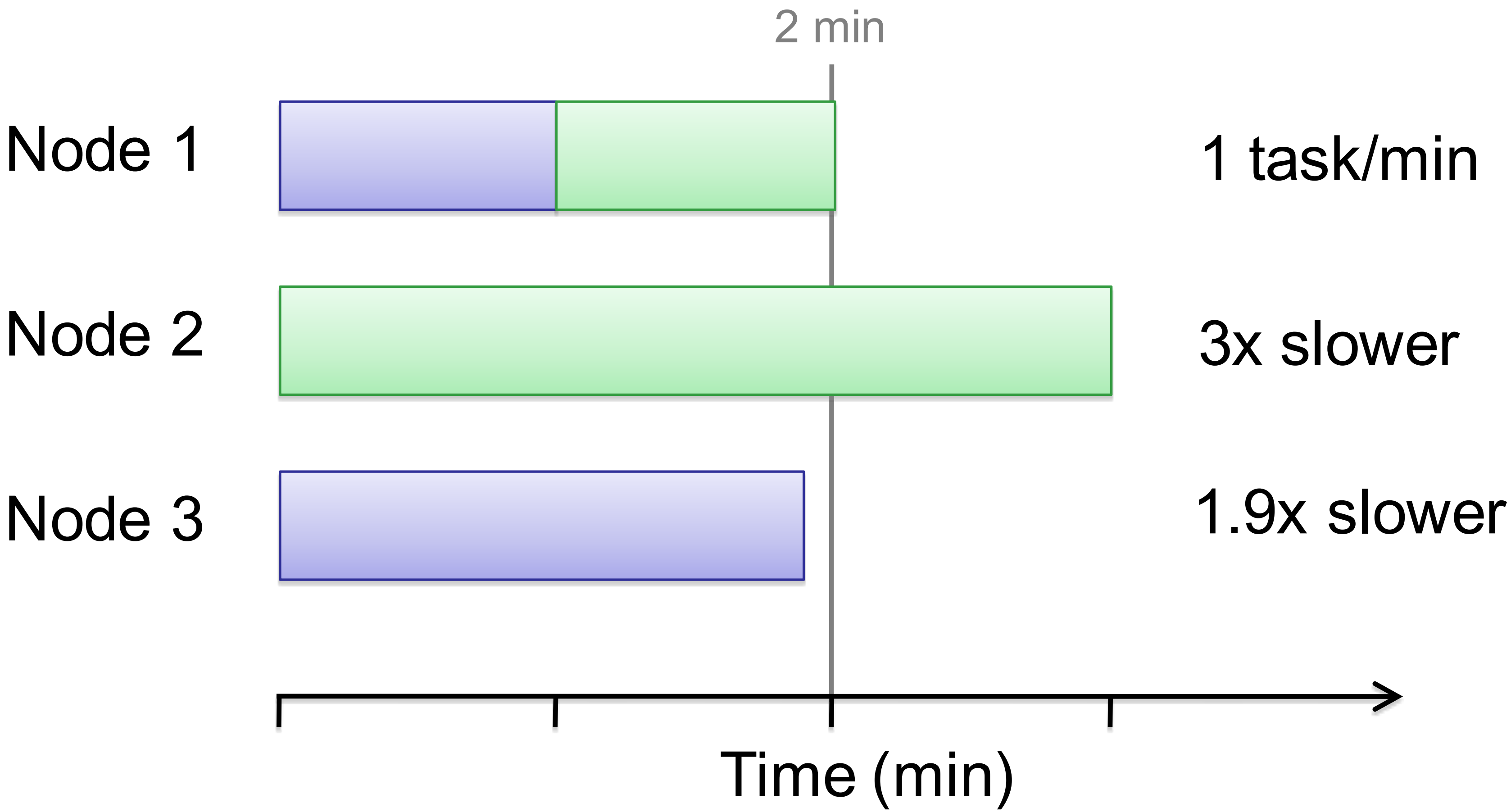
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example



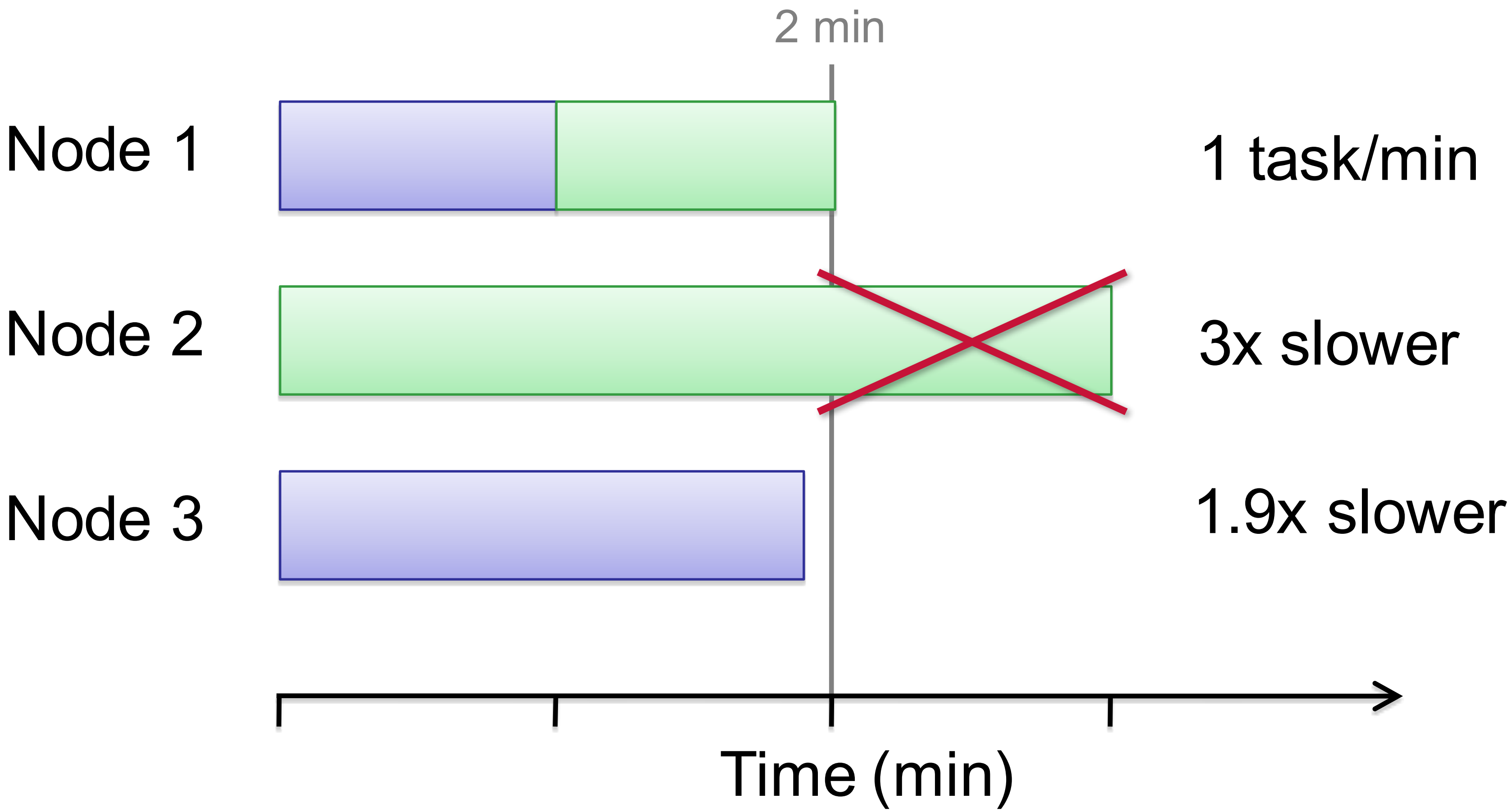
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example

What if the job had 5 tasks?

Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Node 1

Node 2

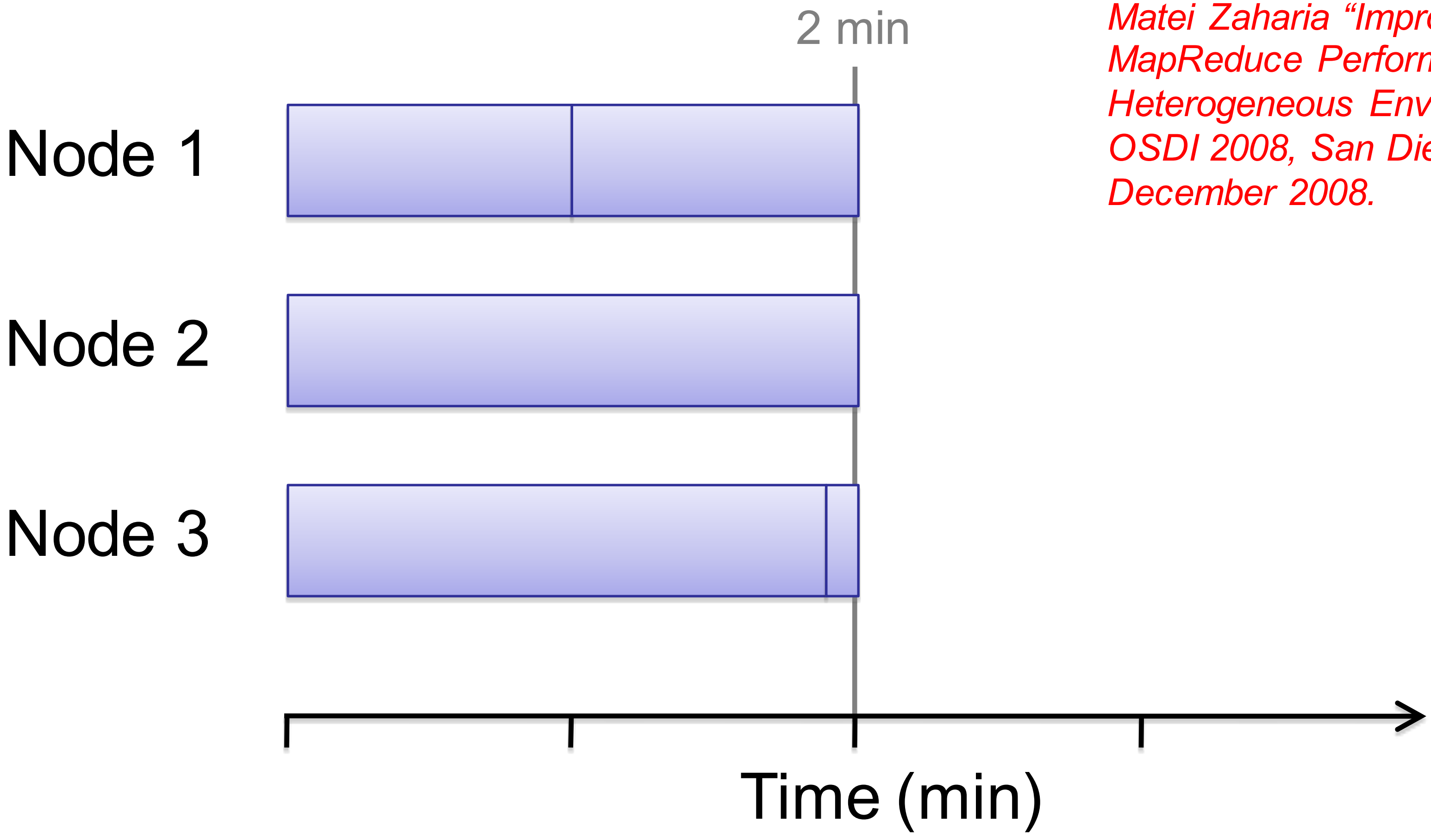
Node 3

Time (min)

Progress Rate Example

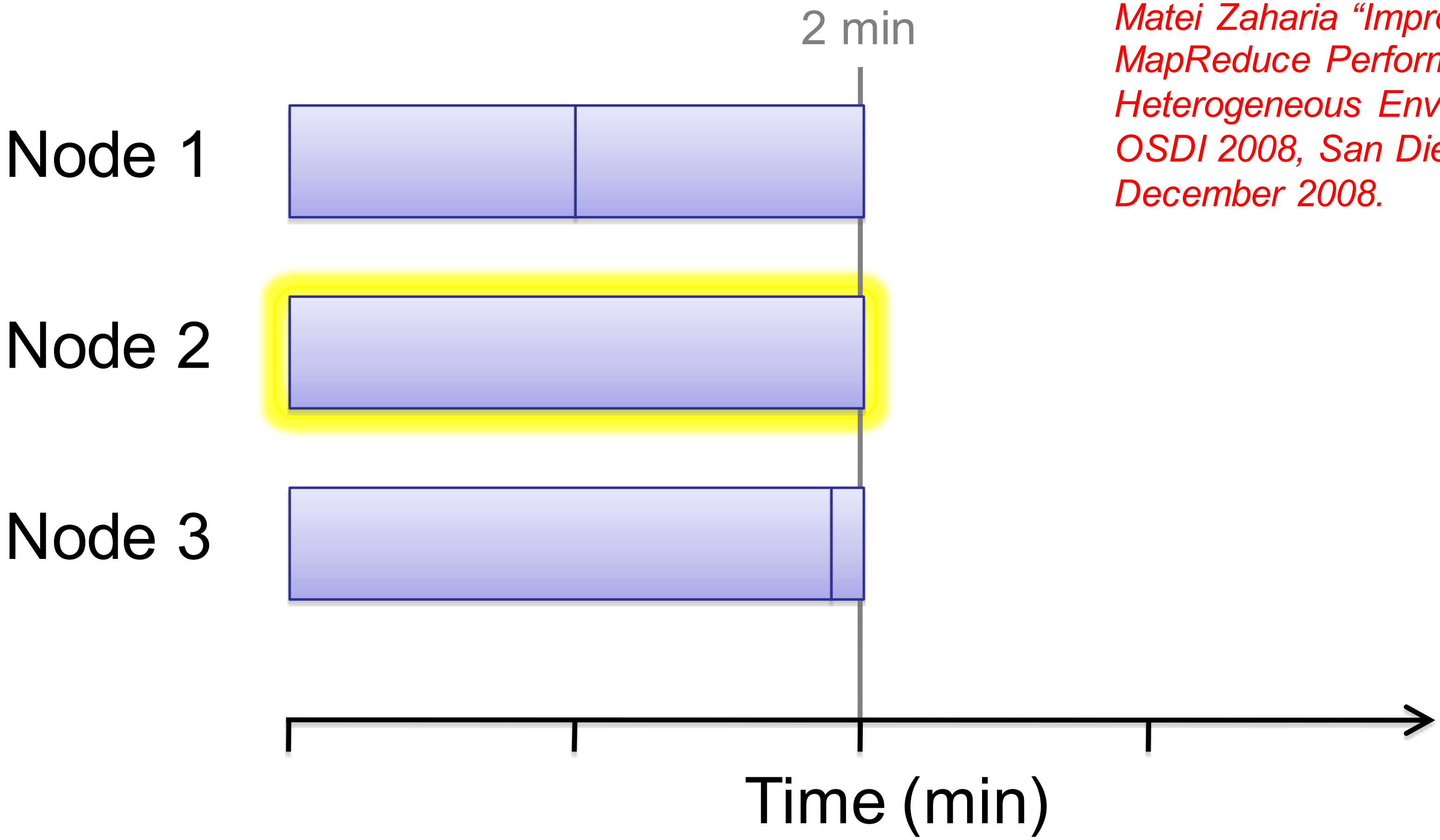
What if the job had 5 tasks?

Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.



Progress Rate Example

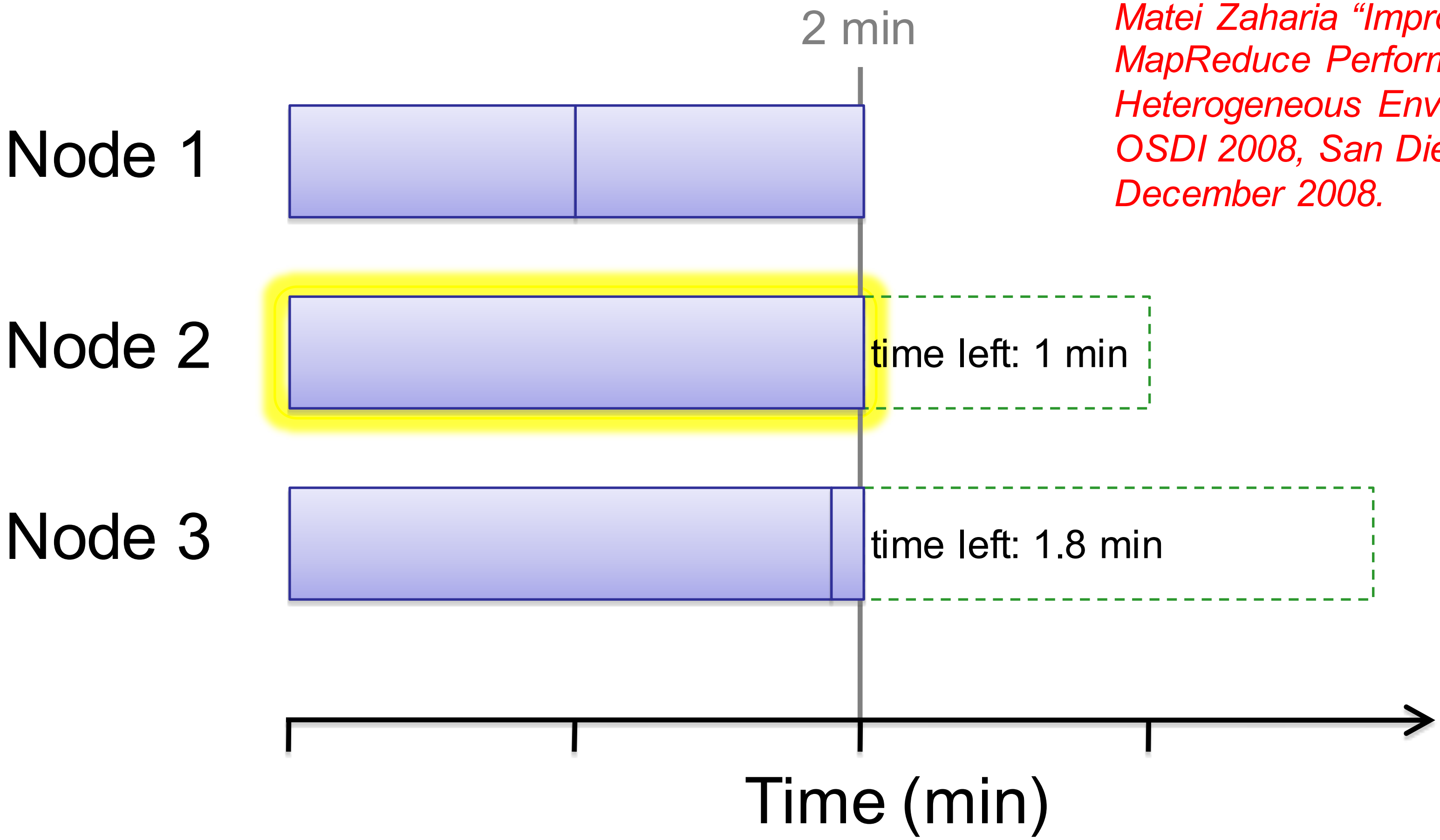
What if the job had 5 tasks?



Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example

What if the job had 5 tasks?

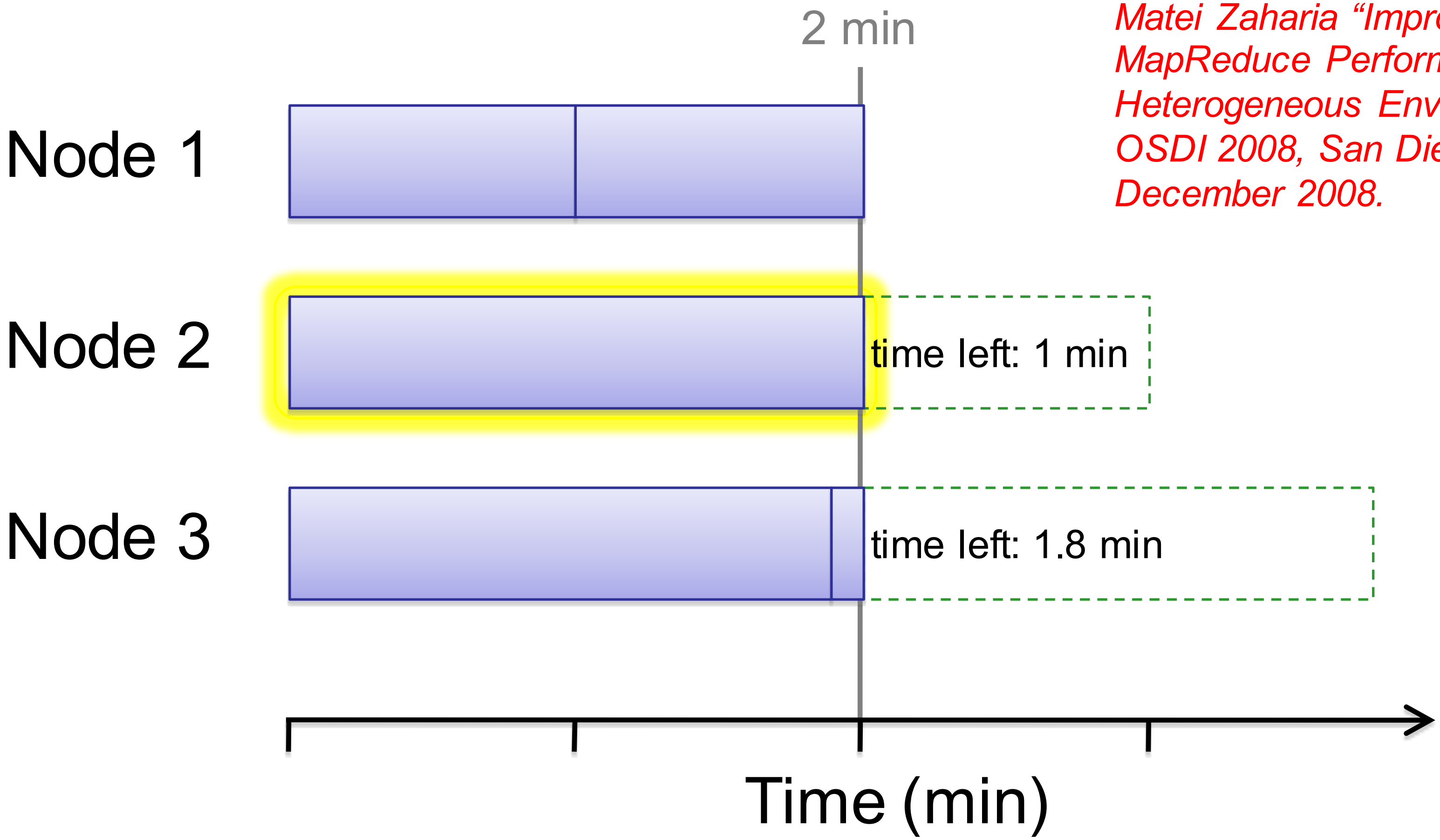


Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Progress Rate Example

What if the job had 5 tasks?

Adopted from a presentation by Matei Zaharia “Improving MapReduce Performance in Heterogeneous Environments”, OSDI 2008, San Diego, CA, December 2008.



Node 2 is slowest, but should back up Node 3’s task!

Scheduler: LATE

- Insight: back up the task with the largest *estimated finish time*
  - "Longest Approximate Time to End"
  - Look forward instead of looking backward
- Sanity thresholds:
  - *Cap* number of backup tasks
  - Launch backups on *fast nodes*
  - Only back up tasks that are *sufficiently slow*

Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

LATE Details

- Estimating finish times:

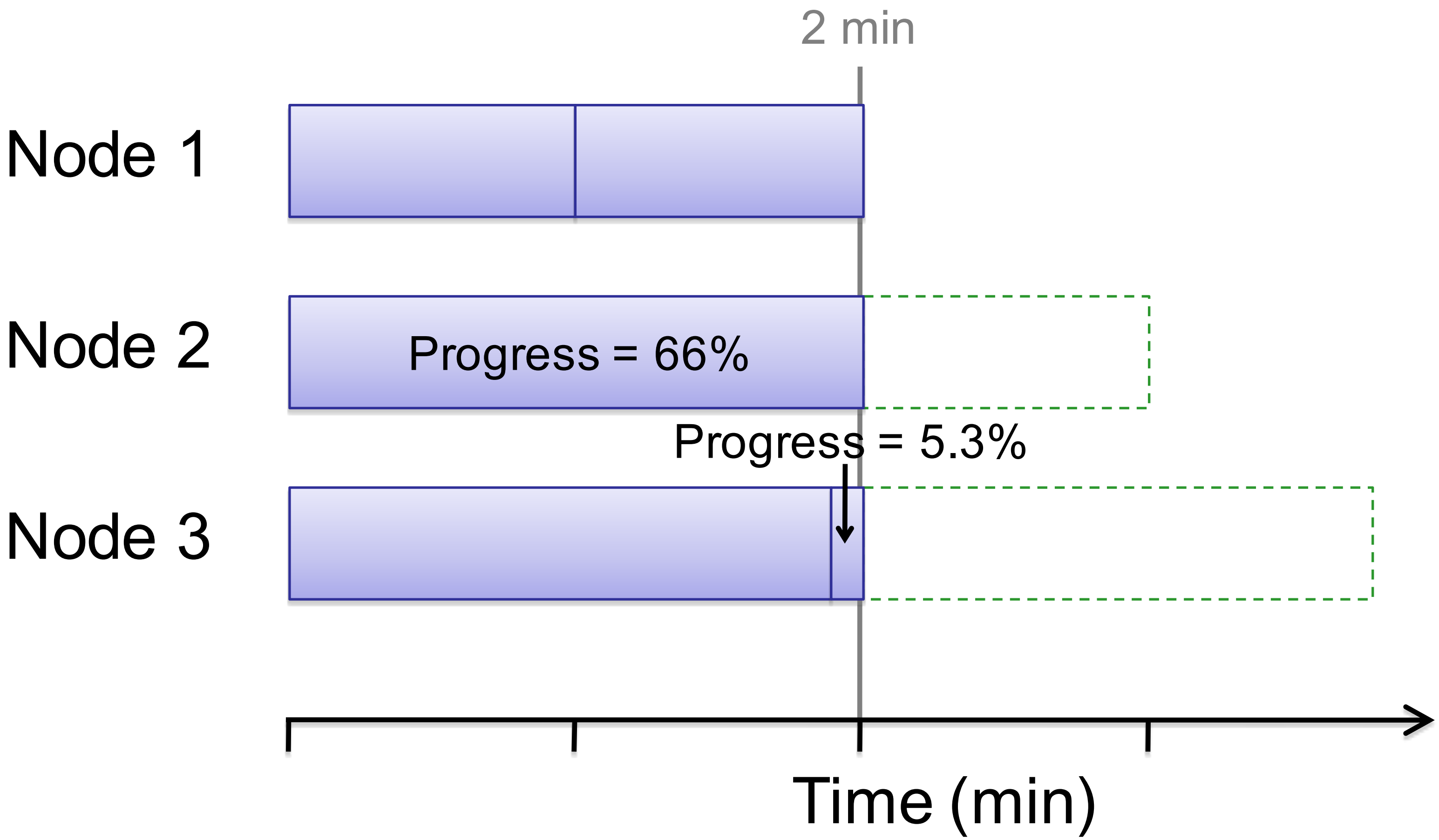
$$\textit{progress rate} = \frac{\text{progress score}}{\text{execution time}}$$

$$\textit{estimated time left} = \frac{1 - \text{progress score}}{\text{progress rate}}$$

- Threshold values:
  - 10% cap on backups, 25^{th} percentiles for slow node/task
  - Validated by sensitivity analysis

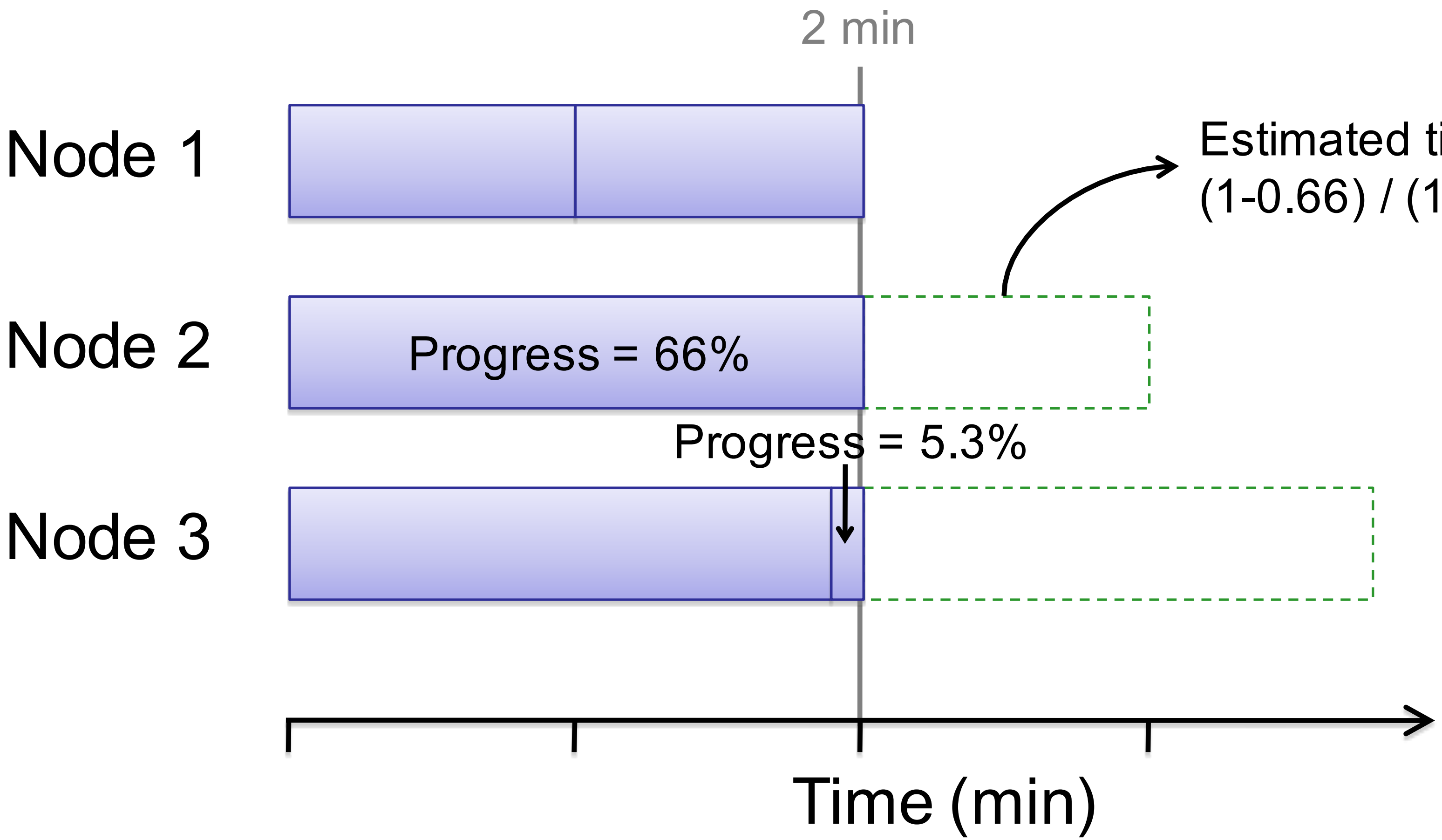
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

LATE Example



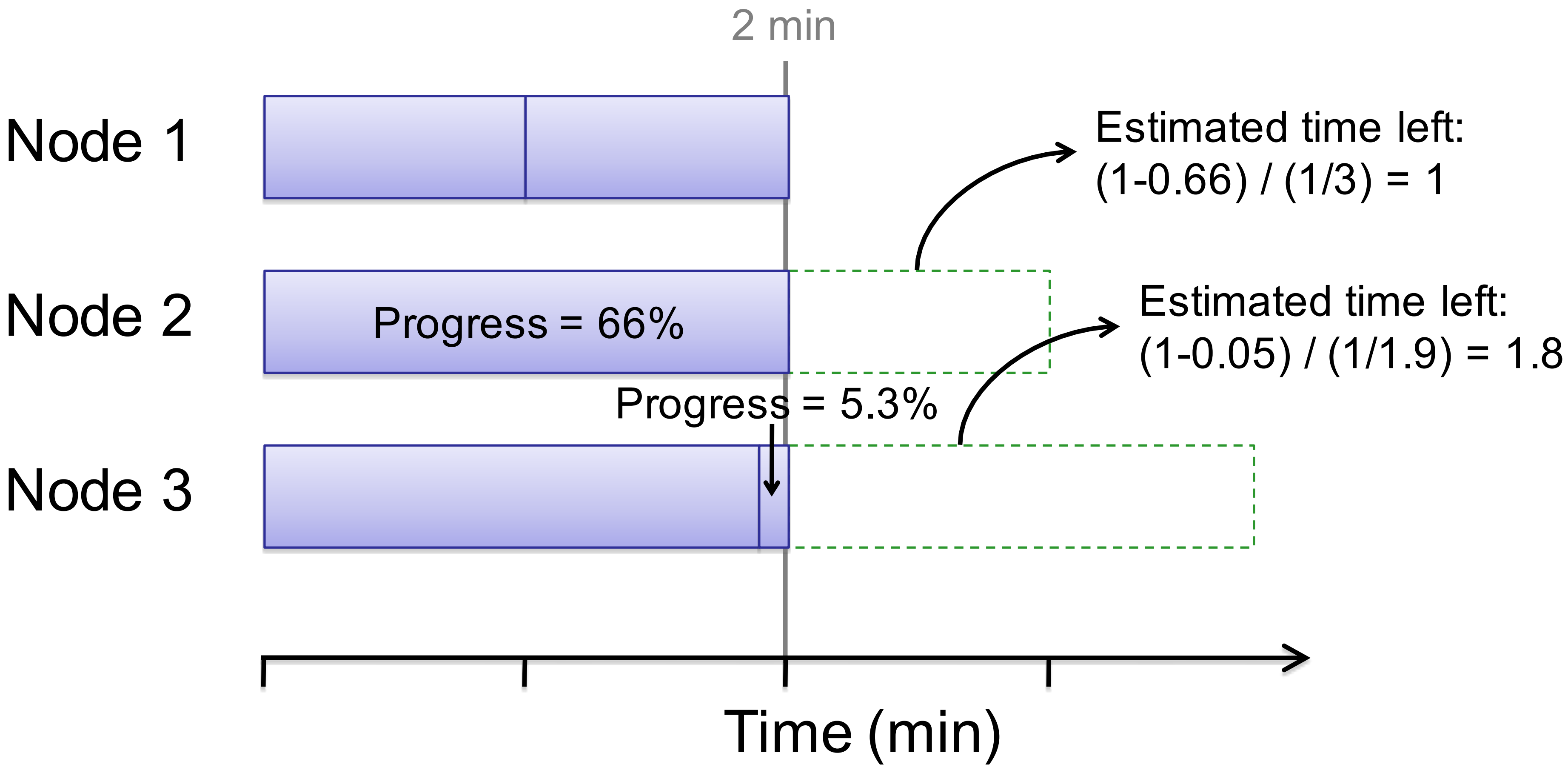
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

LATE Example



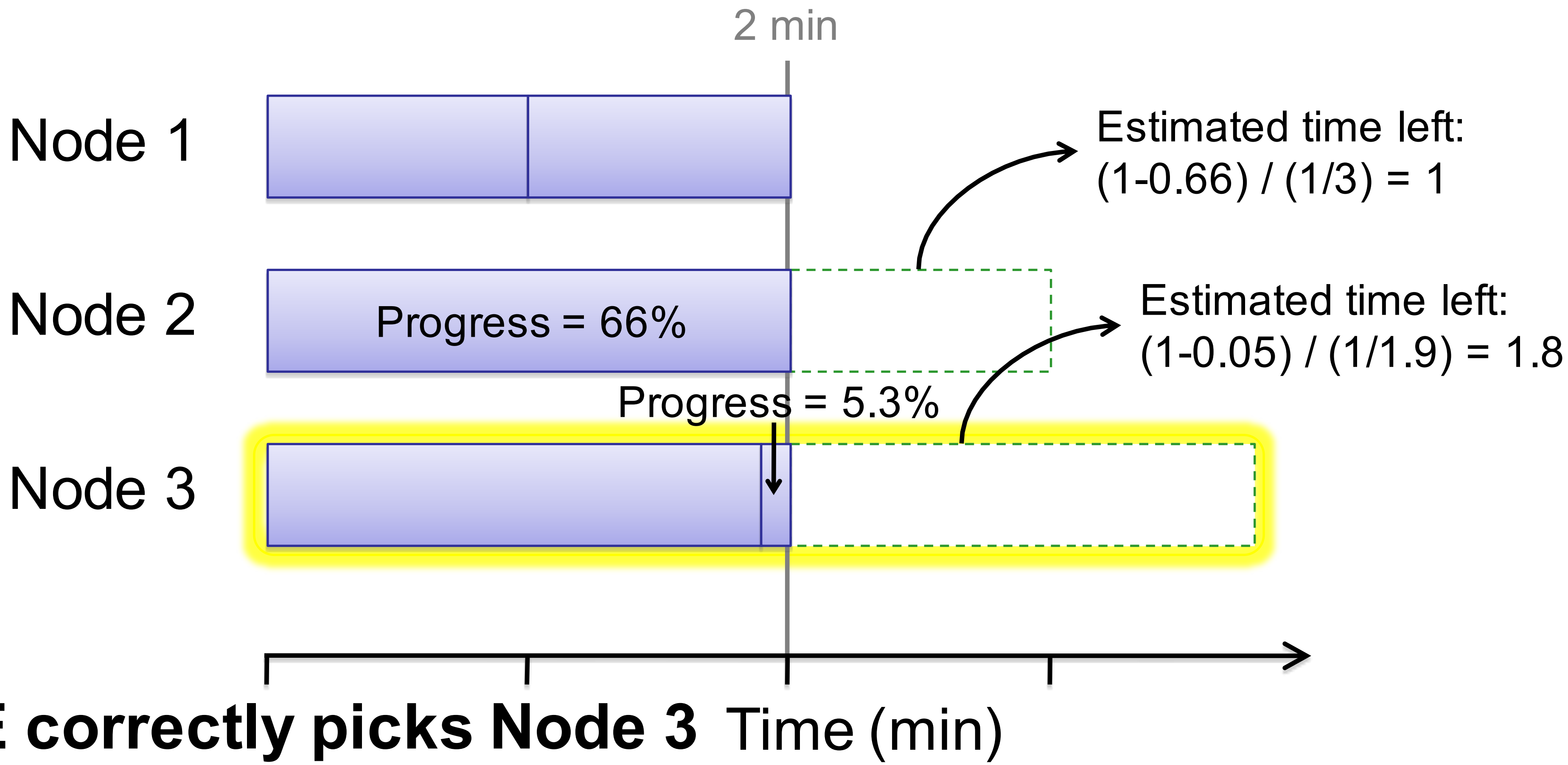
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

LATE Example



Adopted from a presentation by Matei Zaharia “Improving MapReduce Performance in Heterogeneous Environments”, OSDI 2008, San Diego, CA, December 2008.

LATE Example



LATE correctly picks Node 3

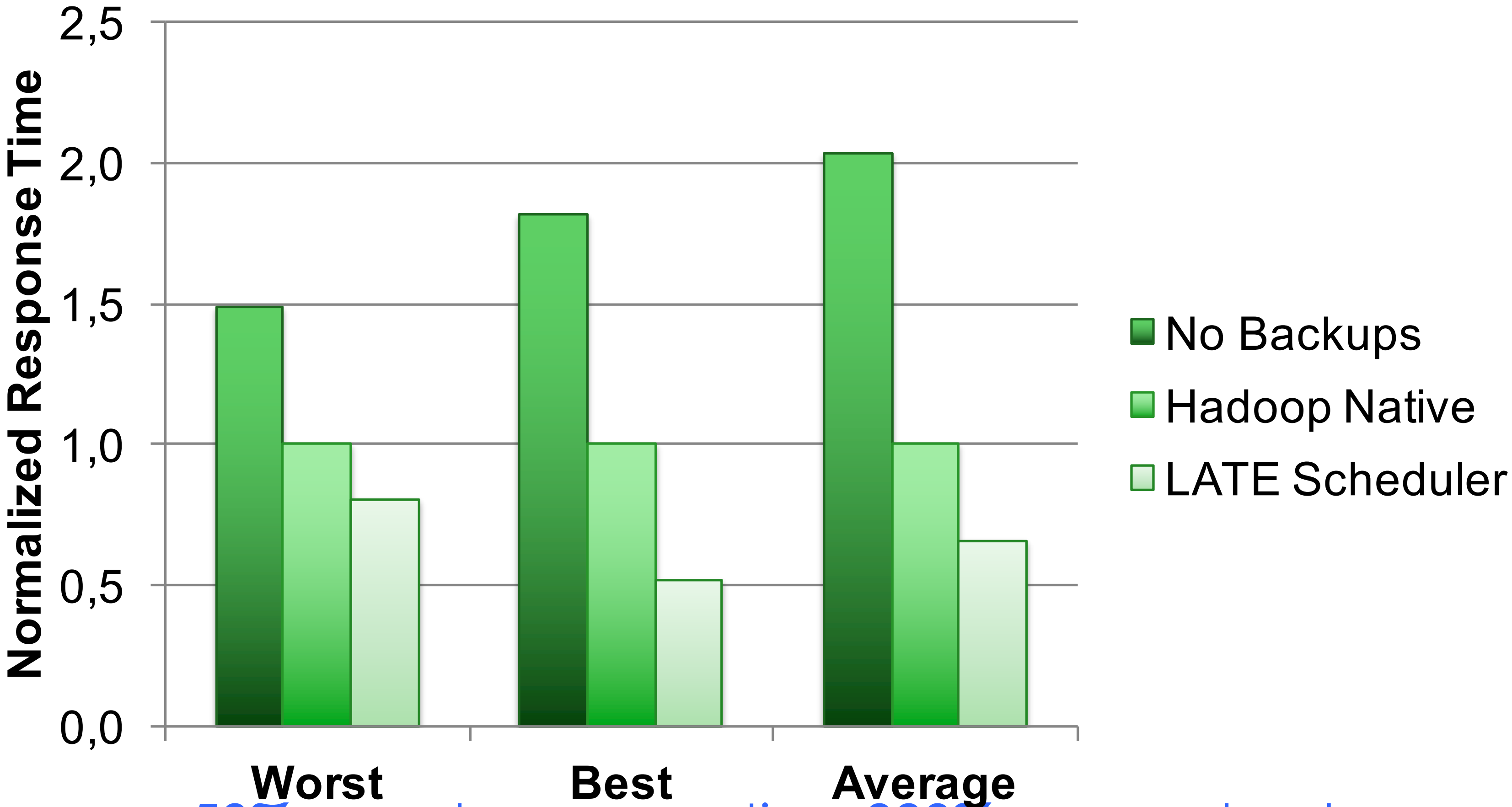
Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

Evaluation

- Environments:
  - EC2 (3 job types, 200-250 nodes)
  - Small local testbed
- Self-contention through VM placement
- Stragglers through background processes

Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

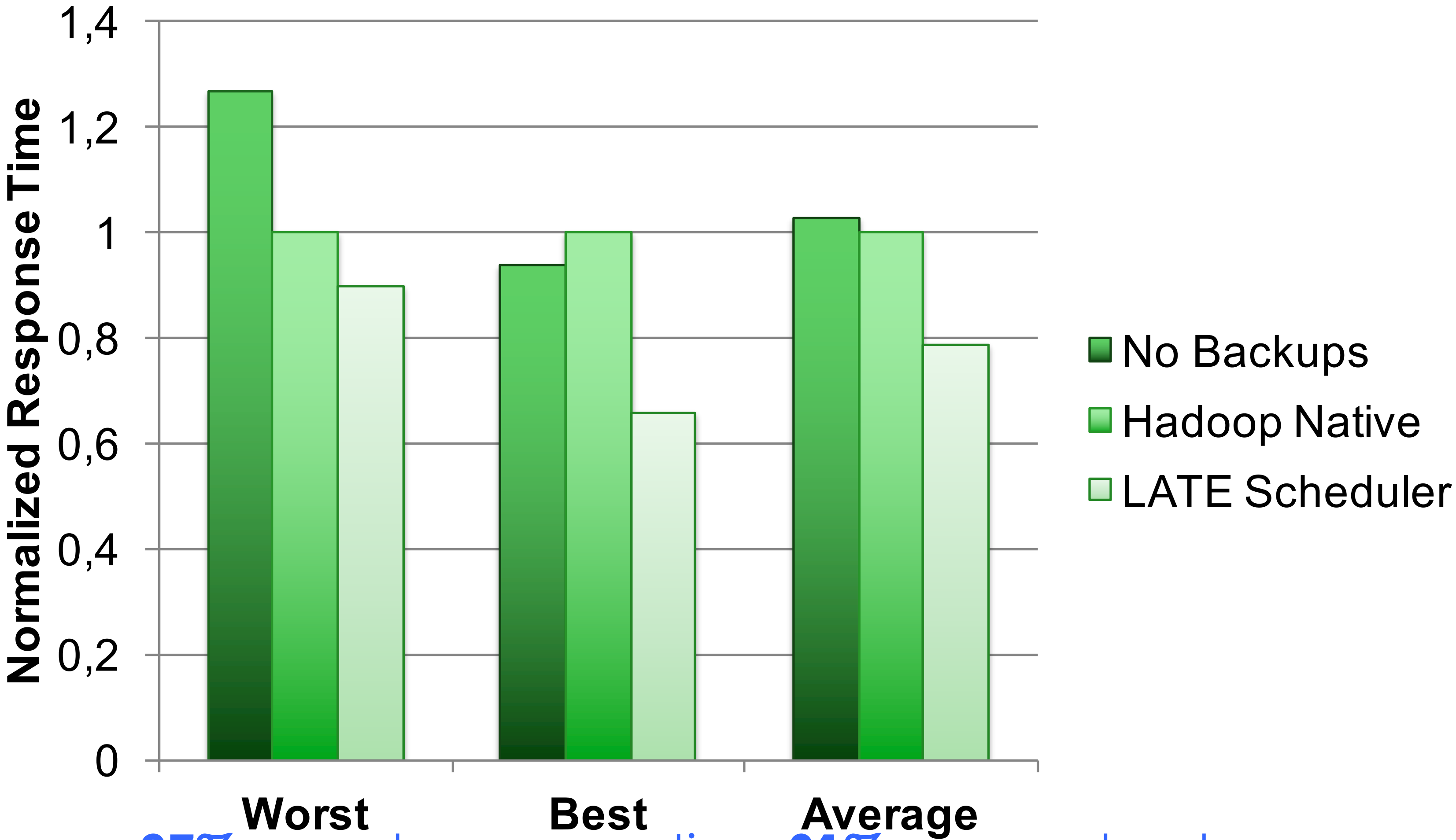
EC2 Sort with Stragglers



- Average **58%** speedup over native, **220%** over no backups
- **93%** max speedup over native

Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

EC2 Sort without Stragglers



- Average **27%** speedup over native, **31%** over no backups

Adopted from a presentation by Matei Zaharia "Improving MapReduce Performance in Heterogeneous Environments", OSDI 2008, San Diego, CA, December 2008.

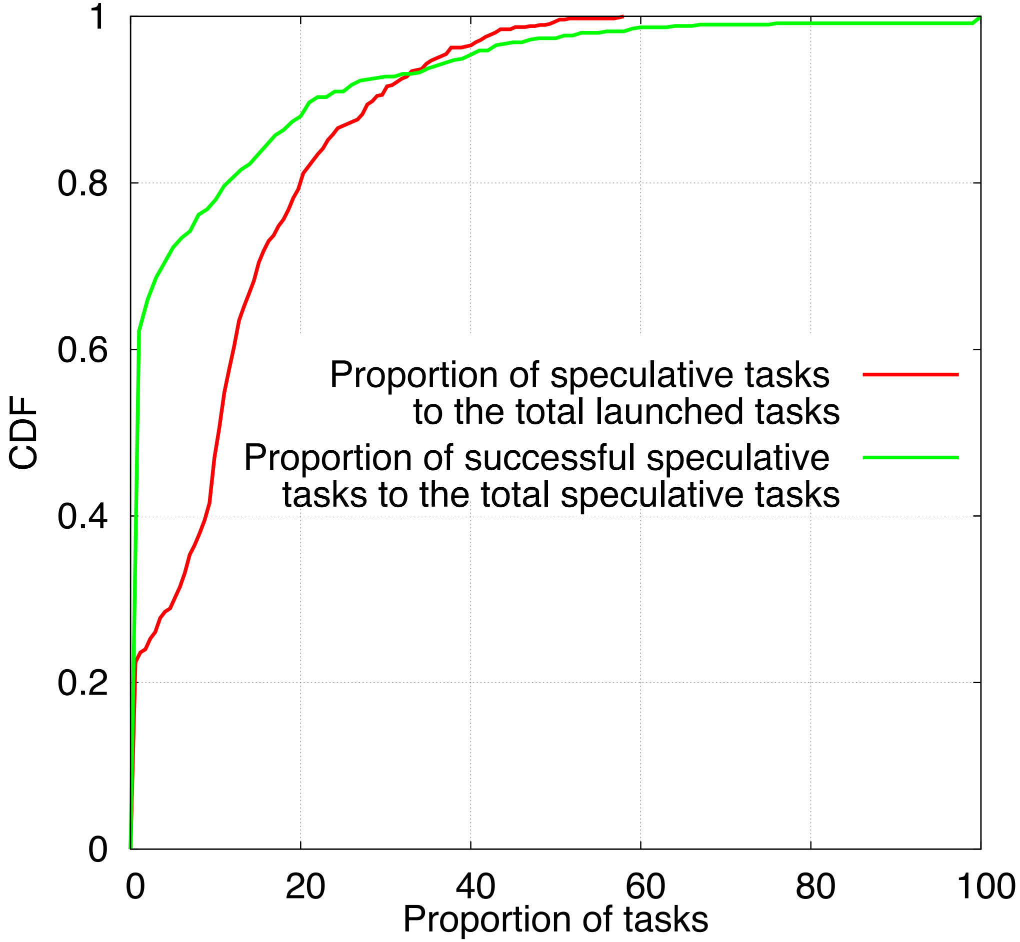
Cost of speculative execution

T-D. Phan, S. Ibrahim, G. Antoniu, L. Bouge . *On Understanding the Energy Impact of Speculative Execution in Hadoop.* *Greencom 2016*

CMU trace of production Hadoop cluster

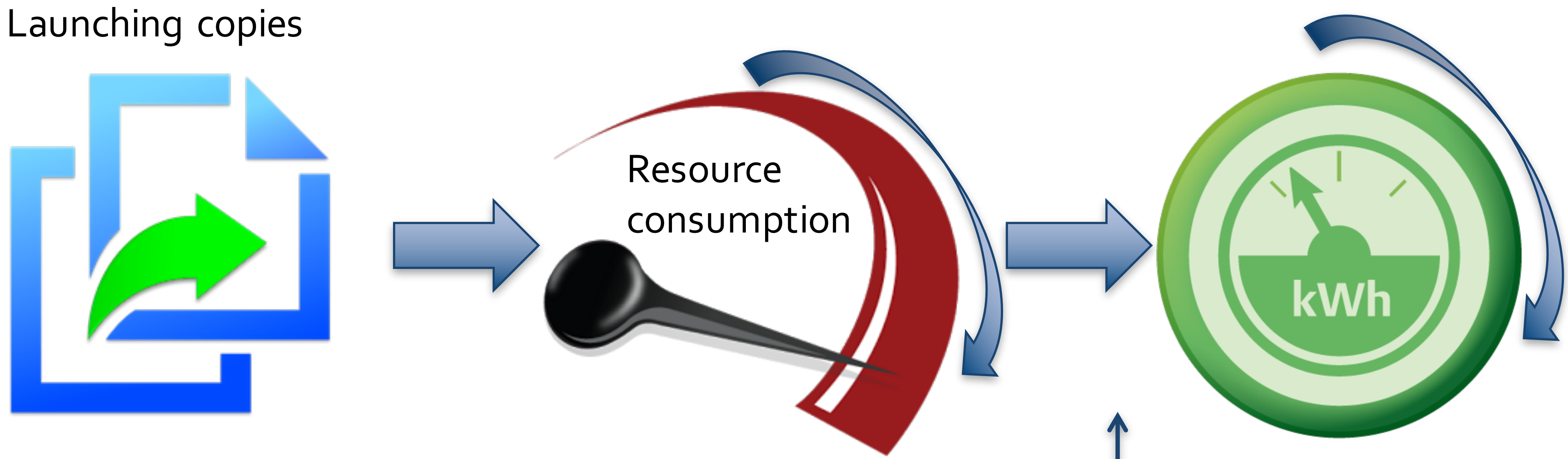
Cost of speculative execution

T-D. Phan, S. Ibrahim, G. Antoniu, L. Bouge . *On Understanding the Energy Impact of Speculative Execution in Hadoop.* *Greencom 2016*

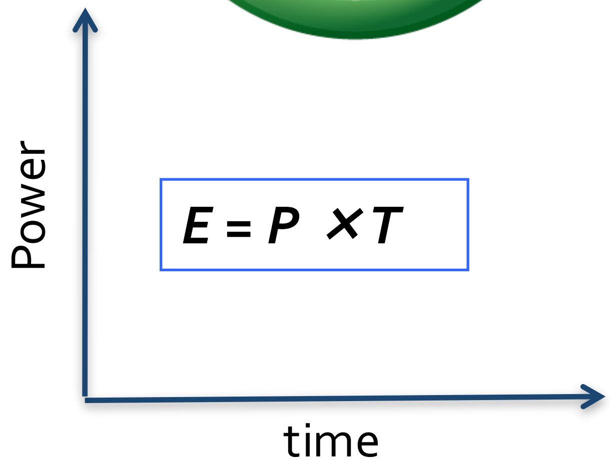


CMU trace of production Hadoop cluster

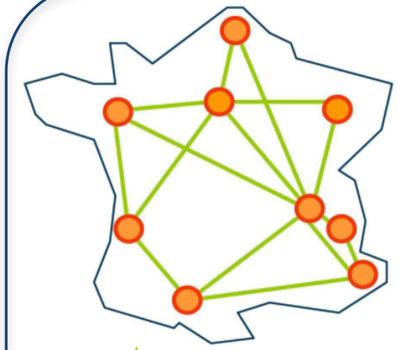
Energy vs. Speculation



Understand the impact of the speculative execution on the energy consumption



Methodology: testbed and platform



Grid'5000

21 nodes on
Nancy Site

Each node has:

- Intel 4-core CPU
- 16GB memory
- 256 GB storage
- Gigabit connection
- PDU for power monitoring

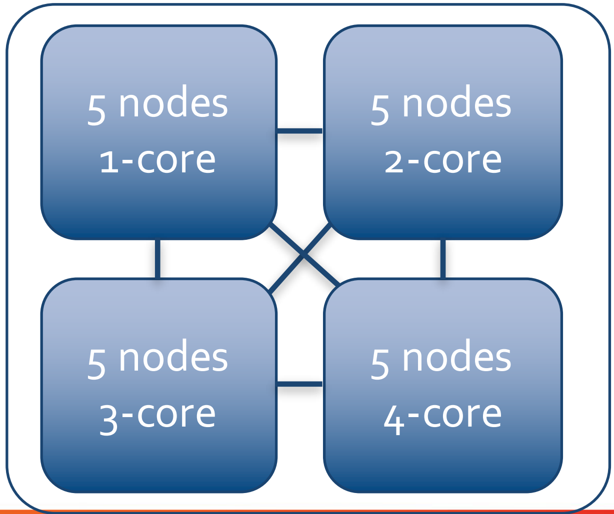
Hadoop 1.2.1

8 Map slots, 2 Reduce slots per node
(8 for CloudBurst)
Replication factor: 3
Block size: 64MB

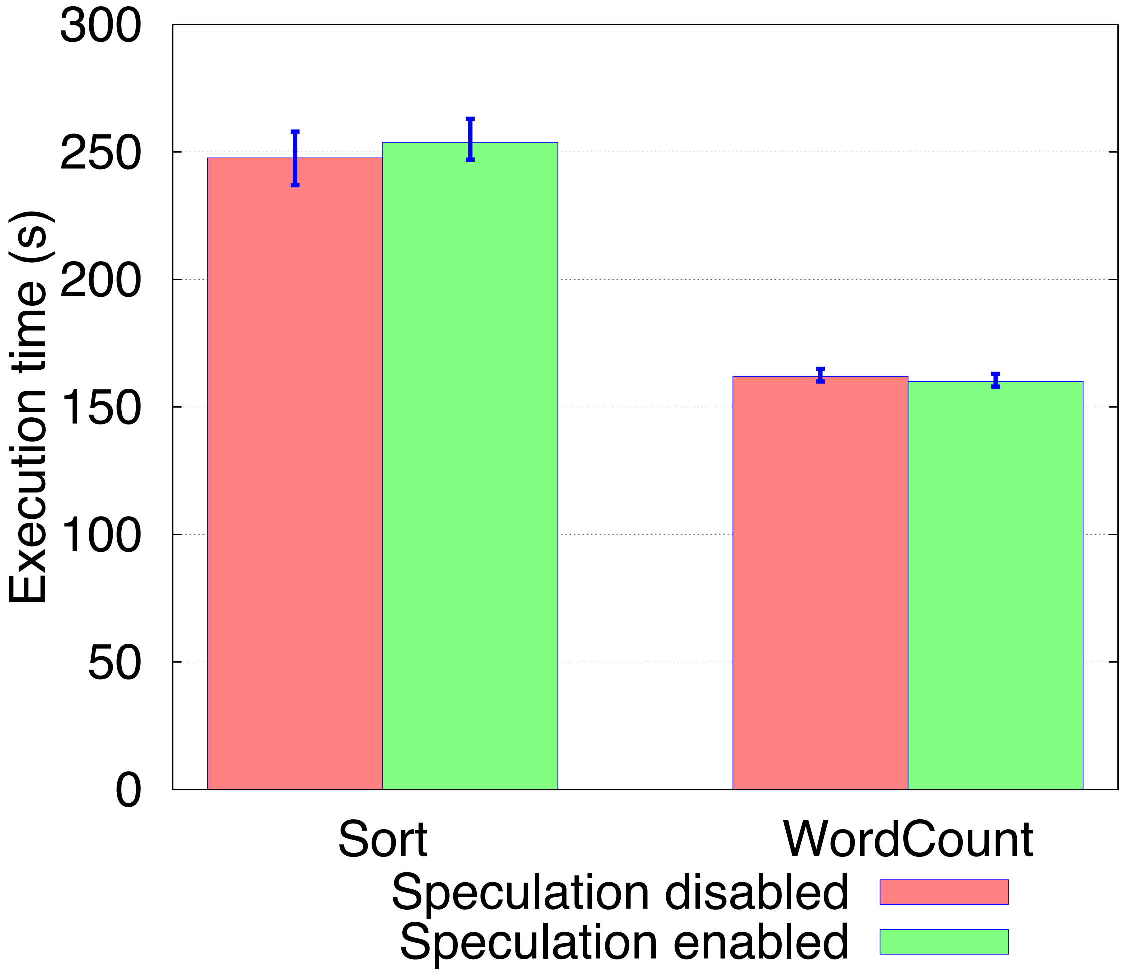
Methodology: benchmarks

Application	Sort	WordCount	CloudBurst
Dominating phase	Shuffle	Map	Reduce
Resource	Network	CPU	CPU
Input size	24.5GB	24.6GB	100MB
Output size	24.5GB	200MB	9.7GB
Map tasks	394	396	200
Reduce tasks	40	40	160

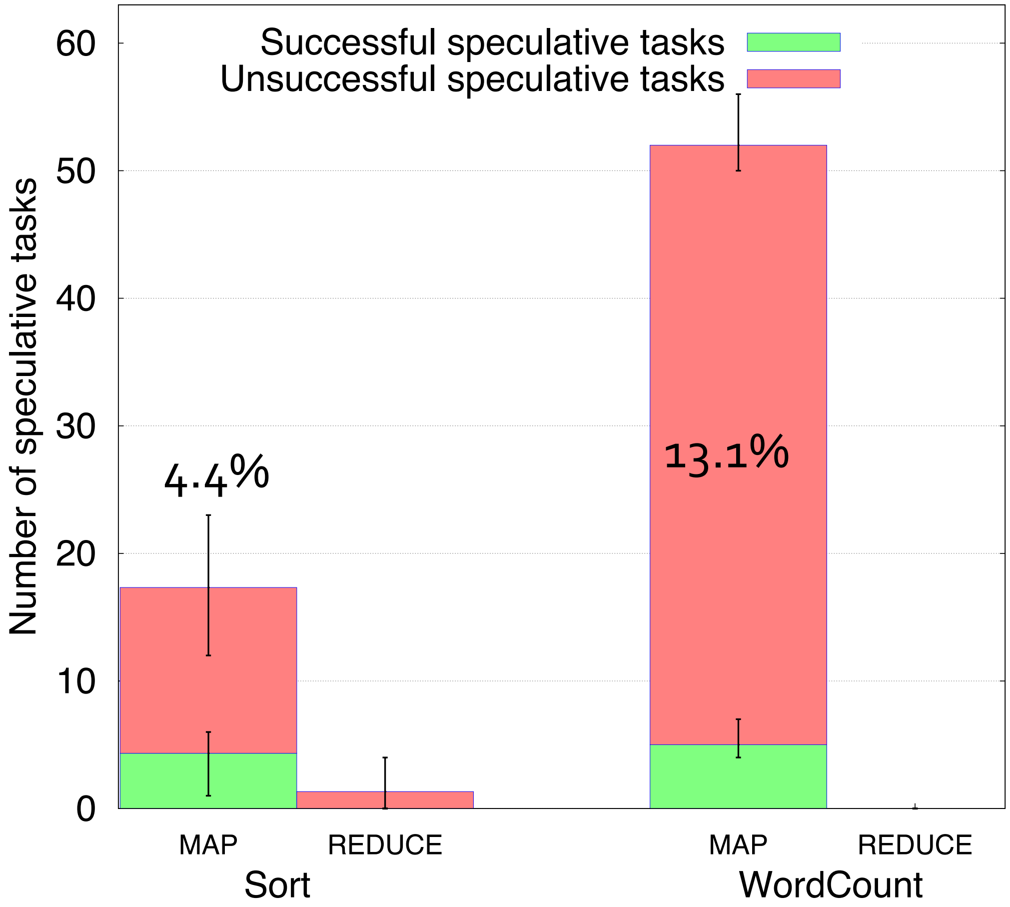
Heterogeneous cluster



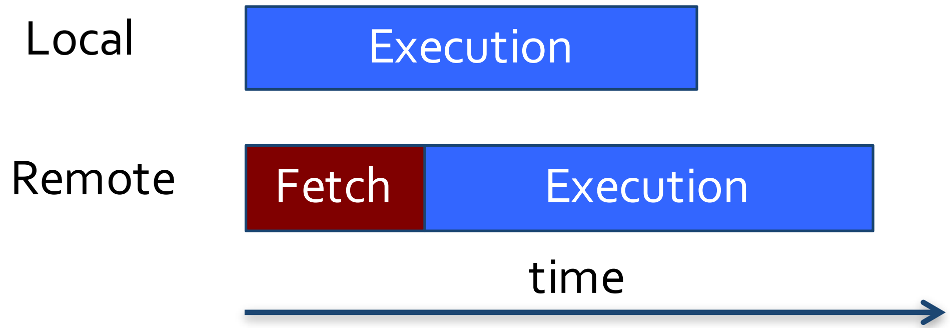
Homo-environment: Execution time



Homo-environment:
Unsuccessful speculative ratio

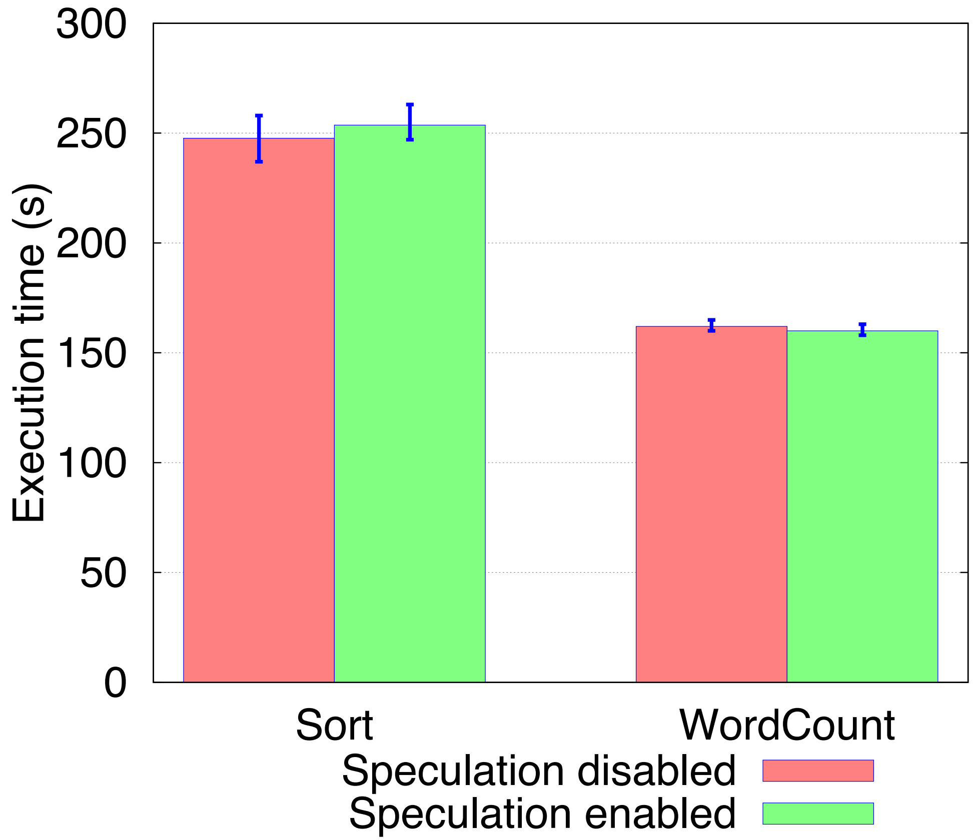


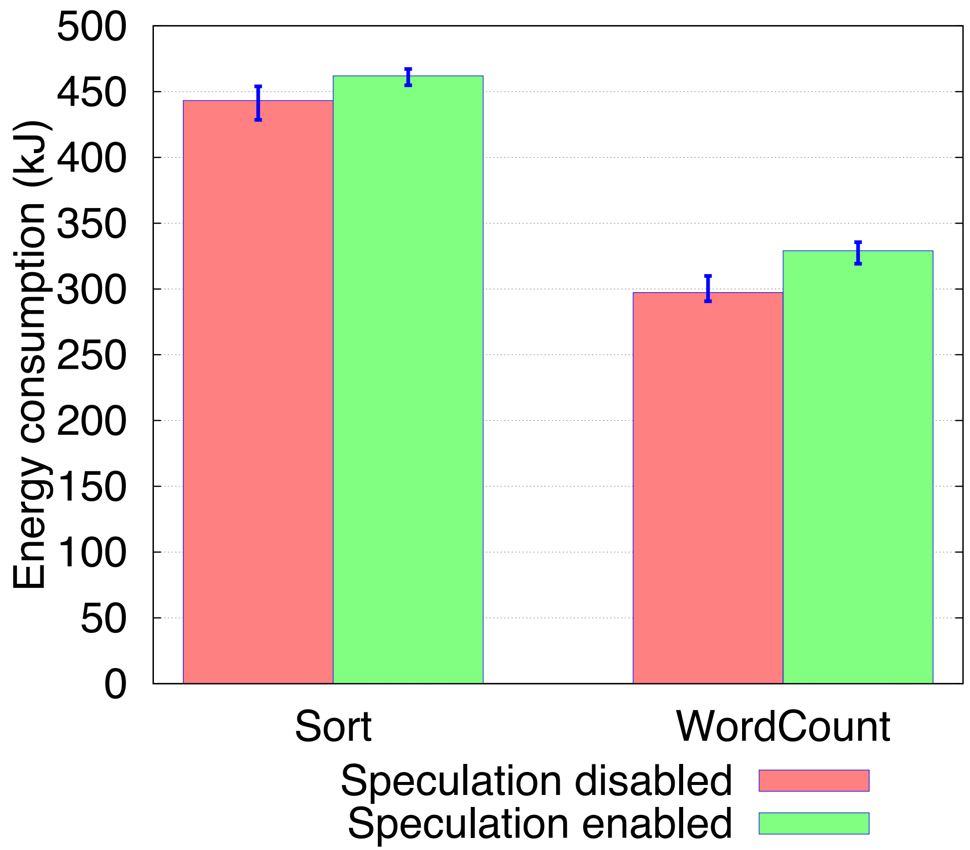
Non-local tasks take longer time to finish[1]



[1] Ibrahim et al., Maestro: Replica- aware map scheduling for MapReduce, CCgrid2012

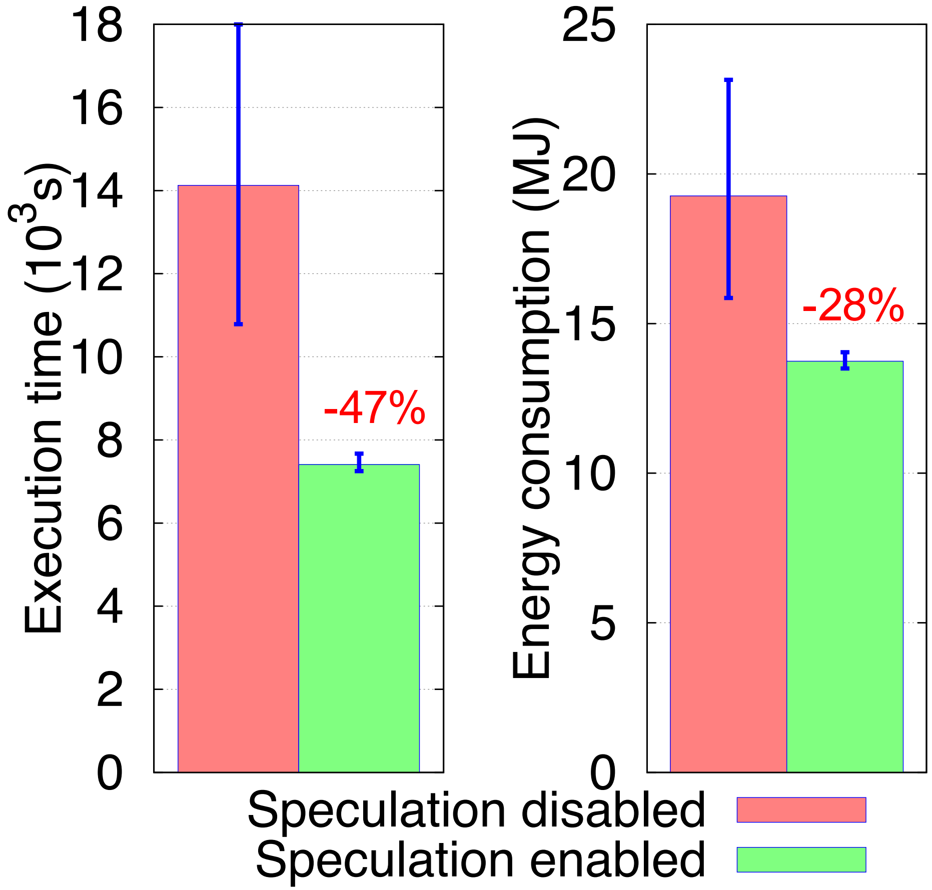
Homo-environment:
Energy consumption





The unsuccessful speculative copies result in extra energy consumption

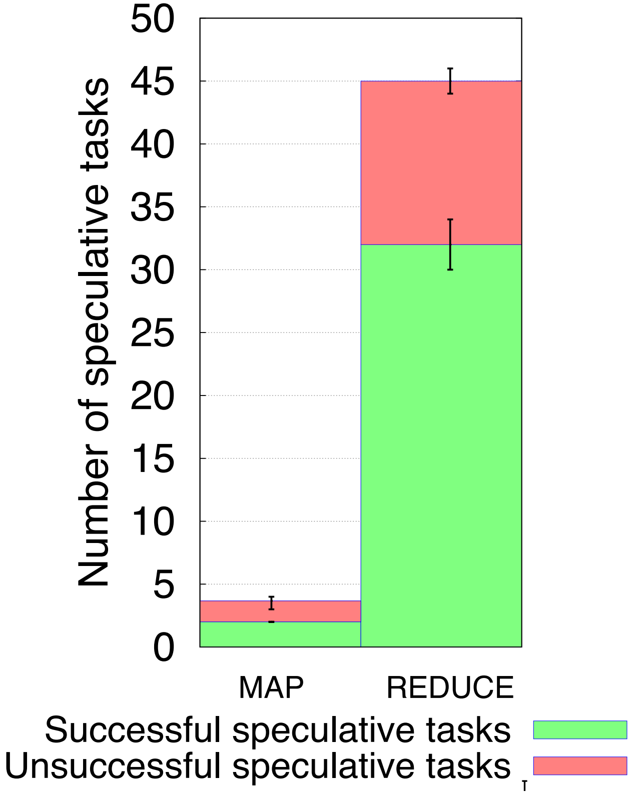
Speculation benefit in Hete-environment



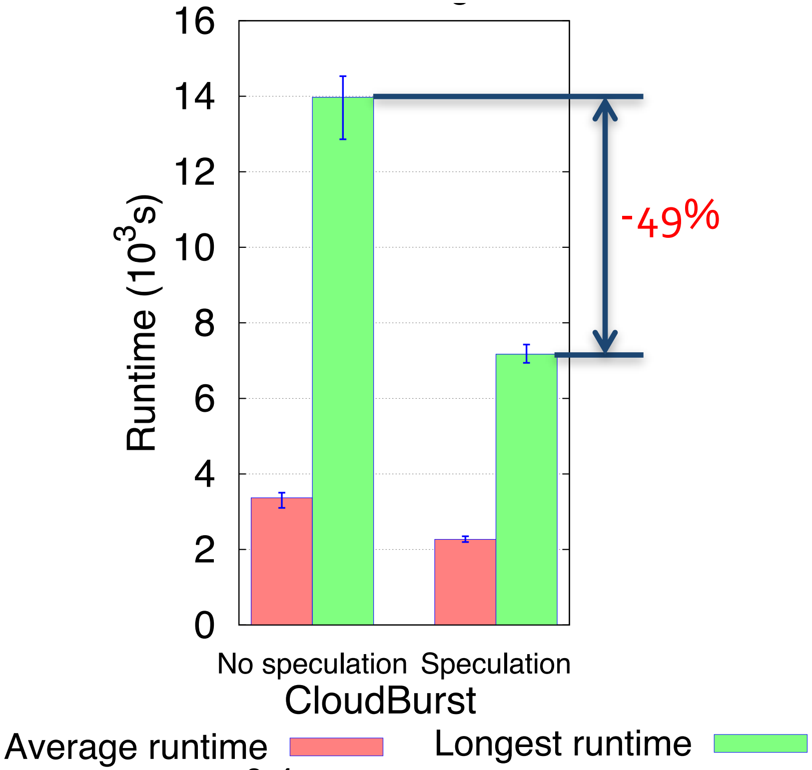
The energy reduction is not proportional to the execution time improvement

Zoom on speculation behavior and impact

- High ratio of successful copies

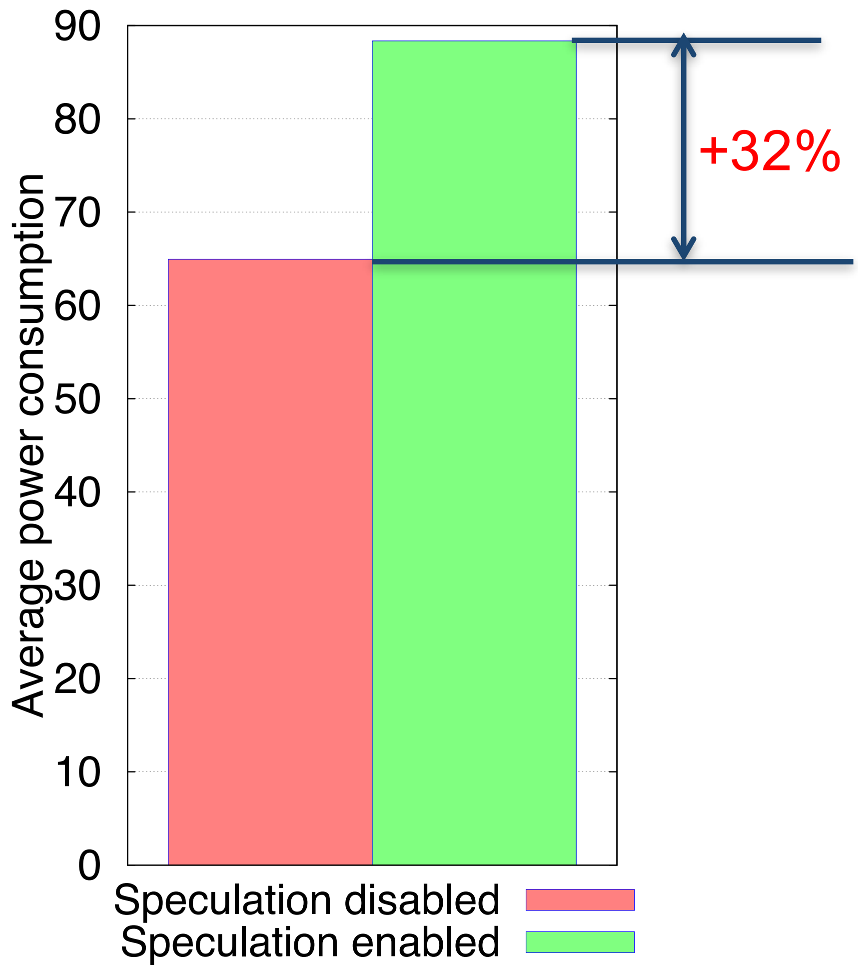


- Reduce the longest tasks

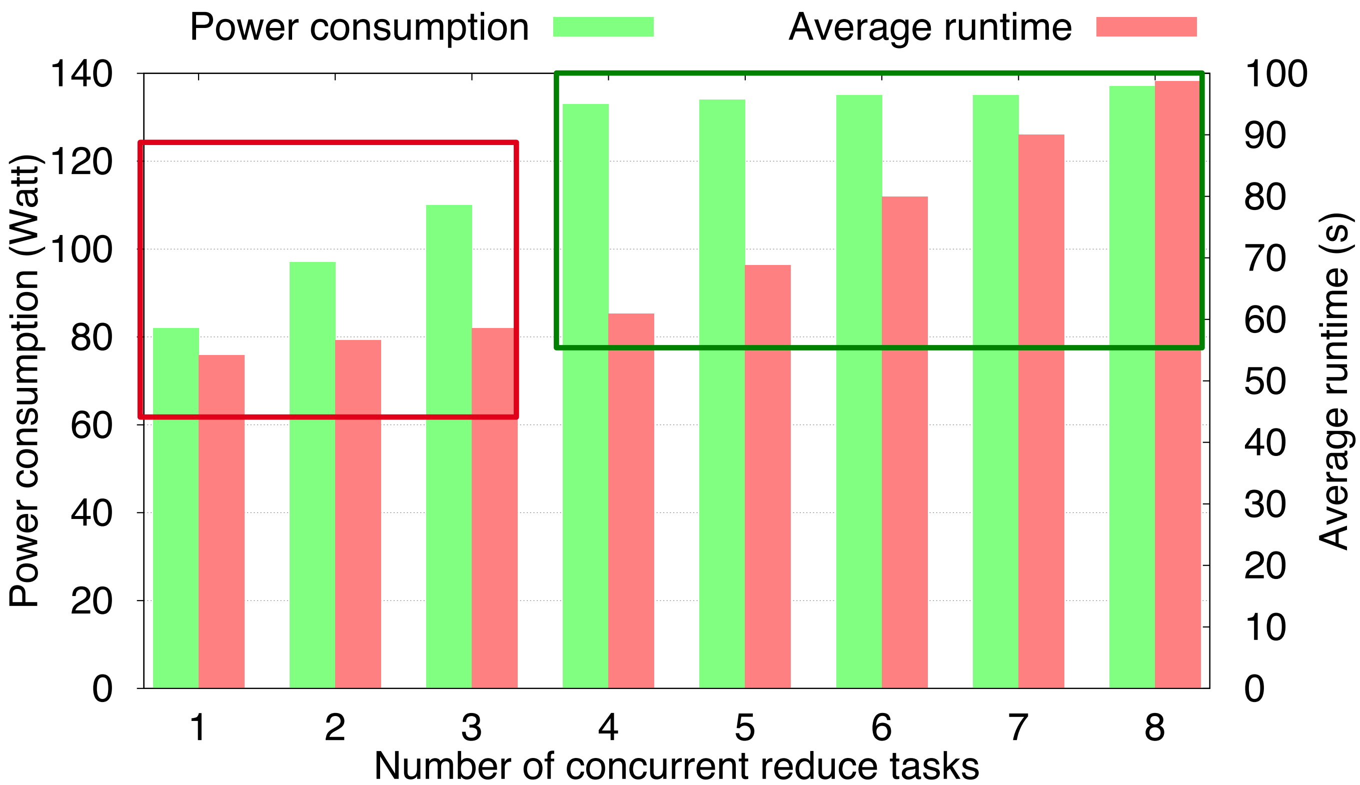


Successful speculative copies reduce the execution time of slow tasks and result in significant performance improvement

Impact of speculation on power consumption



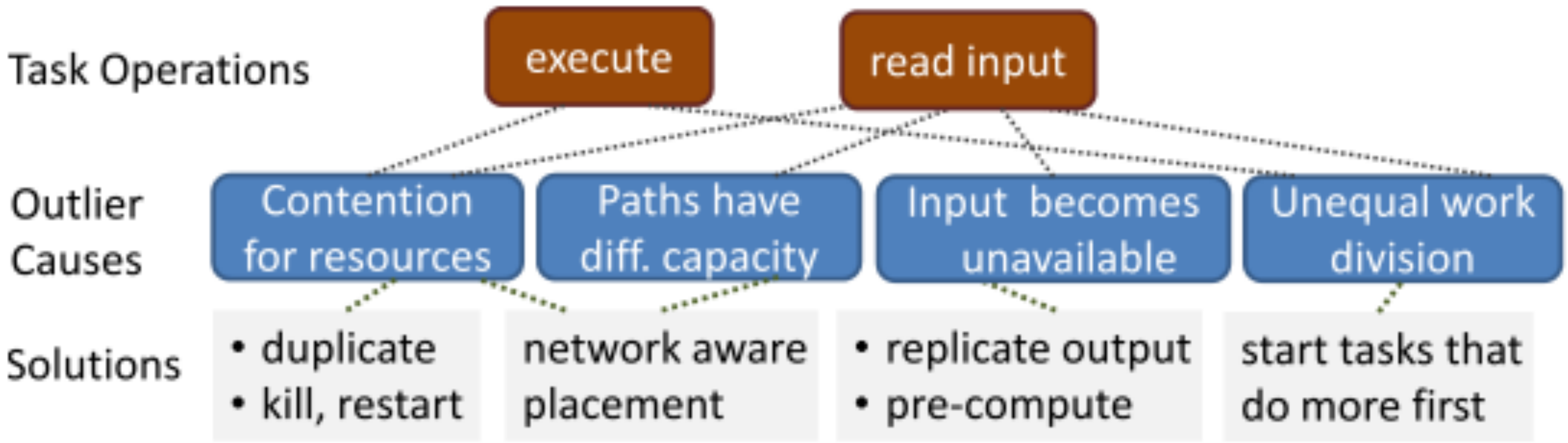
Power and Performance under different task allocations



Tradeoffs between power cost and the performance gain of different speculative copies allocations

Straggler handling

- Mantri (Ananthanarayanan et al.)
- Cloning (Ananthanarayanan et al.)



Open Issues

- ...in production clusters

- LATE: The slowest task runs **8 times** slower* than the median task in a job
- Mantri: The slowest task runs **6 times** slower* than the median task in a job
- (but they work well for large jobs...)

Effective Straggler Mitigation: Attack of the Clones. NSDI 2013

Open Issues

Considering When and where ?

Provide better task/job scheduling

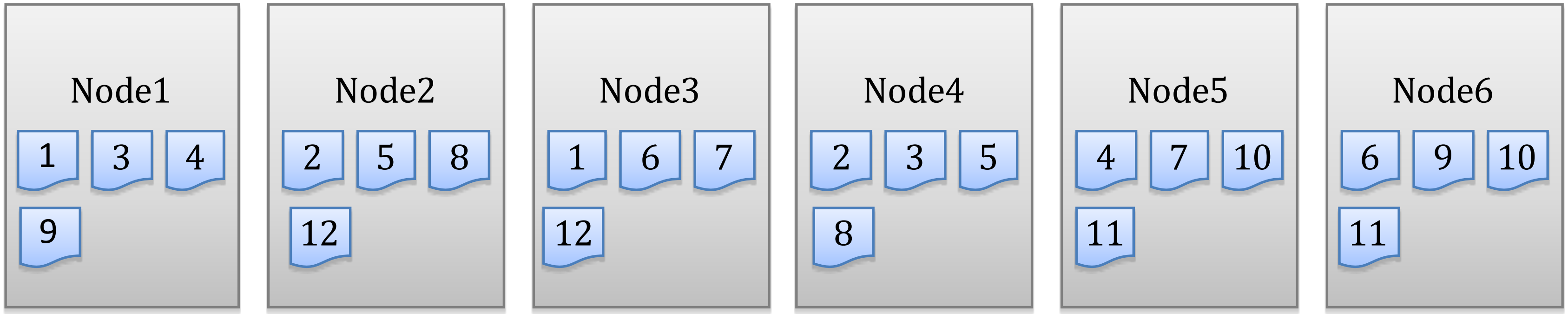
Data Locality: Task and job scheduling

Maestro: Replica-Aware Map Scheduling for MapReduce.
The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing CCGrid 2012, May 13-16, 2012, Ottawa, Canada (CCGRID2012).

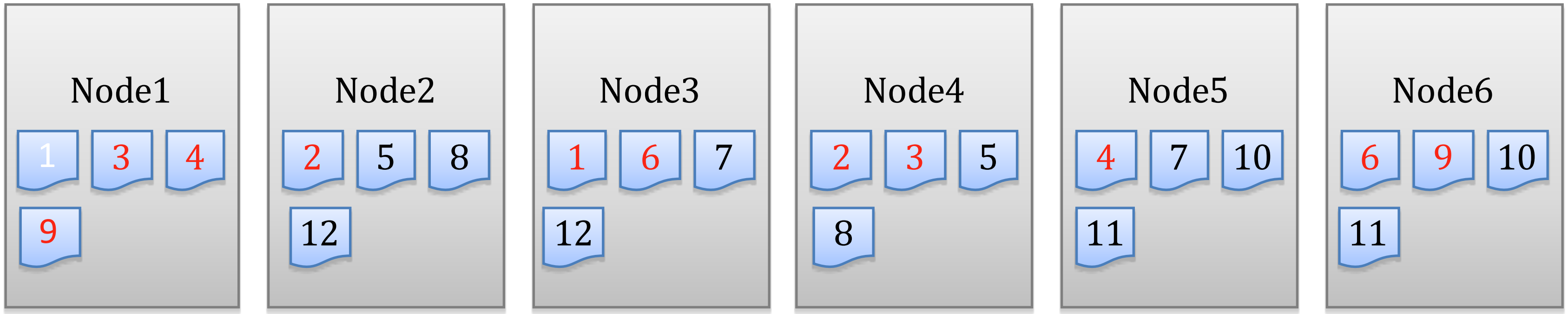
Why Data locality?

- Data locality is crucial for Hadoop's performance
- How can we expose data-locality of Hadoop in the Cloud efficiently?
- Hadoop in the Cloud
  - Unaware of network topology
  - Node-aware or non-local map tasks

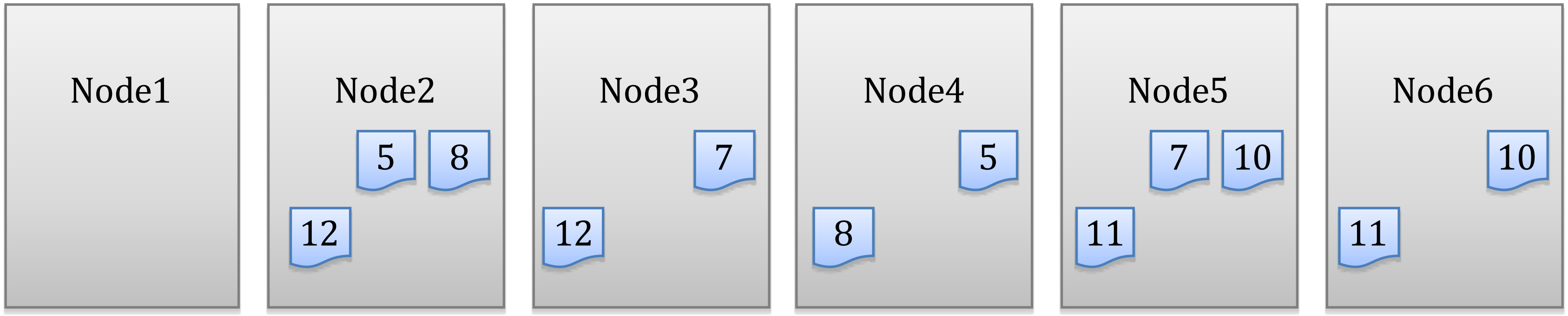
Data locality in the Cloud



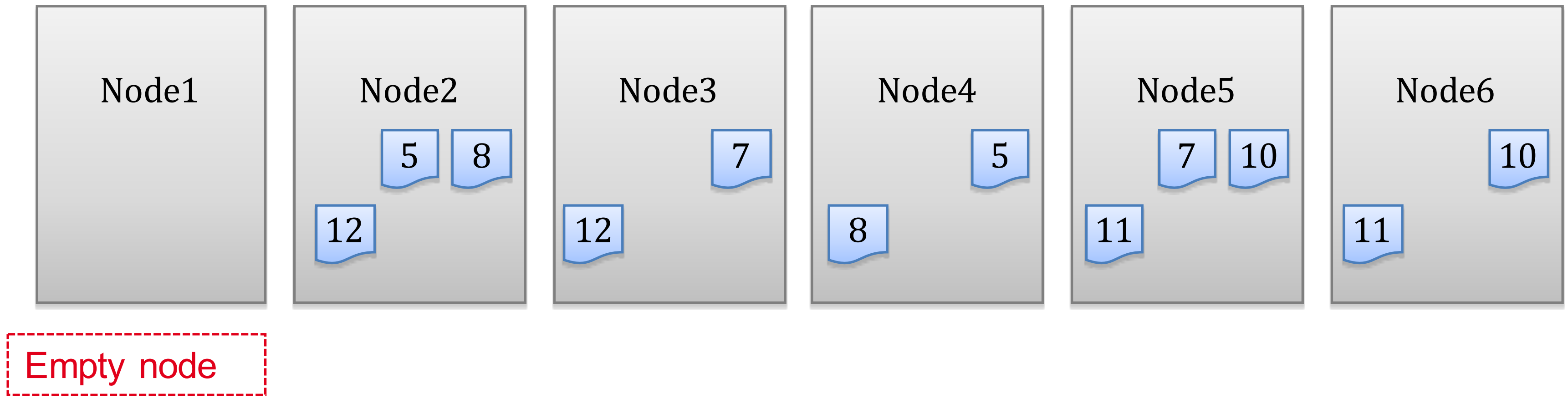
Data locality in the Cloud



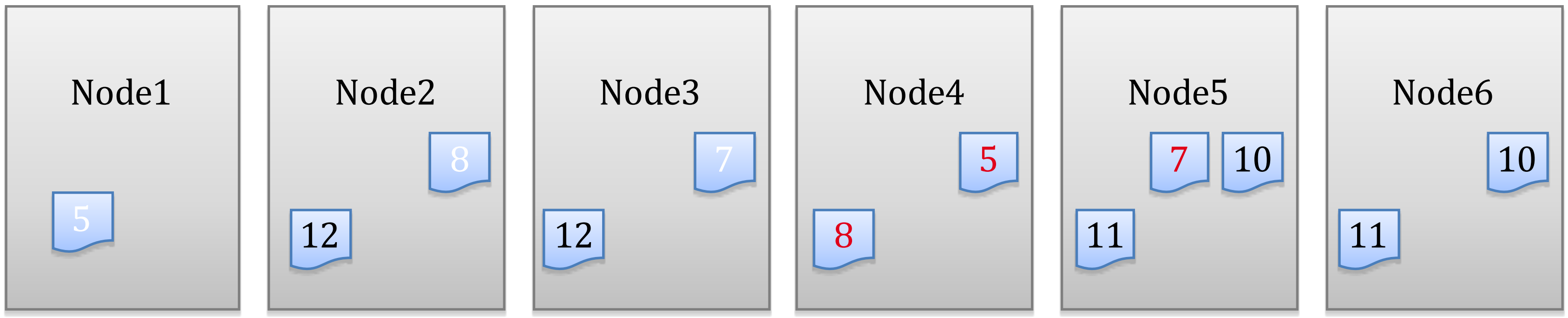
Data locality in the Cloud



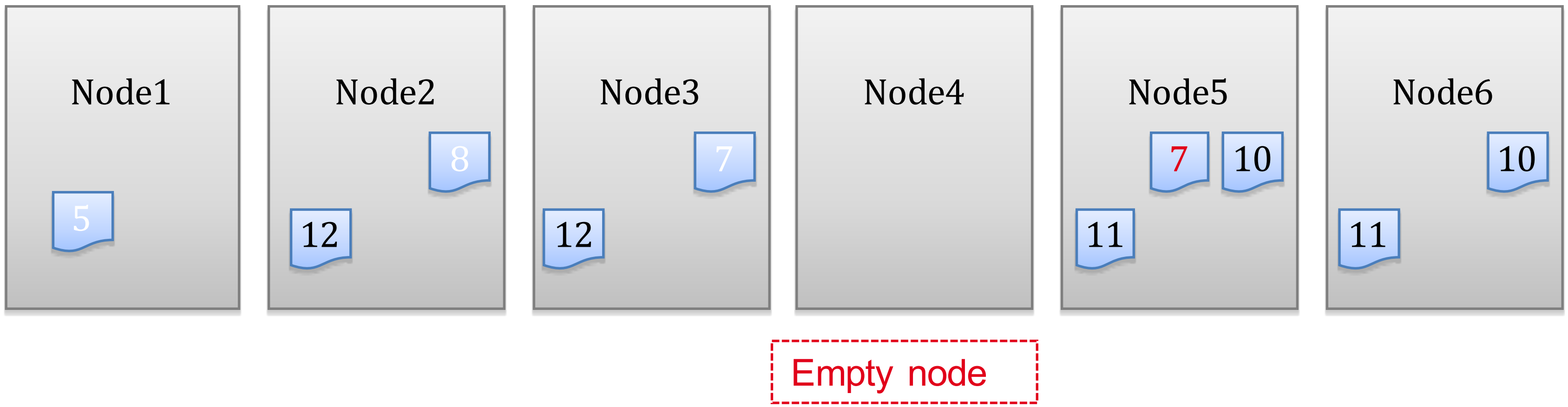
Data locality in the Cloud



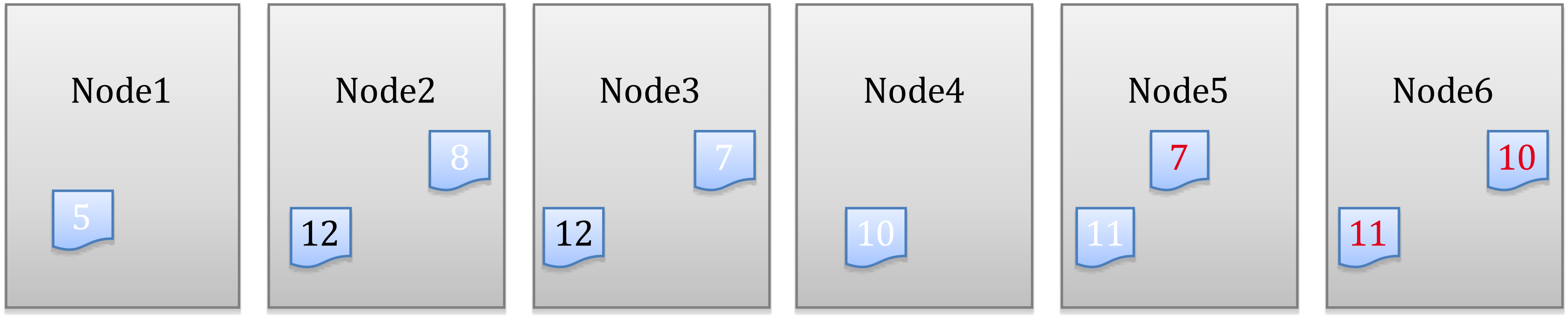
Data locality in the Cloud



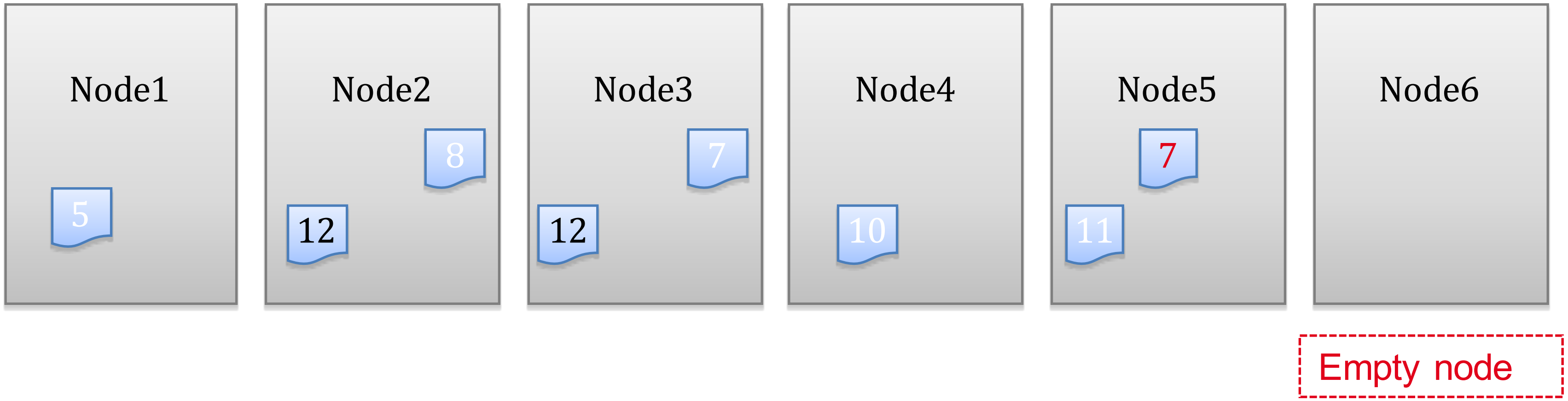
Data locality in the Cloud



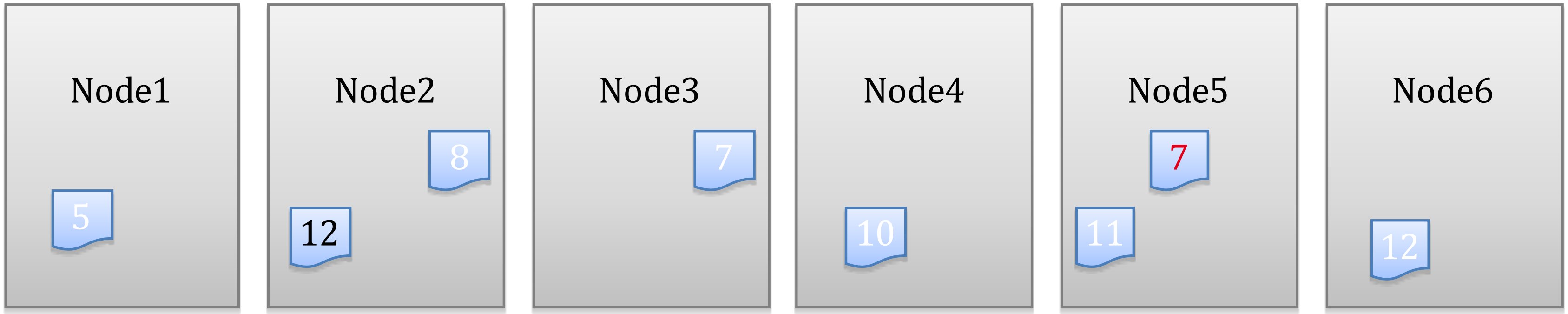
Data locality in the Cloud



Data locality in the Cloud



Data locality in the Cloud



The simplicity of Map tasks Scheduling leads to

Non-local maps execution (25%)

Side impacts:

- Increase the execution time
- Increase the number of useless speculation
- Slot occupying
  - Imbalance in the Map execution among nodes

Maestro: Replica-Aware scheduling in Hadoop

Schedule the map tasks in two waves:

First wave: fills the empty slots of each data node based on the number of hosted map tasks and on the replication scheme for their input data

Second wave: runtime scheduling takes into account the probability of scheduling a map task on a given machine depending on the replicas of the task's input data.

Results: Maestro can achieve optimal data locality even if data are not uniformly distributed among nodes and improve the performance of Hadoop applications

Maestro Details

- Selecting Data nodes
  - Has minimal potential to execute map tasks localy
  - Has minimal impacts on other nodes
    - Share chunks with more nodes
- Selecting chunks
  - Has maximal probability of not being processed locally
- The three heuristics are all applied for the first wave
- Only the third one applied in the runtime wave (heartbeat)

Maestro: First Wave

No. of Chunks

Replication Factor

$$NodeW_i = \sum_{j=1}^{HCN_i} \left(1 - \sum_{k=1}^{r} \frac{1}{HCN_k + Sc_k^i}\right)$$

Node weight

No. of shared chunks between $node_i$ and the $node_j$

- Sort the node ascendingly according to their NodeW:
  - Node with less chunks are preferred
  - Node with low share rate are preferred
    - The higher the share rate indicates higher NodeW

Maestro: Runtime

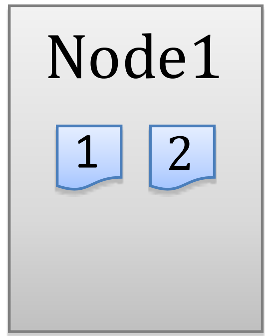
$$\underbrace{Cw_i}_{\text{Chunk weight}} = 1 - \left(\frac{1}{HCN_1} + \frac{1}{\underbrace{HCN_2}_{\text{No. of chunks hosted by the node host the replica}}} + \ldots + \frac{1}{HCN_r}\right)$$

- Chunks share data with nodes host more data are prioritized

Maestro: Refinements

Fault-Tolerance

$$\underbrace{[Cw'_i = 1 - \sum_{j=1}^{k-1} \frac{1}{HCN_j}]}_{\text{Chunk weight of failed Map tasks}} > \underbrace{[Cw_i = 1 - \sum_{j=1}^{k} \frac{1}{HCN_j}]}_{\text{The weight of other chunks within The same node}}$$



- *$[Cw_1 = \frac{1}{2}] > [Cw_2 = 1 - (1/2 + 1/HCN_r)]$*

Maestro: Refinements

- Heterogeneous Cloud

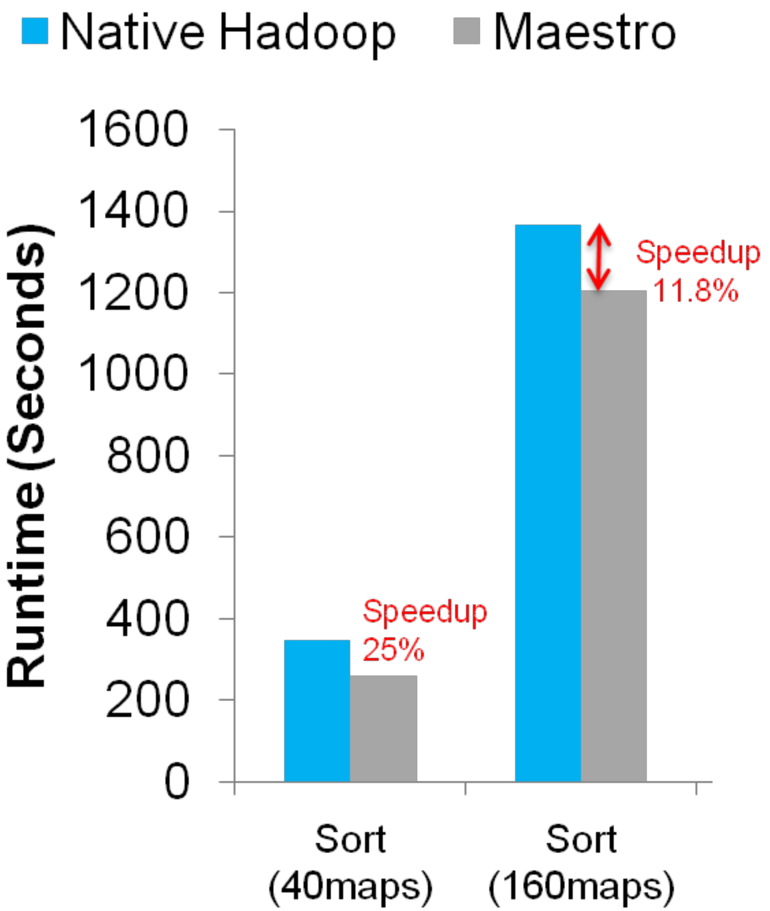
Higher S_r results with lower Cw

$$Cw_i = 1 - \left(\frac{s_1}{HCN_1} + \frac{s_2}{HCN_2} + \ldots + \frac{s_r}{HCN_r}\right)$$

- Maestro prefers the chunks shared with low computation capacity
- Prevent the nodes with higher computation capacity to be out of chunks

Results

- Sort application



		Local maps
Virtual Cluster	Native Hadoop	83%
	Maestro	96%
Grid5000	Native Hadoop	84%
	Maestro	97%

S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, S. Wu, Maestro: Replica-aware map scheduling for mapreduce, in: CCGrid 2012

Data Locality in Shared Cluster

Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems* (EuroSys '10).

Motivation

- MapReduce / Hadoop originally designed for high throughput batch processing

Motivation

- MapReduce / Hadoop originally designed for high throughput batch processing

Motivation

- MapReduce / Hadoop originally designed for high throughput batch processing
- Today's workload is far more diverse:

Motivation

- MapReduce / Hadoop originally designed for high throughput batch processing
- Today's workload is far more diverse:
  - *Many users* want to share a cluster

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

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*

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

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

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

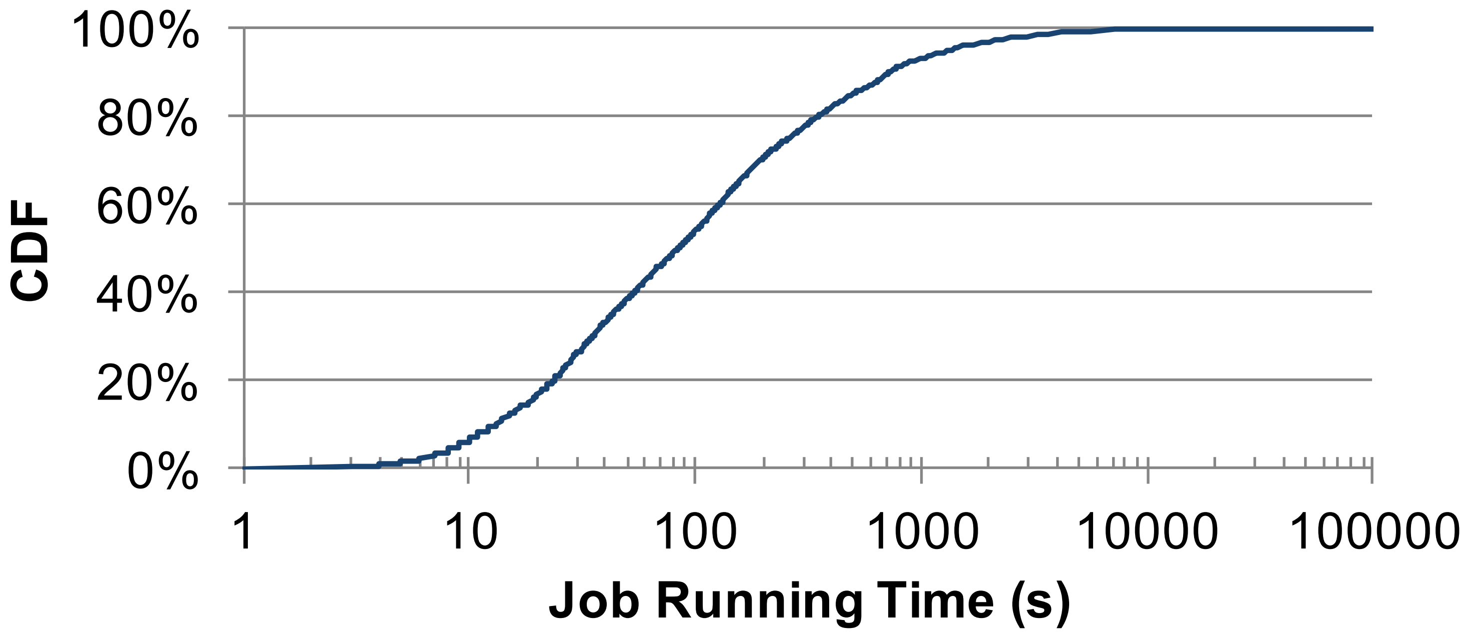
Motivation

- 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?

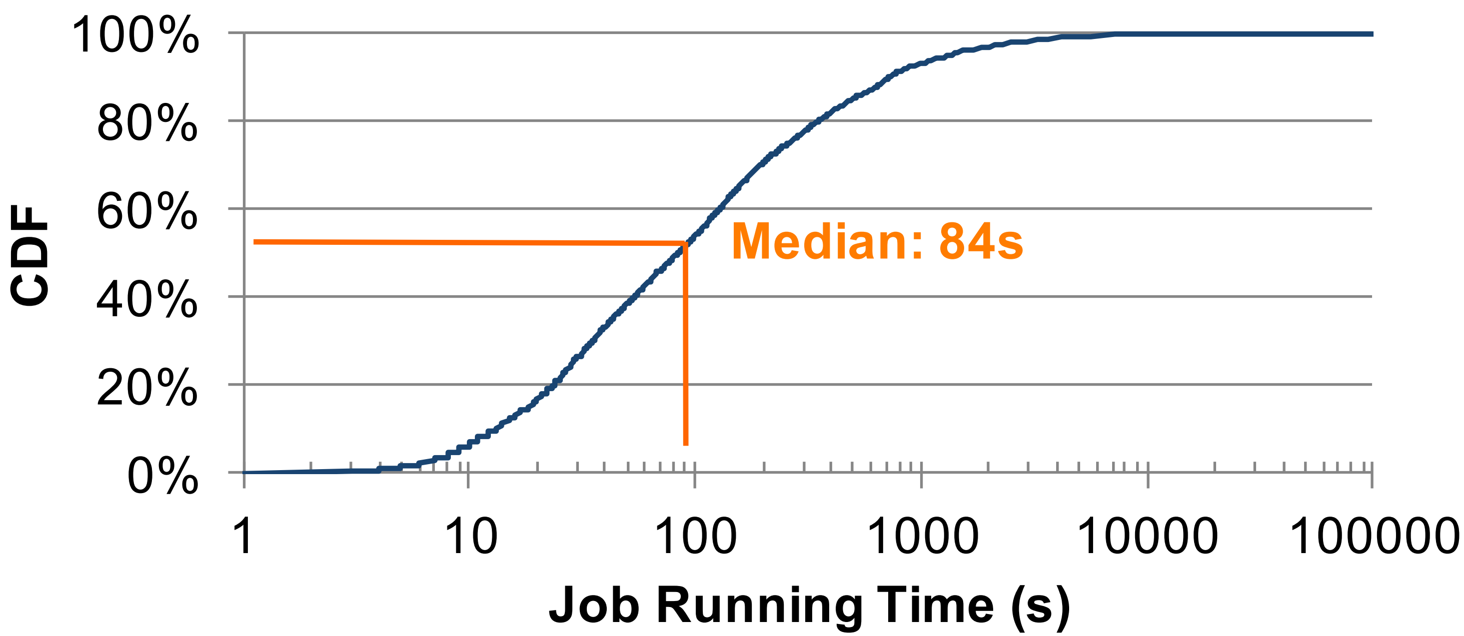
Example: Hadoop at Facebook

- 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



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



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

Solution

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

Solution

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

Solution

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

Solution

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

Solution

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

Solution

- 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

Solution

- 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

- 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

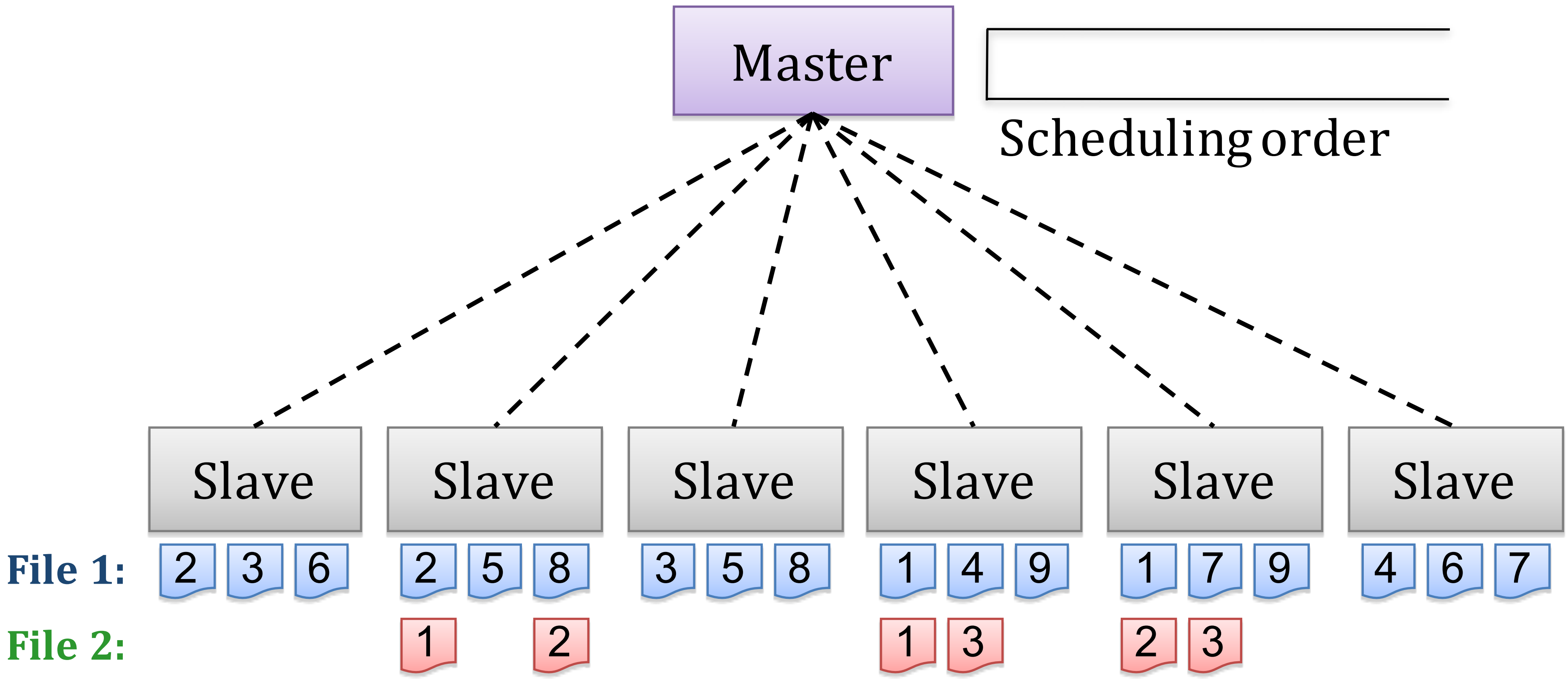
Solution

- 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

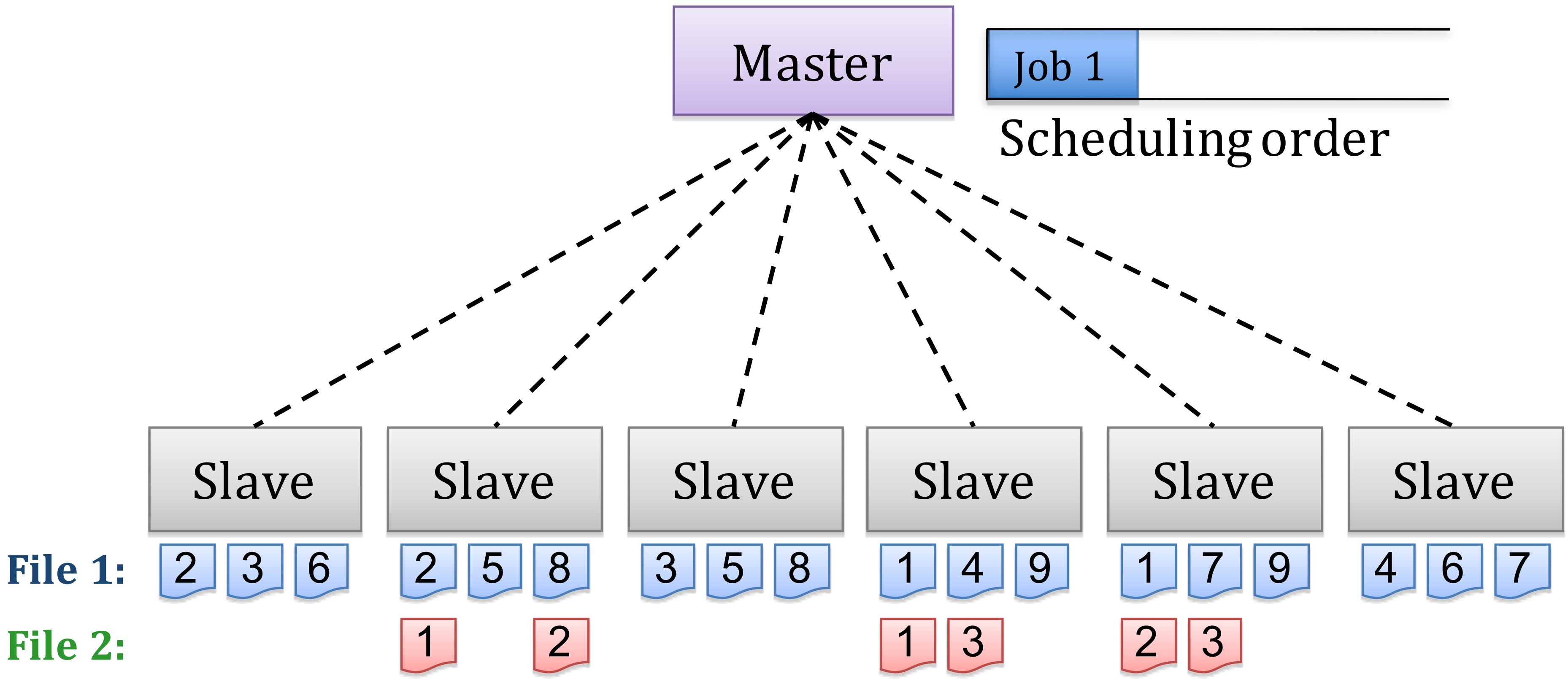
Solution

- 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

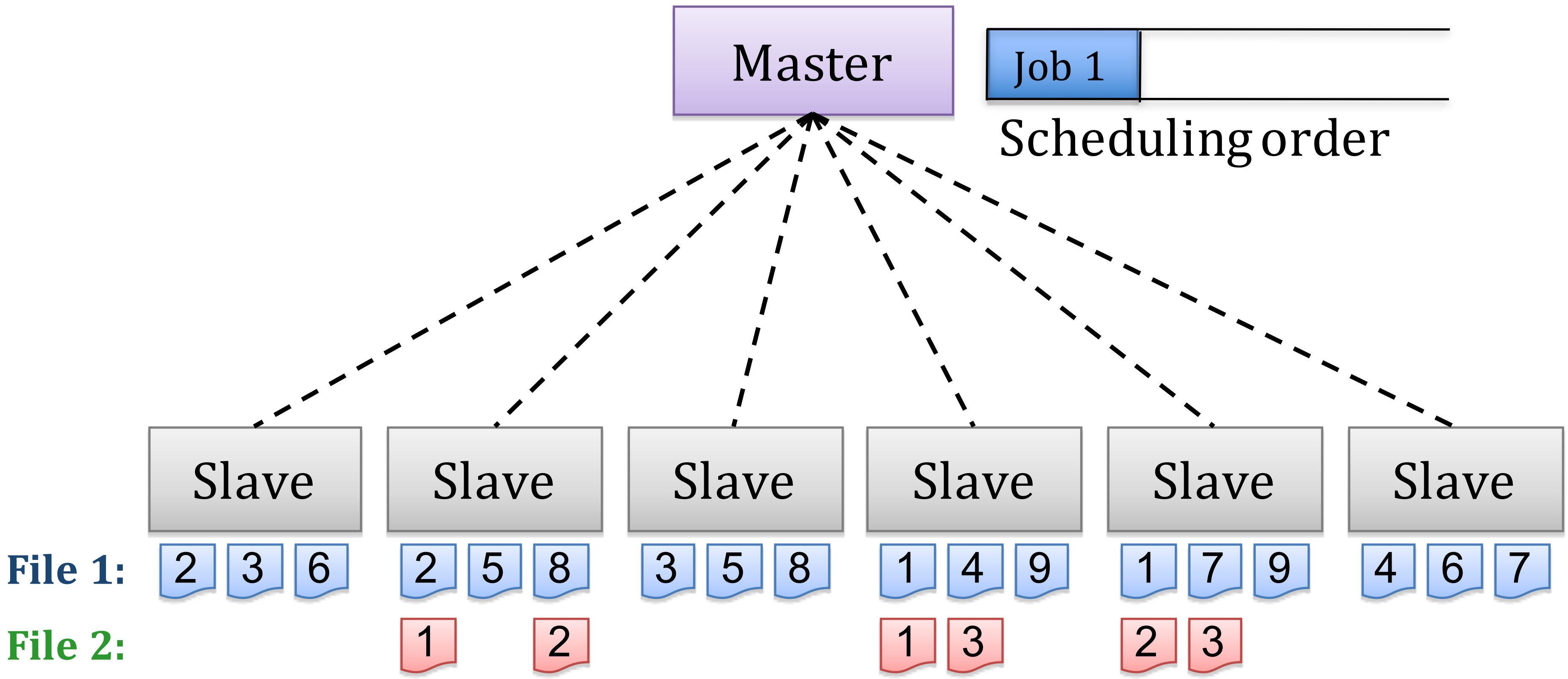
The Problem


Master
Scheduling order
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

The Problem

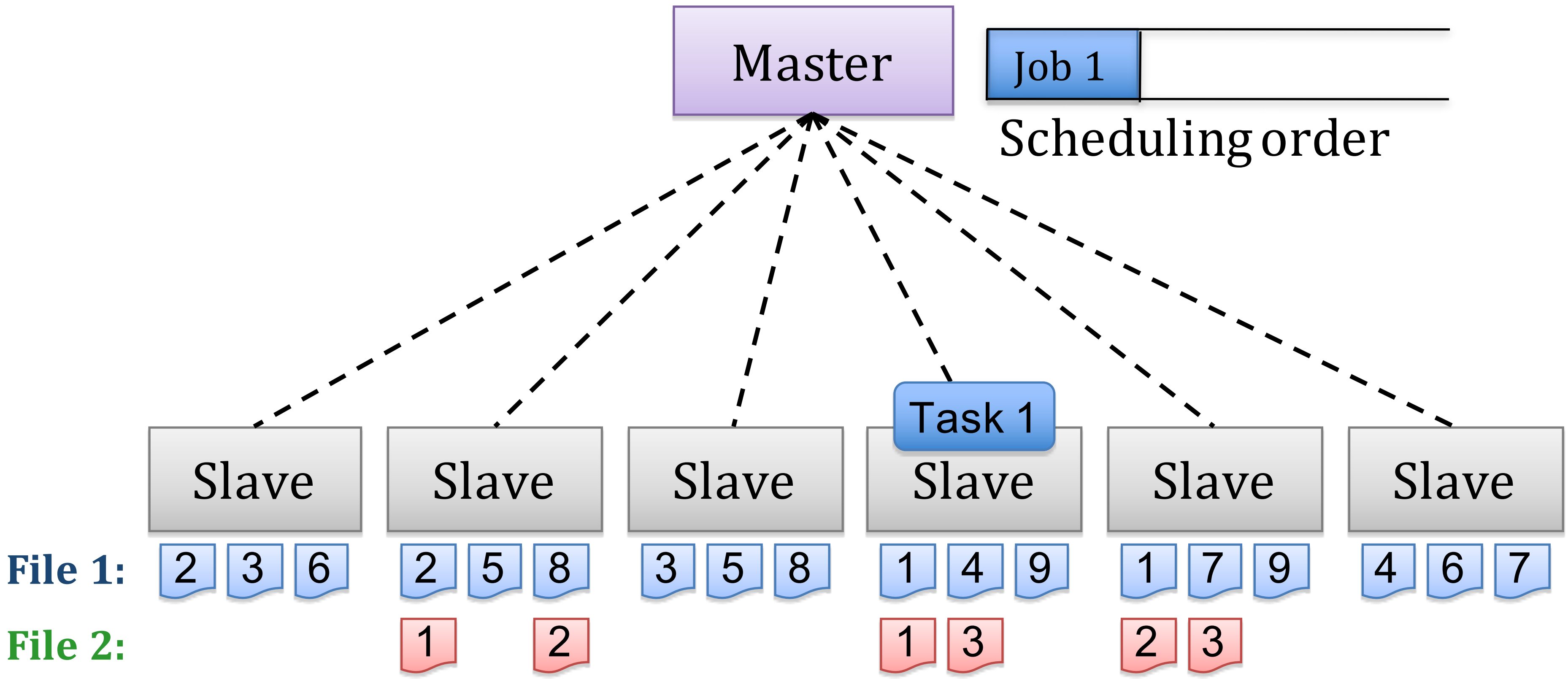

Master
Job 1
Scheduling order
Slave
Slave
Slave
Slave
Slave
Slave
File 1:
2 3 6
2 5 8
3 5 8
1 4 9
1 7 9
4 6 7
File 2:
1 2
1 3
2 3

The Problem

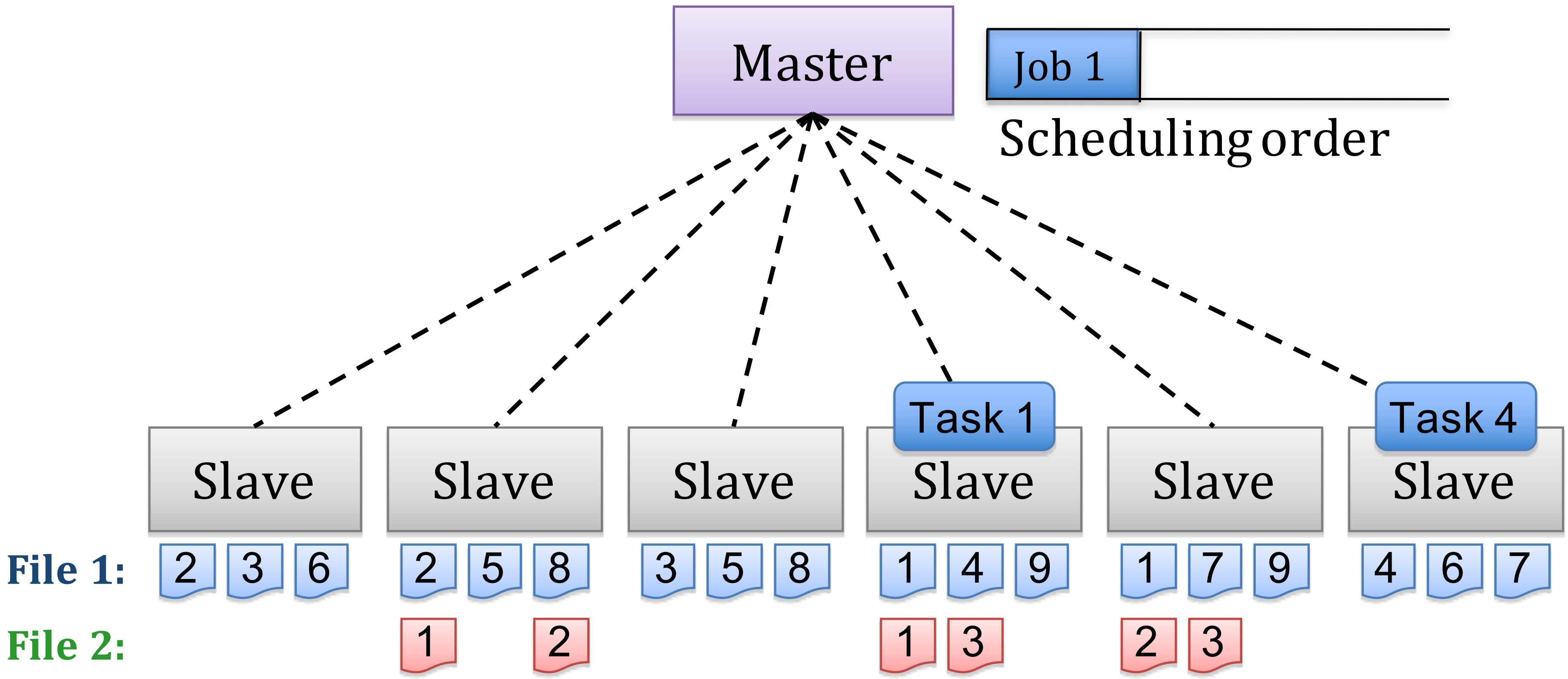


Master
Job 1
Scheduling order
Slave
Slave
Slave
Slave
Slave
Slave
File 1:
2 3 6
2 5 8
3 5 8
1 4 9
1 7 9
4 6 7
File 2:
1 2
1 3
2 3

The Problem

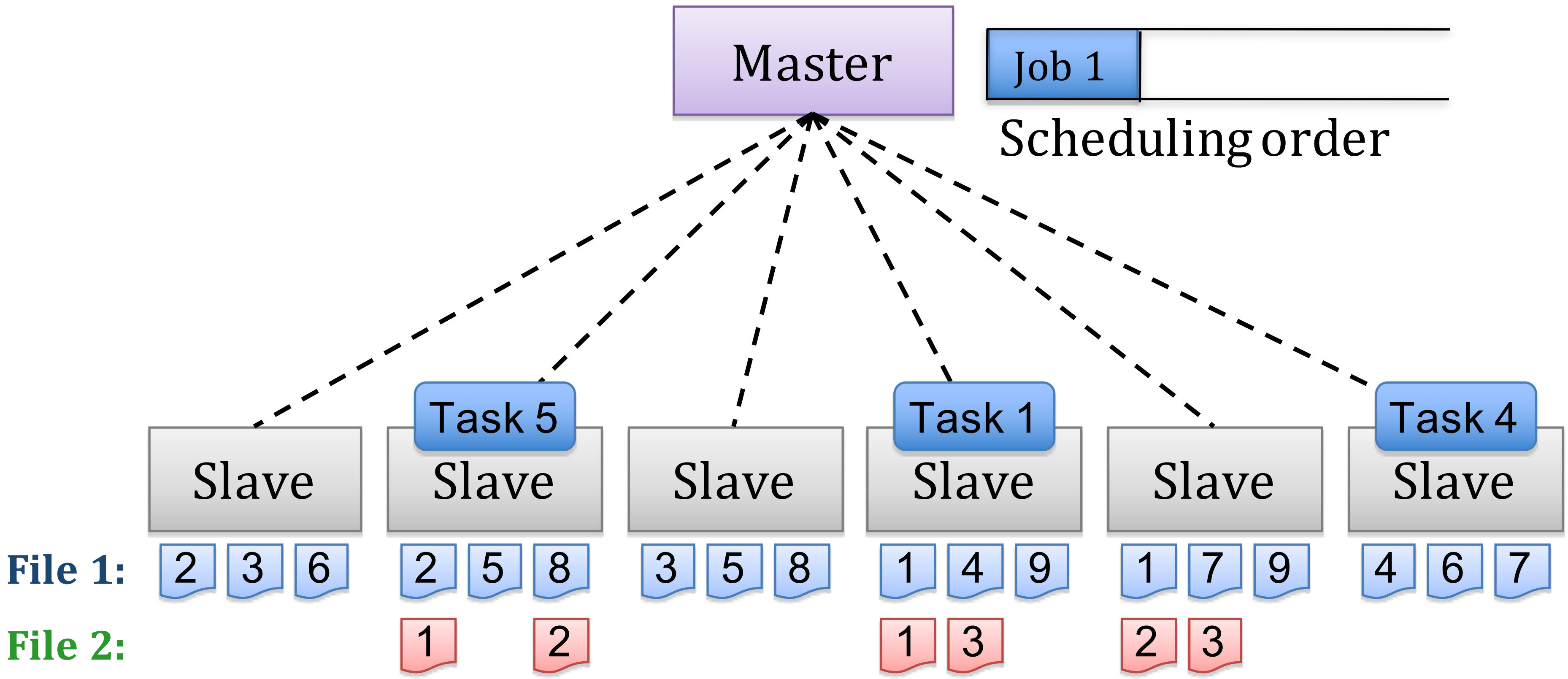

Master
Job 1
Scheduling order
Task 1
Slave
Slave
Slave
Slave
Slave
Slave
File 1:
2 3 6
2 5 8
3 5 8
1 4 9
1 7 9
4 6 7
File 2:
1 2
1 3
2 3

The Problem

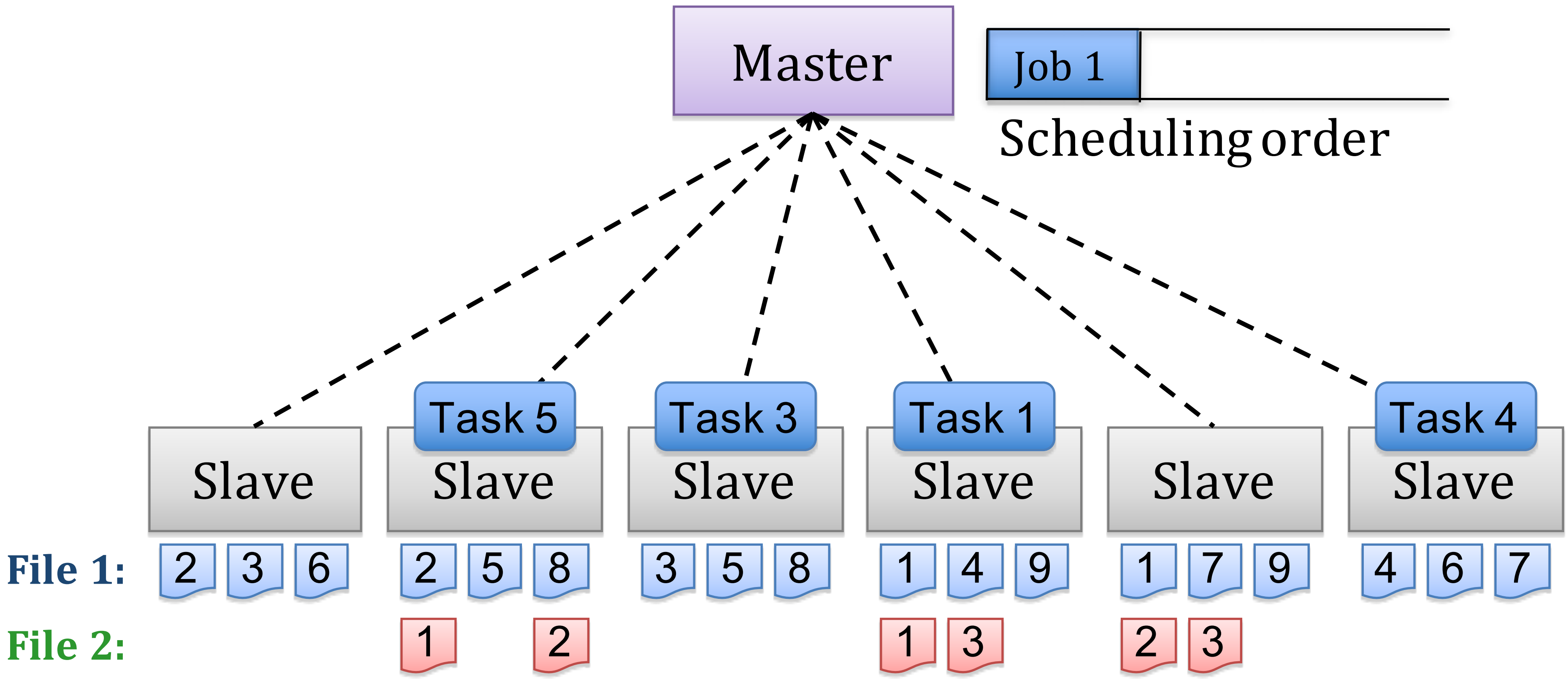


Master
Job 1
Scheduling order
Task 1
Task 4
Slave
Slave
Slave
Slave
Slave
Slave
File 1:
2 3 6
2 5 8
3 5 8
1 4 9
1 7 9
4 6 7
File 2:
1 2
1 3
2 3

The Problem

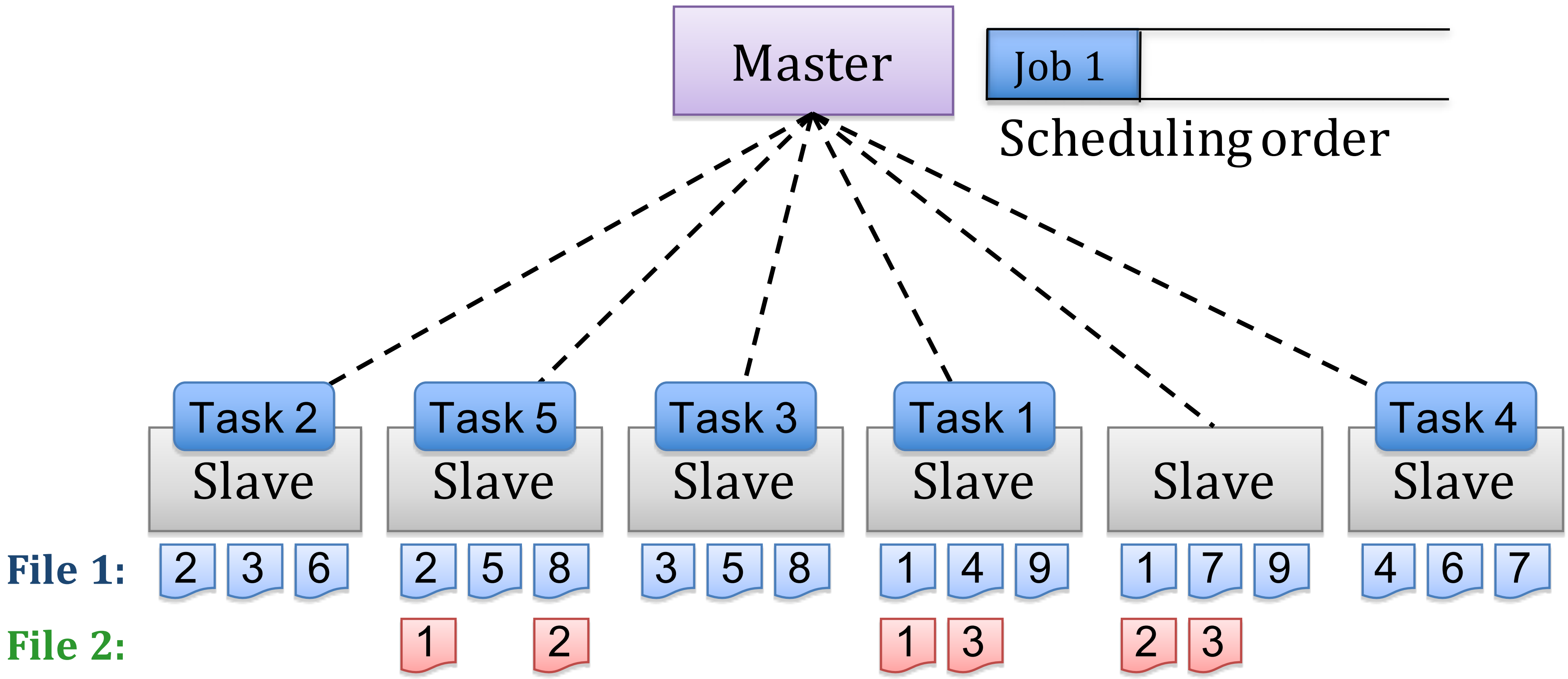

Master
Job 1
Scheduling order
Task 5
Task 1
Task 4
Slave
Slave
Slave
Slave
Slave
Slave
File 1:
2 3 6
2 5 8
3 5 8
1 4 9
1 7 9
4 6 7
File 2:
1 2
1 3
2 3

The Problem

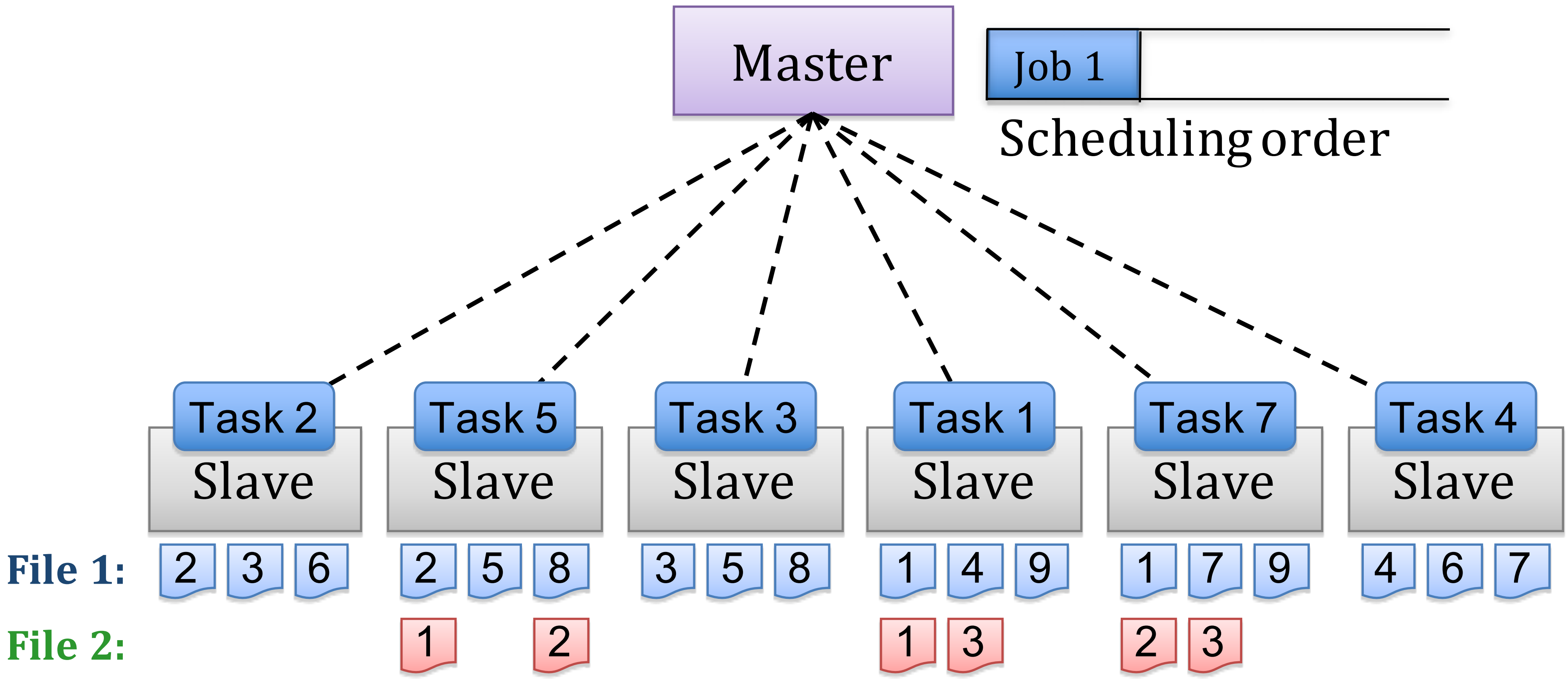


Master
Job 1
Scheduling order
Task 5
Task 3
Task 1
Task 4
Slave
Slave
Slave
Slave
Slave
Slave
File 1:
2 3 6
2 5 8
3 5 8
1 4 9
1 7 9
4 6 7
File 2:
1 2
1 3
2 3

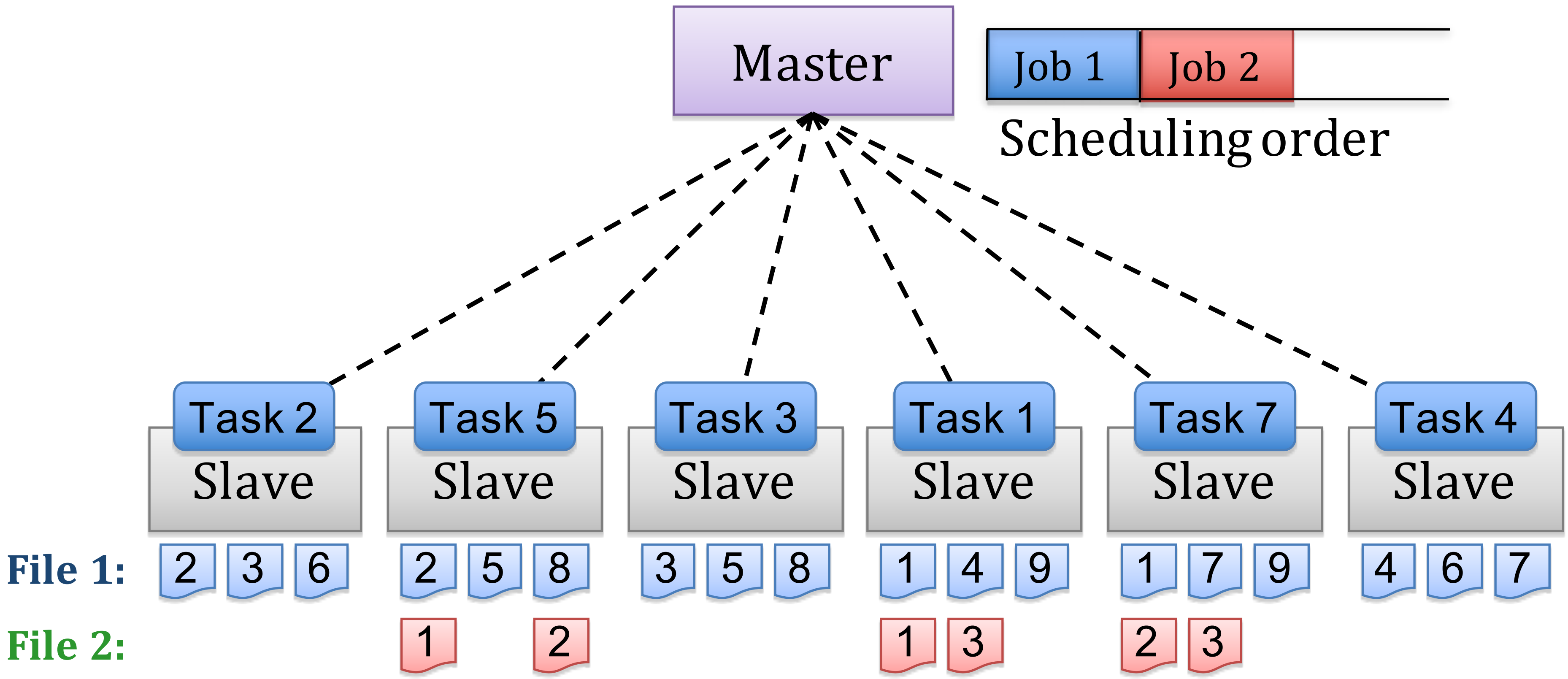
The Problem


Master
Job 1
Scheduling order
Task 2
Slave
Task 5
Slave
Task 3
Slave
Task 1
Slave
Slave
Task 4
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

The Problem

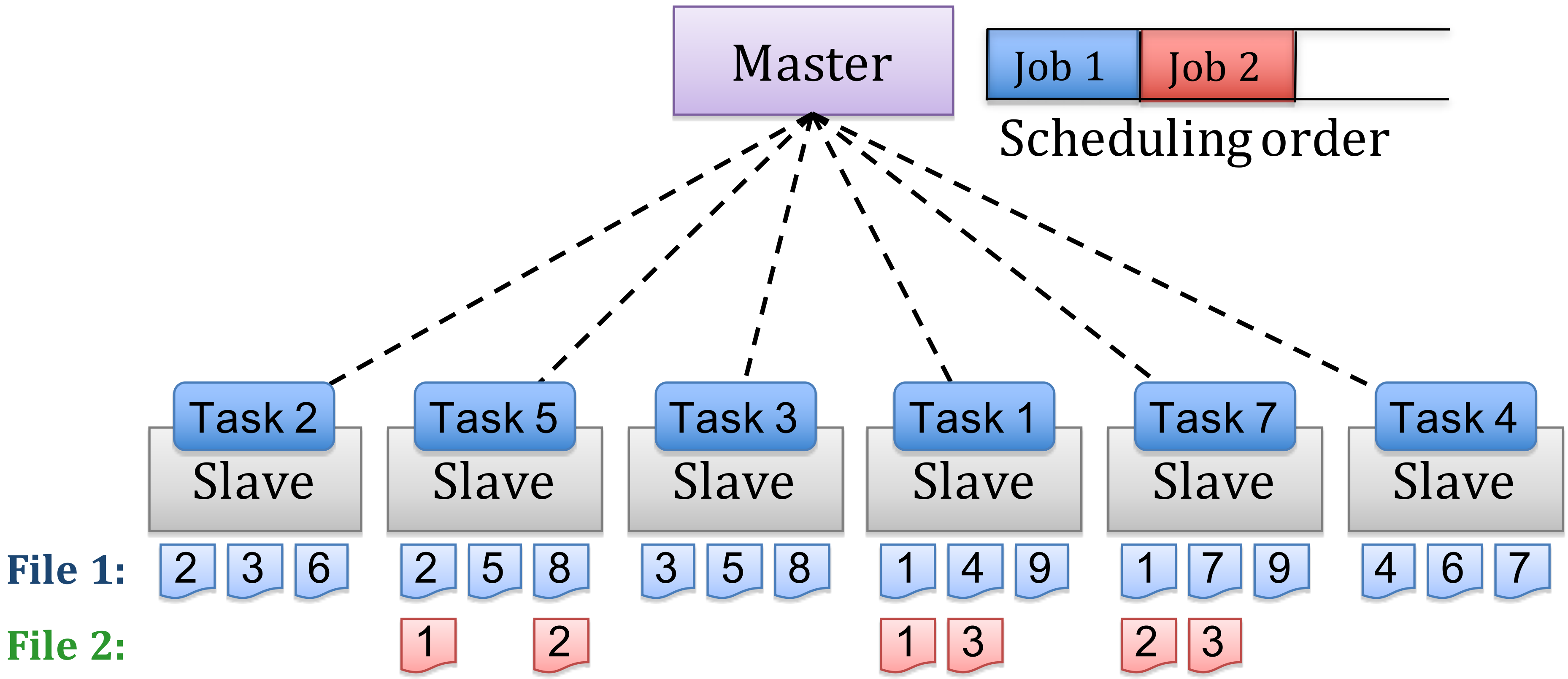

Master
Job 1
Scheduling order
Task 2
Slave
Task 5
Slave
Task 3
Slave
Task 1
Slave
Task 7
Slave
Task 4
Slave
File 1:
2 3 6
2 5 8
3 5 8
1 4 9
1 7 9
4 6 7
File 2:
1 2
1 3
2 3

The Problem



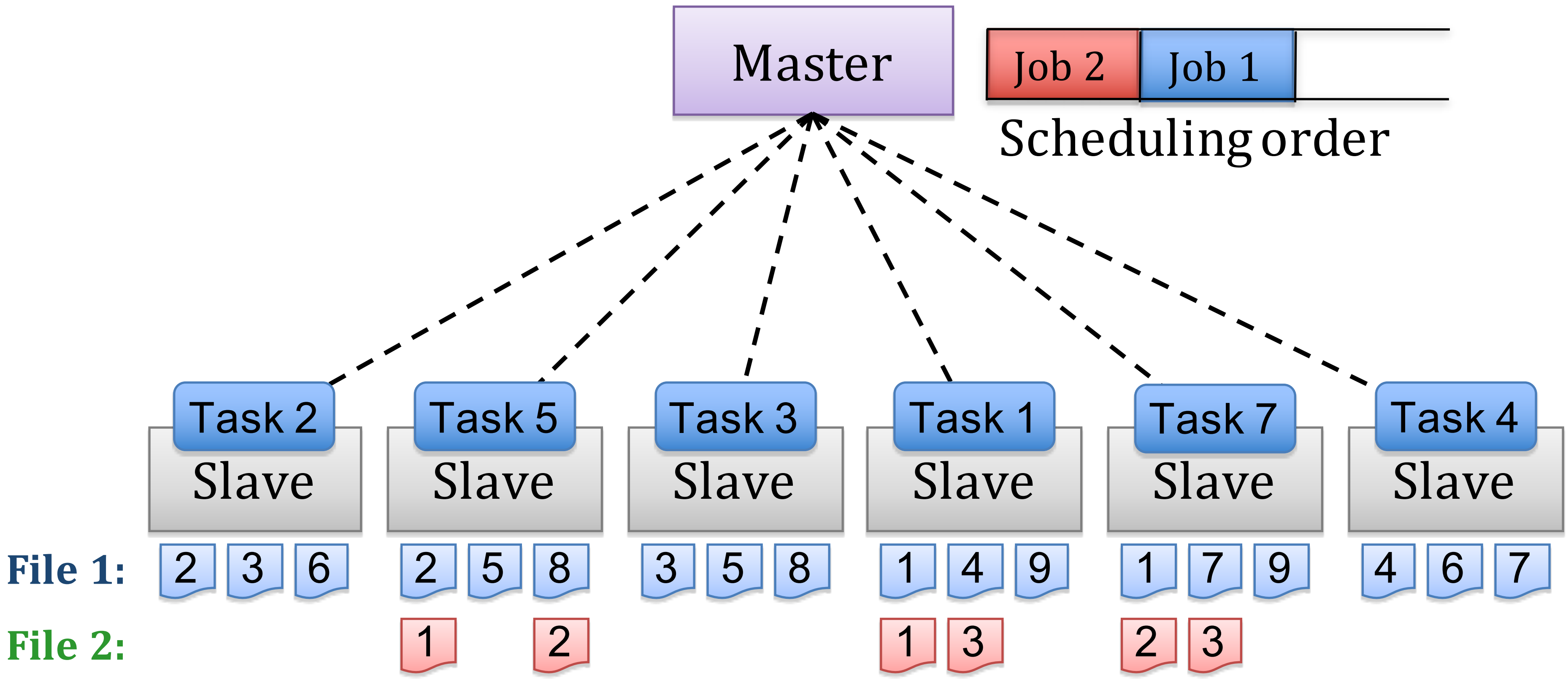
Master
Job 1
Job 2
Scheduling order
Task 2
Slave
Task 5
Slave
Task 3
Slave
Task 1
Slave
Task 7
Slave
Task 4
Slave
File 1:
2 3 6
2 5 8
3 5 8
1 4 9
1 7 9
4 6 7
File 2:
1 2
1 3
2 3

The Problem

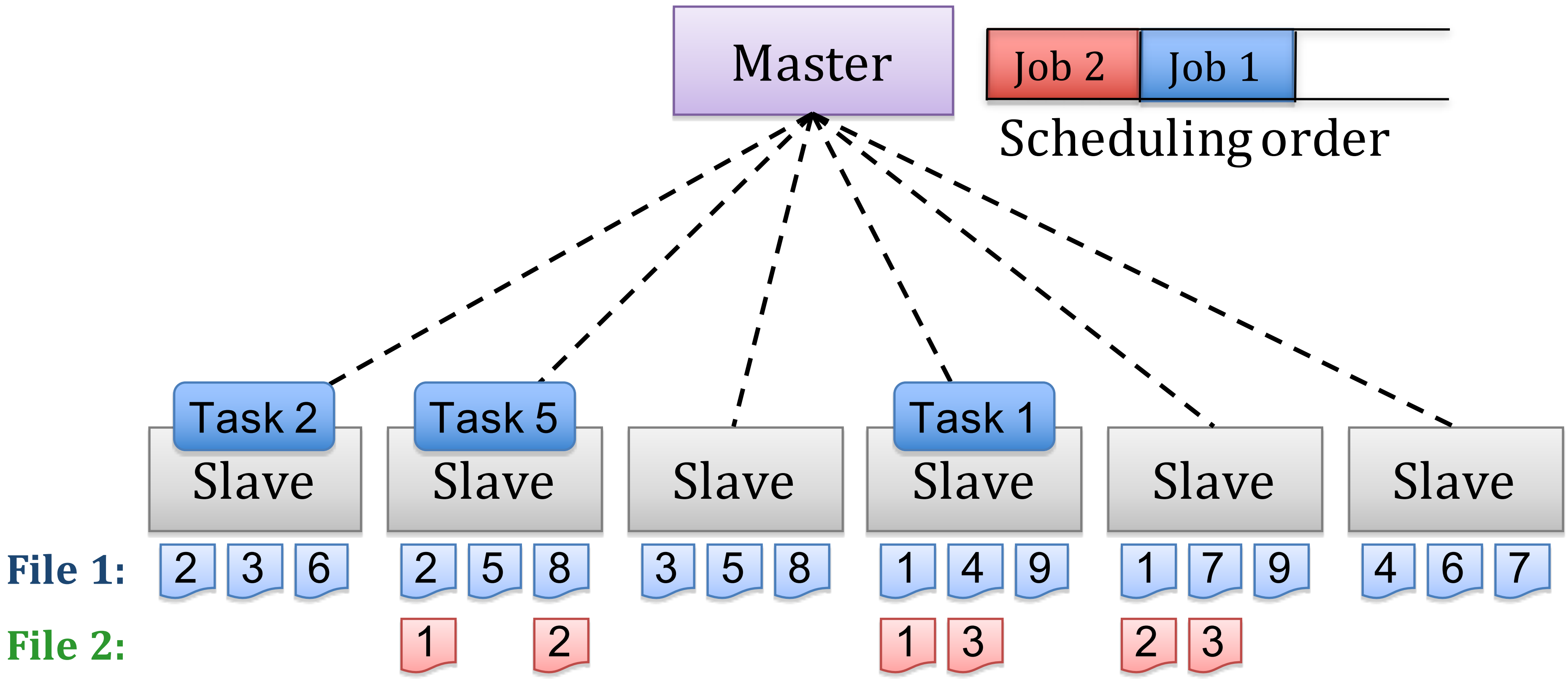


Master
Job 1
Job 2
Scheduling order
Task 2
Slave
Task 5
Slave
Task 3
Slave
Task 1
Slave
Task 7
Slave
Task 4
Slave
File 1:
2 3 6
2 5 8
3 5 8
1 4 9
1 7 9
4 6 7
File 2:
1 2
1 3
2 3

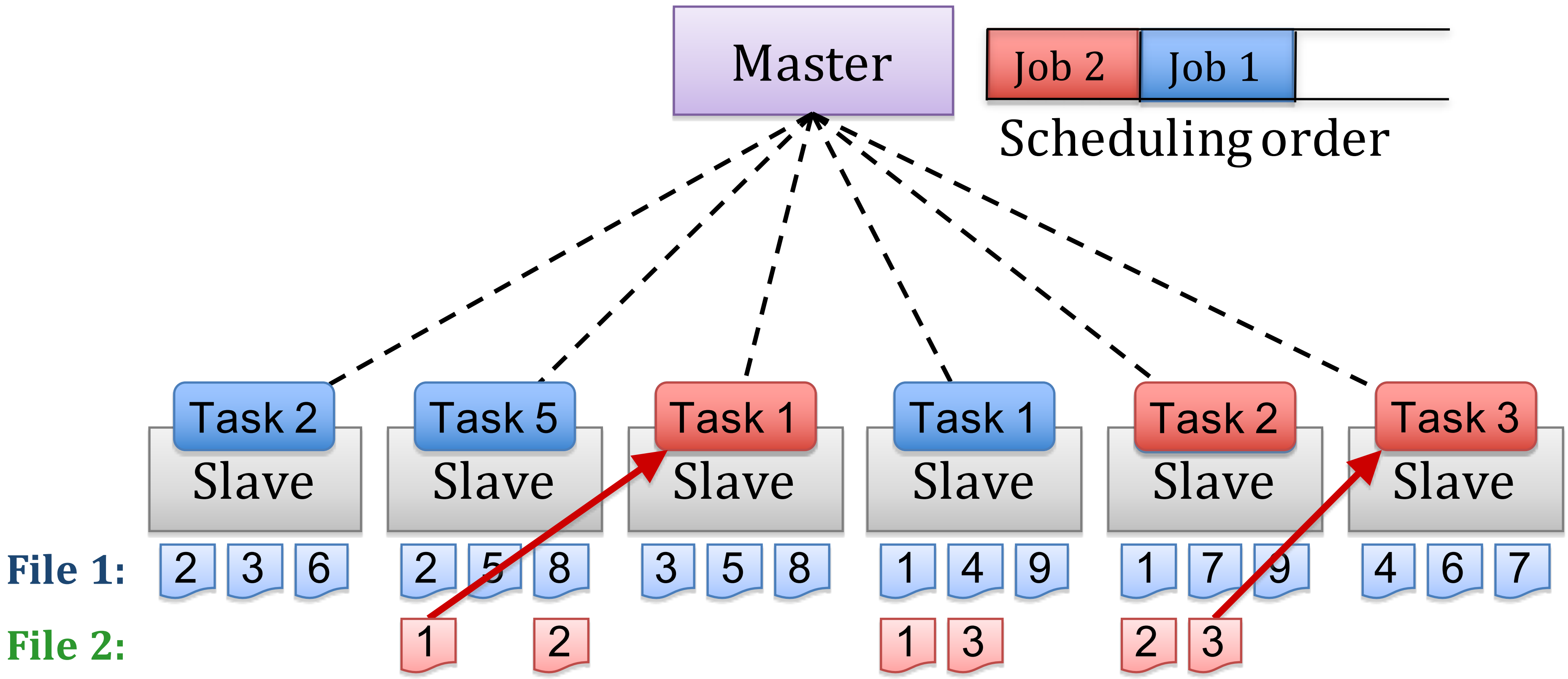
The Problem


Master
Job 2
Job 1
Scheduling order
Task 2
Task 5
Task 3
Task 1
Task 7
Task 4
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

The Problem


Master
Job 2
Job 1
Scheduling order
Task 2
Task 5
Task 1
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

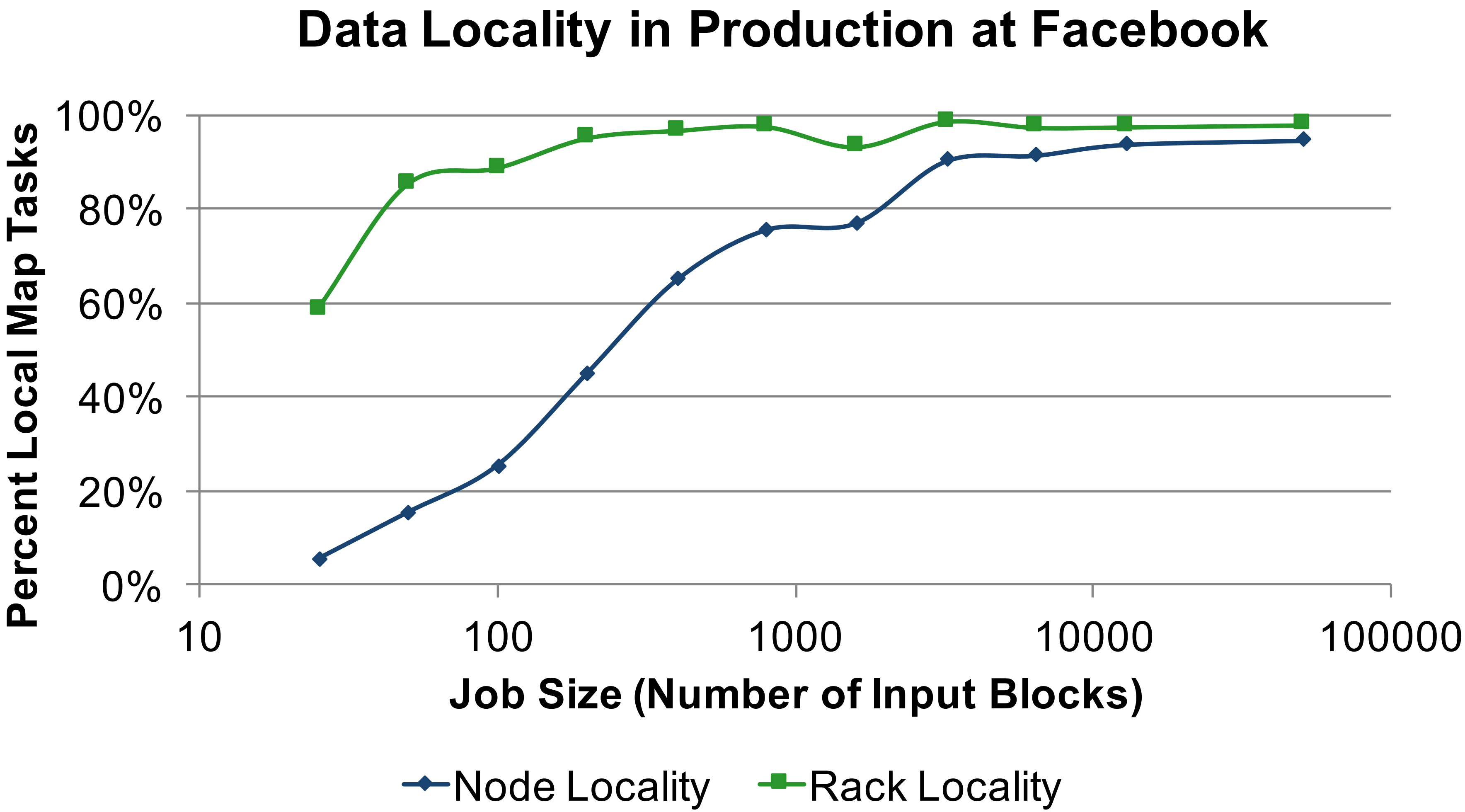
The Problem



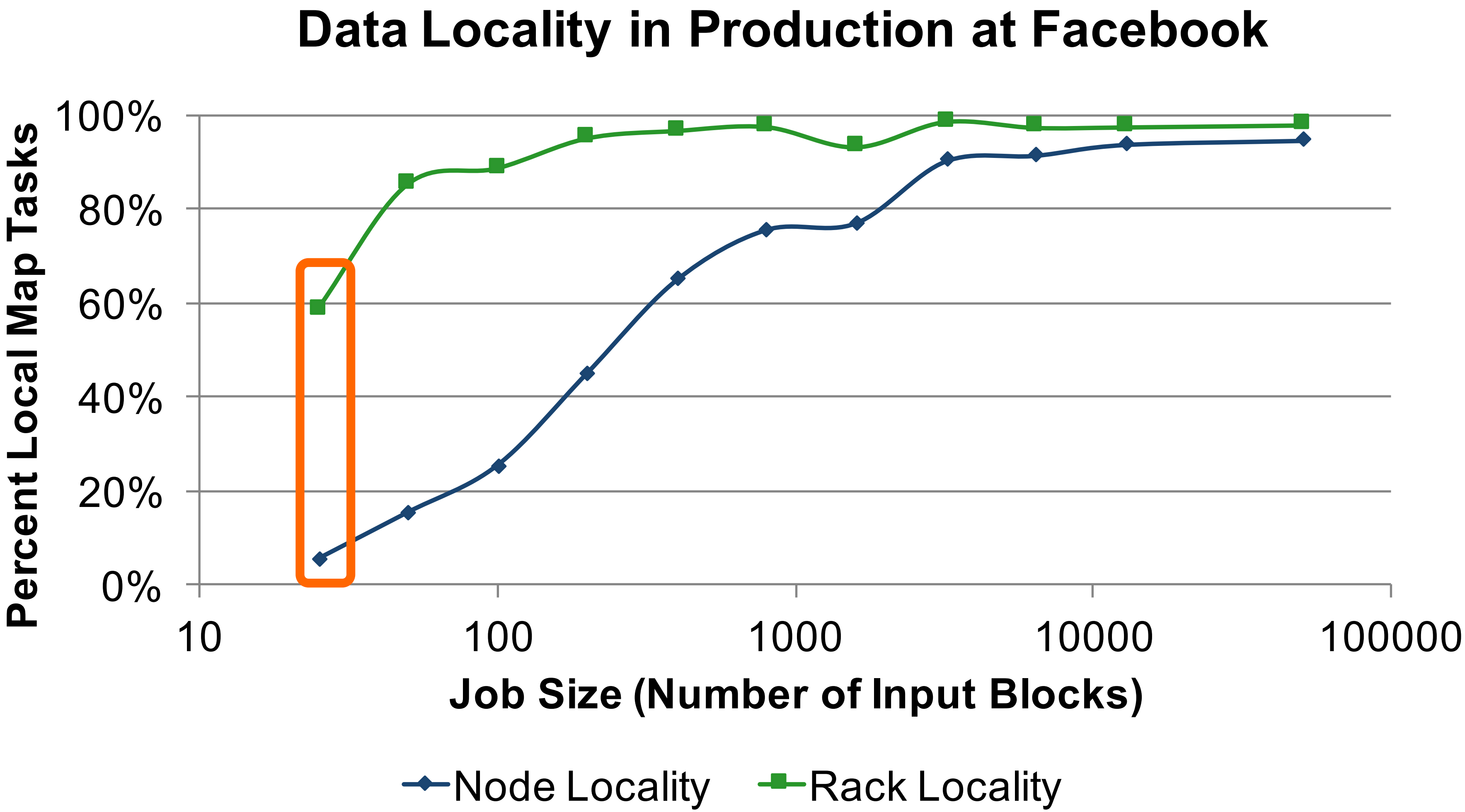
Problem: Fair decision hurts locality

Especially bad for jobs with small input files

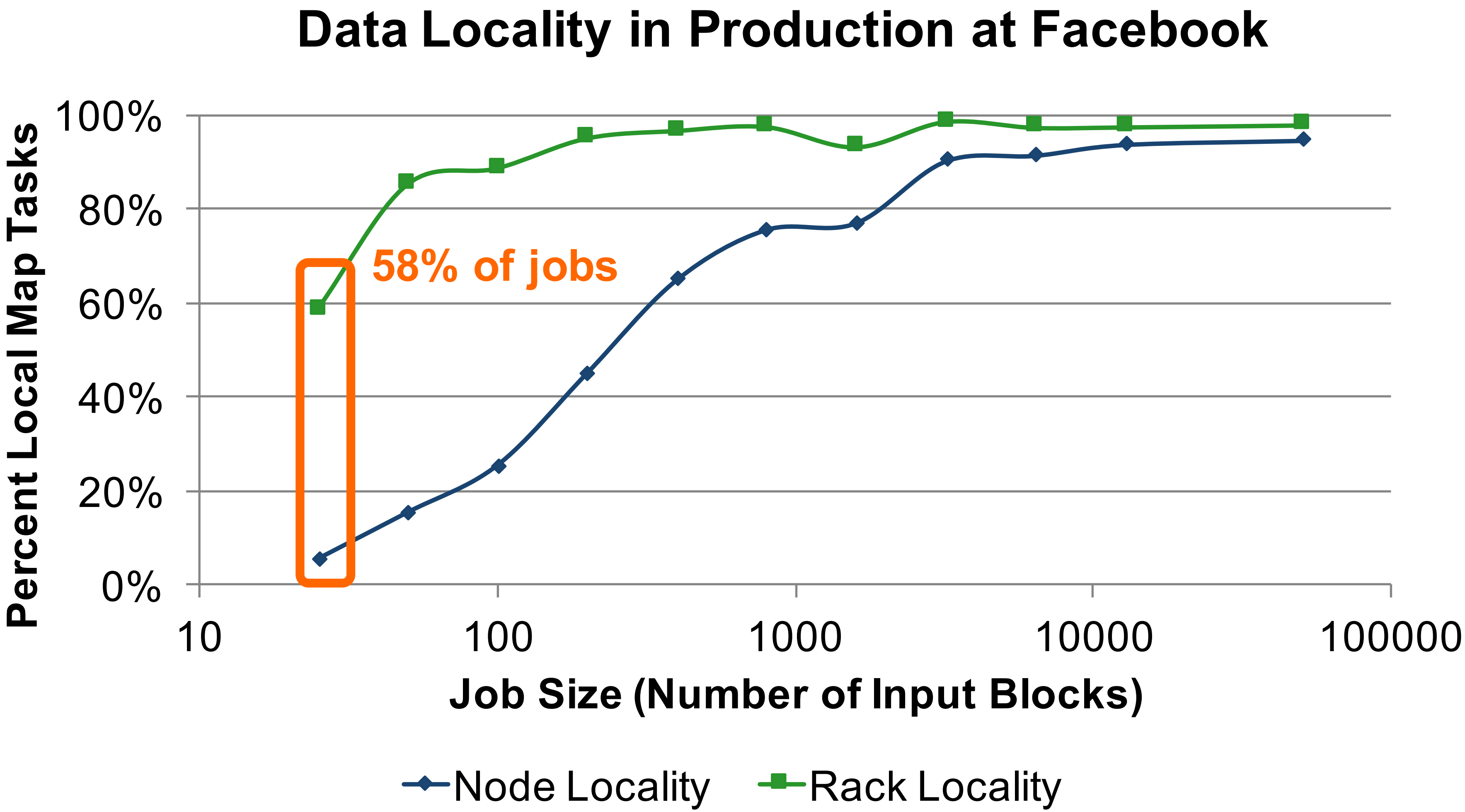
Locality vs. Job Size at Facebook



Locality vs. Job Size at Facebook

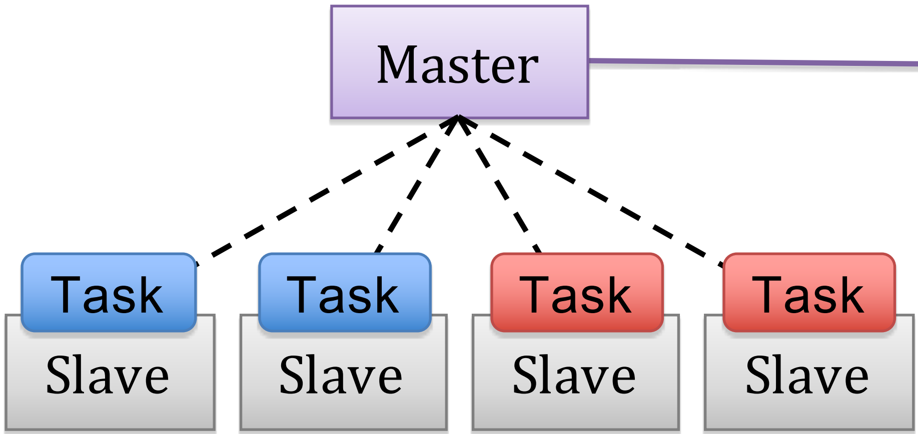


Locality vs. Job Size at Facebook



Special Instance: Sticky Slots

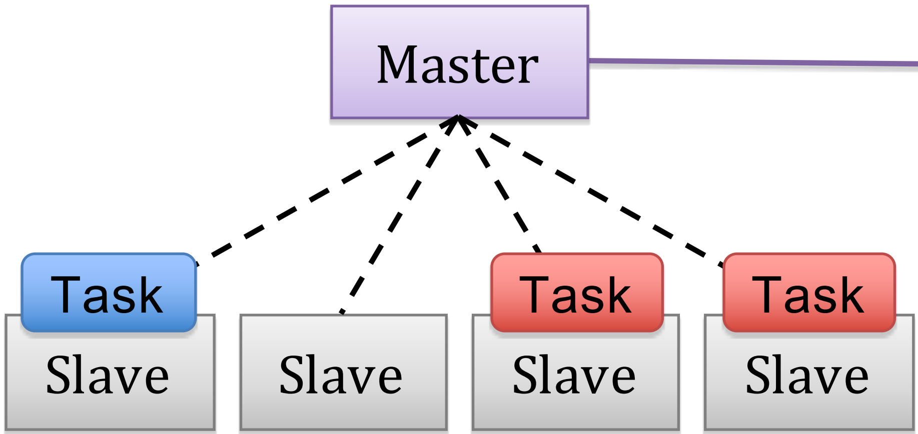
- Under fair sharing, locality can be poor even when all jobs have large input files
- Problem: jobs get "stuck" in the same set of task slots
  - When one a task in job *j* finishes, the slot it was running in is given back to *j*, because *j* is below its share
  - Bad because data files are spread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



Job	Fair Share	Running Tasks
Job 1	2	2
Job 2	2	2

Special Instance: Sticky Slots

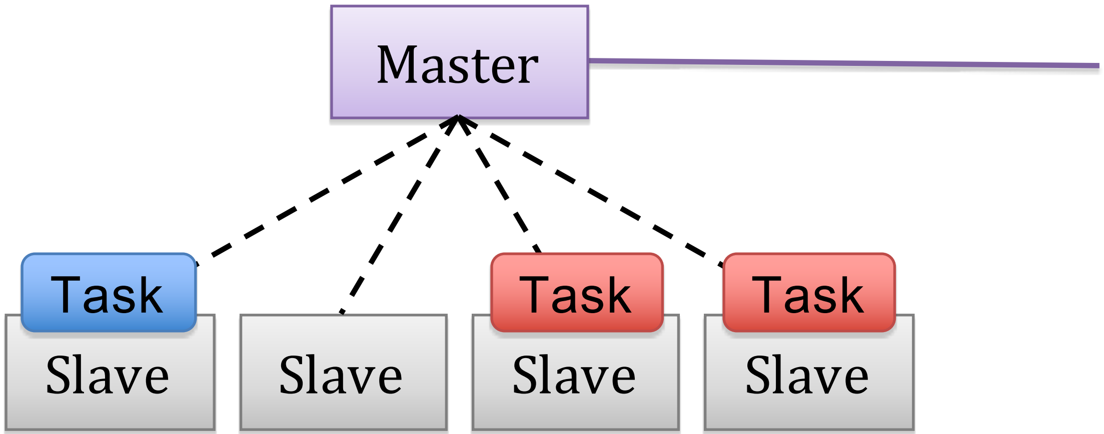
- Under fair sharing, locality can be poor even when all jobs have large input files
- Problem: jobs get "stuck" in the same set of task slots
  - When one a task in job *j* finishes, the slot it was running in is given back to *j*, because *j* is below its share
  - Bad because data files are spread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



Job	Fair Share	Running Tasks
Job 1	2	2
Job 2	2	2

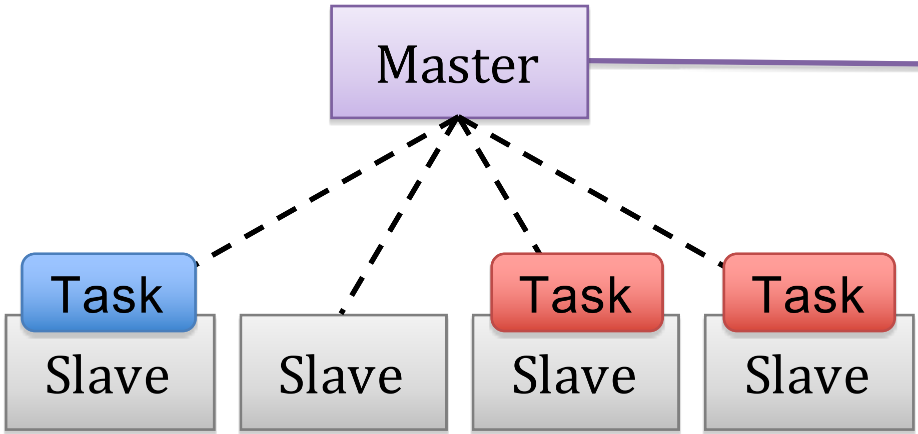
Special Instance: Sticky Slots

- 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



Special Instance: Sticky Slots

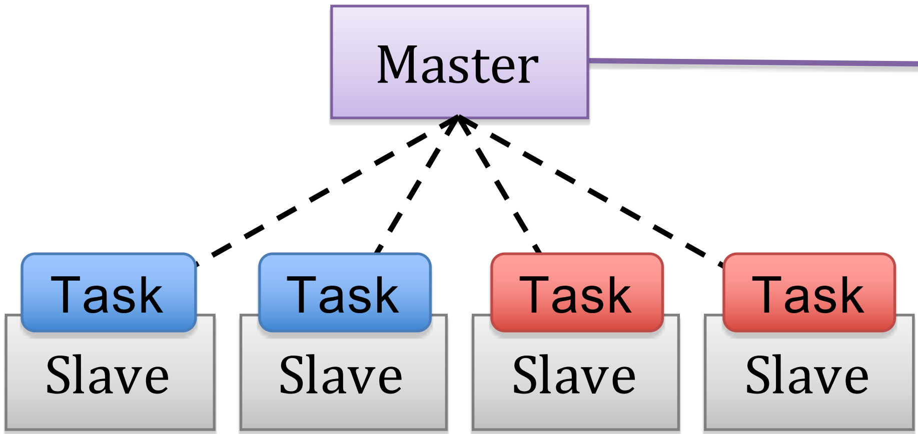
- Under fair sharing, locality can be poor even when all jobs have large input files
- Problem: jobs get "stuck" in the same set of task slots
  - When one a task in job *j* finishes, the slot it was running in is given back to *j*, because *j* is below its share
  - Bad because data files are spread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



Job	Fair Share	Running Tasks
Job 1	2	**1**
Job 2	2	2

Special Instance: Sticky Slots

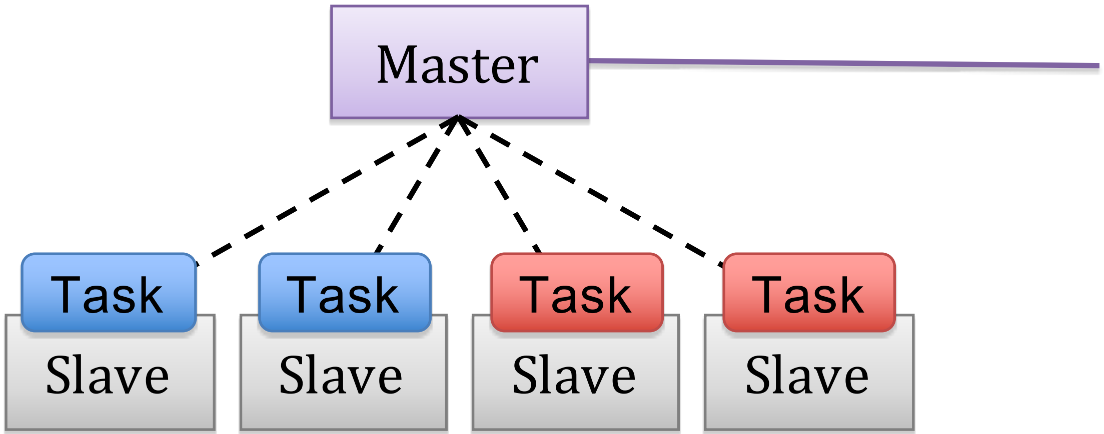
- Under fair sharing, locality can be poor even when all jobs have large input files
- Problem: jobs get "stuck" in the same set of task slots
  - When one a task in job *j* finishes, the slot it was running in is given back to *j*, because *j* is below its share
  - Bad because data files are spread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



Job	Fair Share	Running Tasks
Job 1	2	**1**
Job 2	2	2

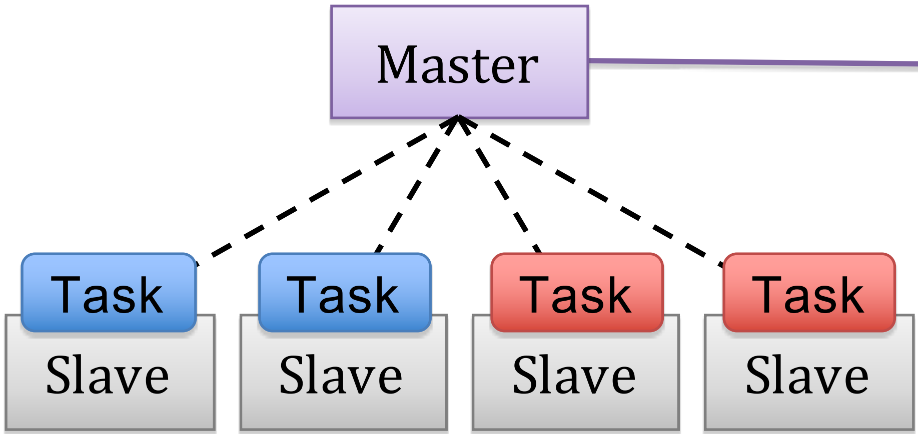
Special Instance: Sticky Slots

- 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



Special Instance: Sticky Slots

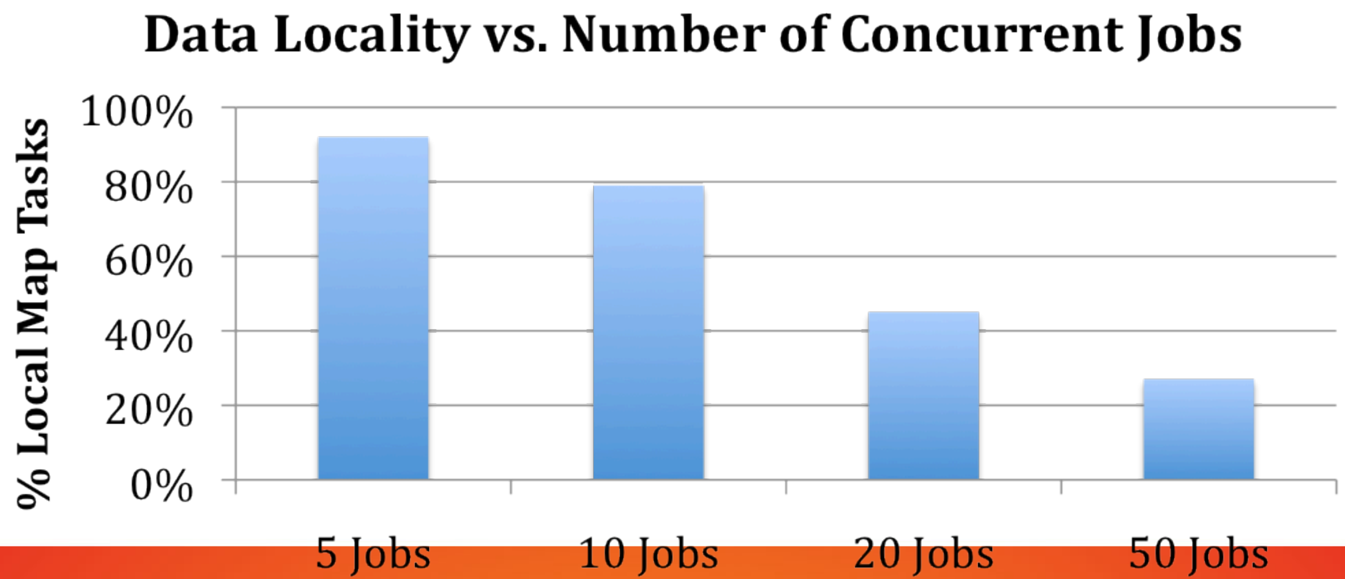
- Under fair sharing, locality can be poor even when all jobs have large input files
- Problem: jobs get "stuck" in the same set of task slots
  - When one a task in job *j* finishes, the slot it was running in is given back to *j*, because *j* is below its share
  - Bad because data files are spread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



Job	Fair Share	Running Tasks
Job 1	2	2
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

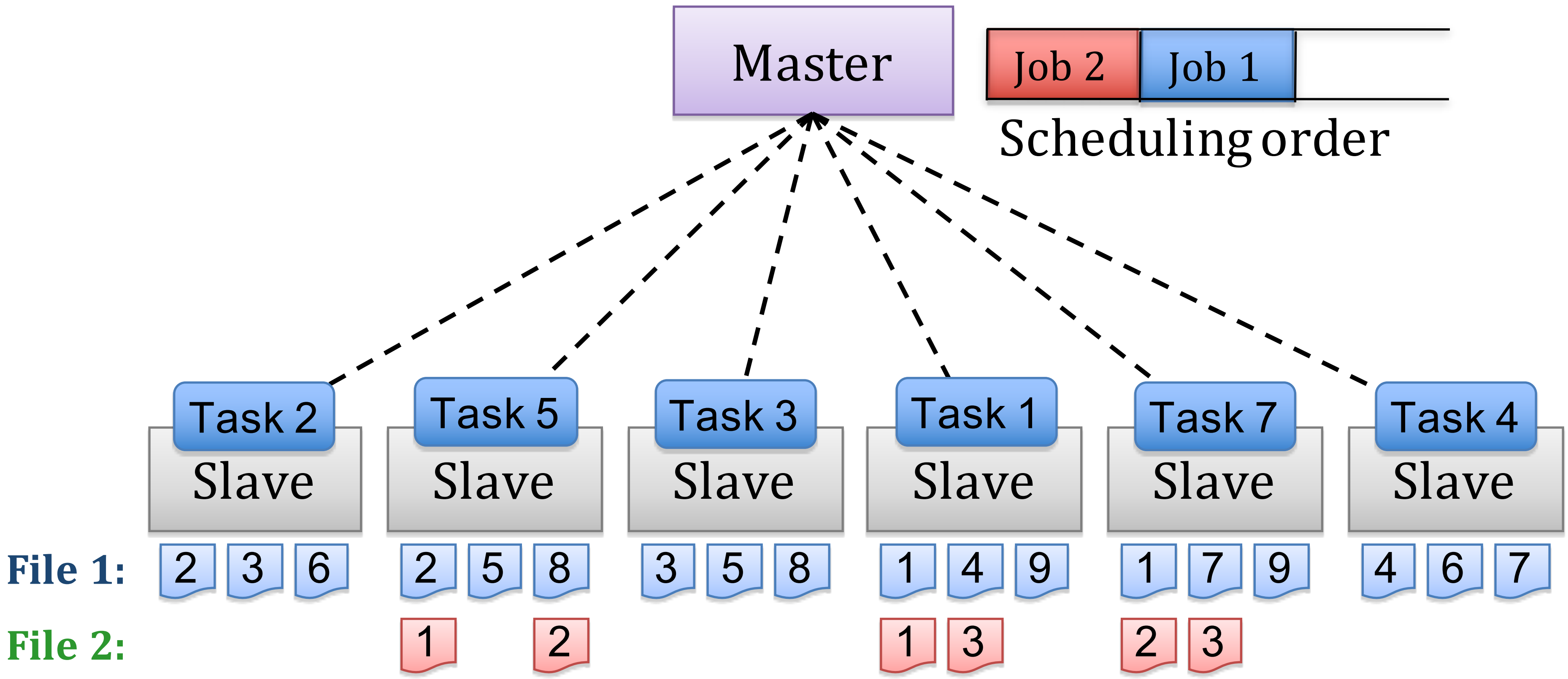


Solution: Delay Scheduling

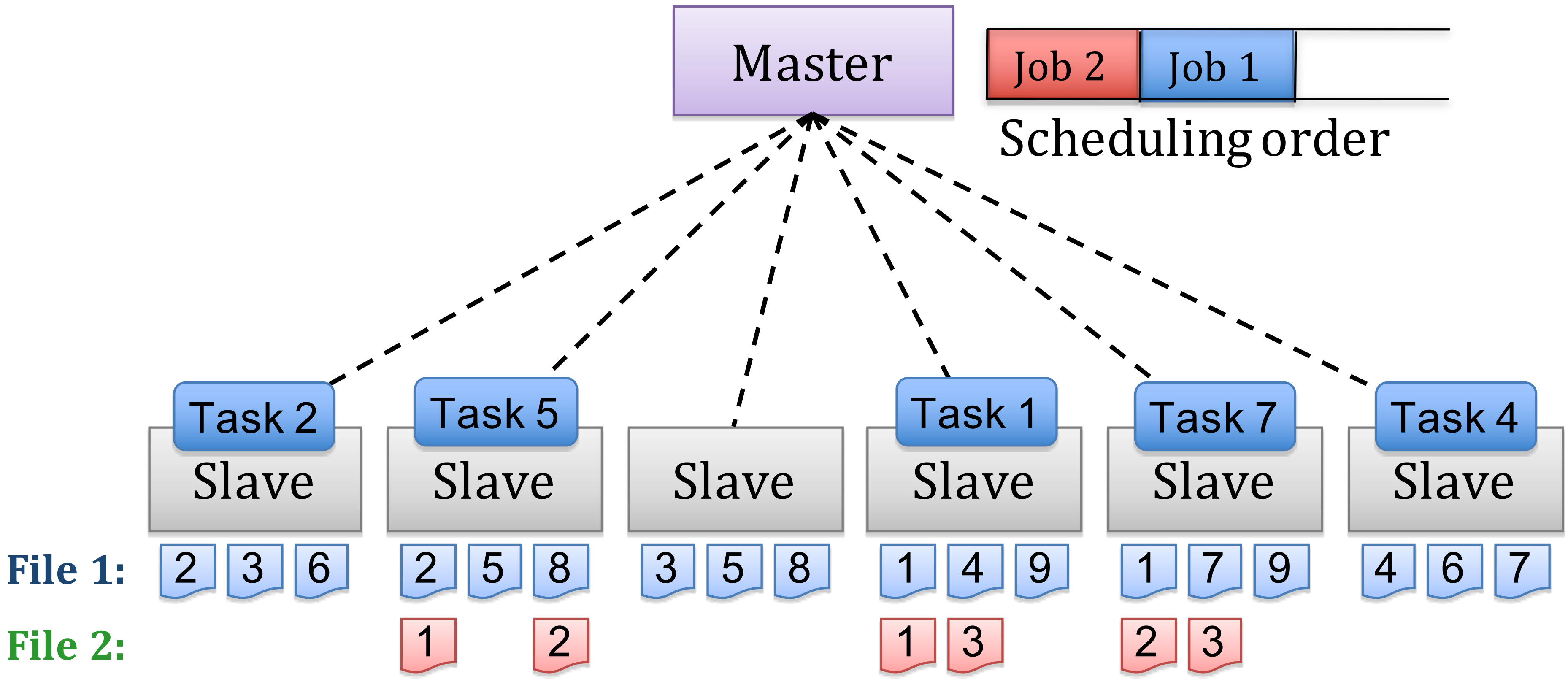
- *Relax queuing policy* to make jobs wait for a limited time if they cannot launch local tasks

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

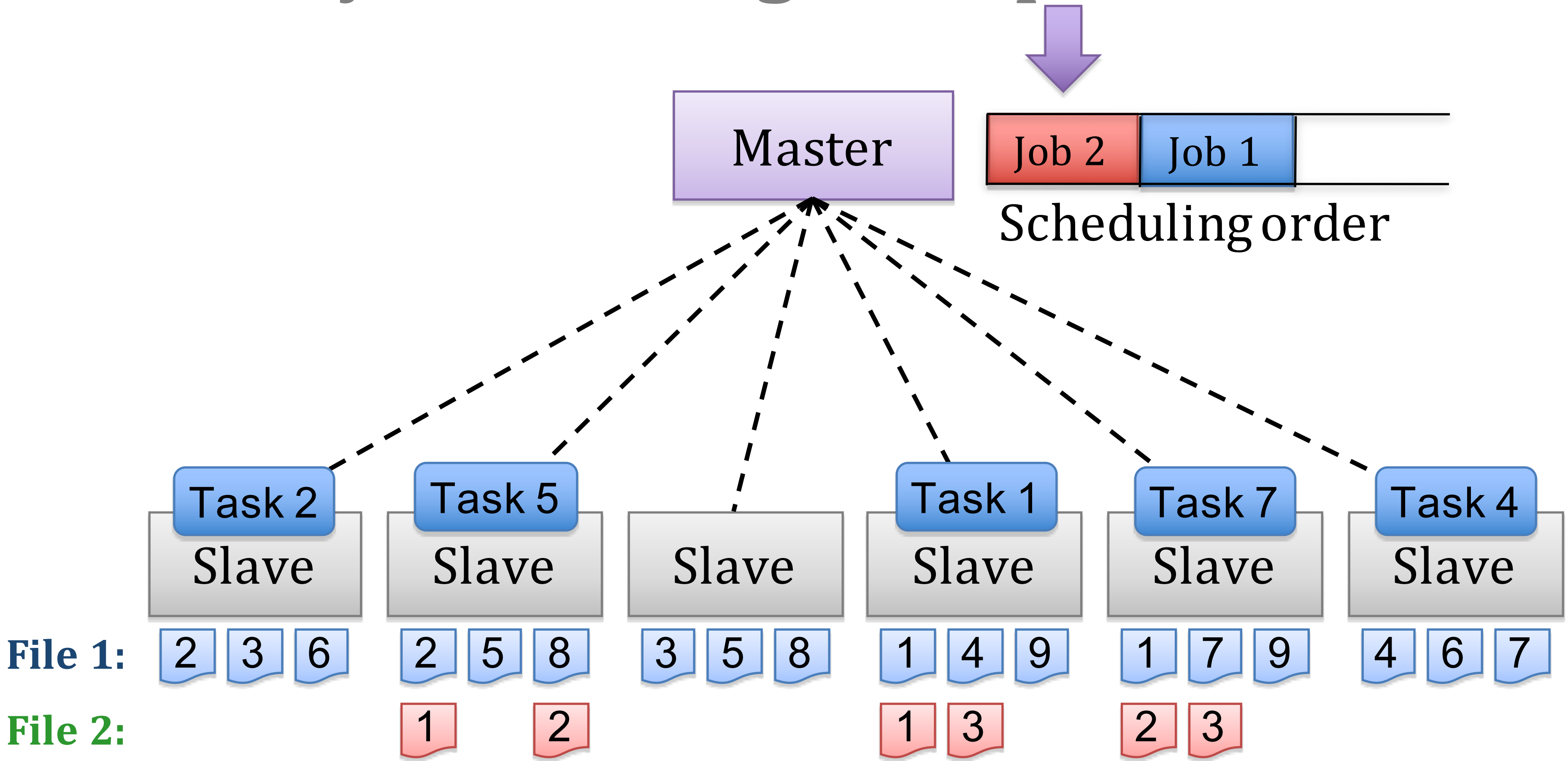
Delay Scheduling Example



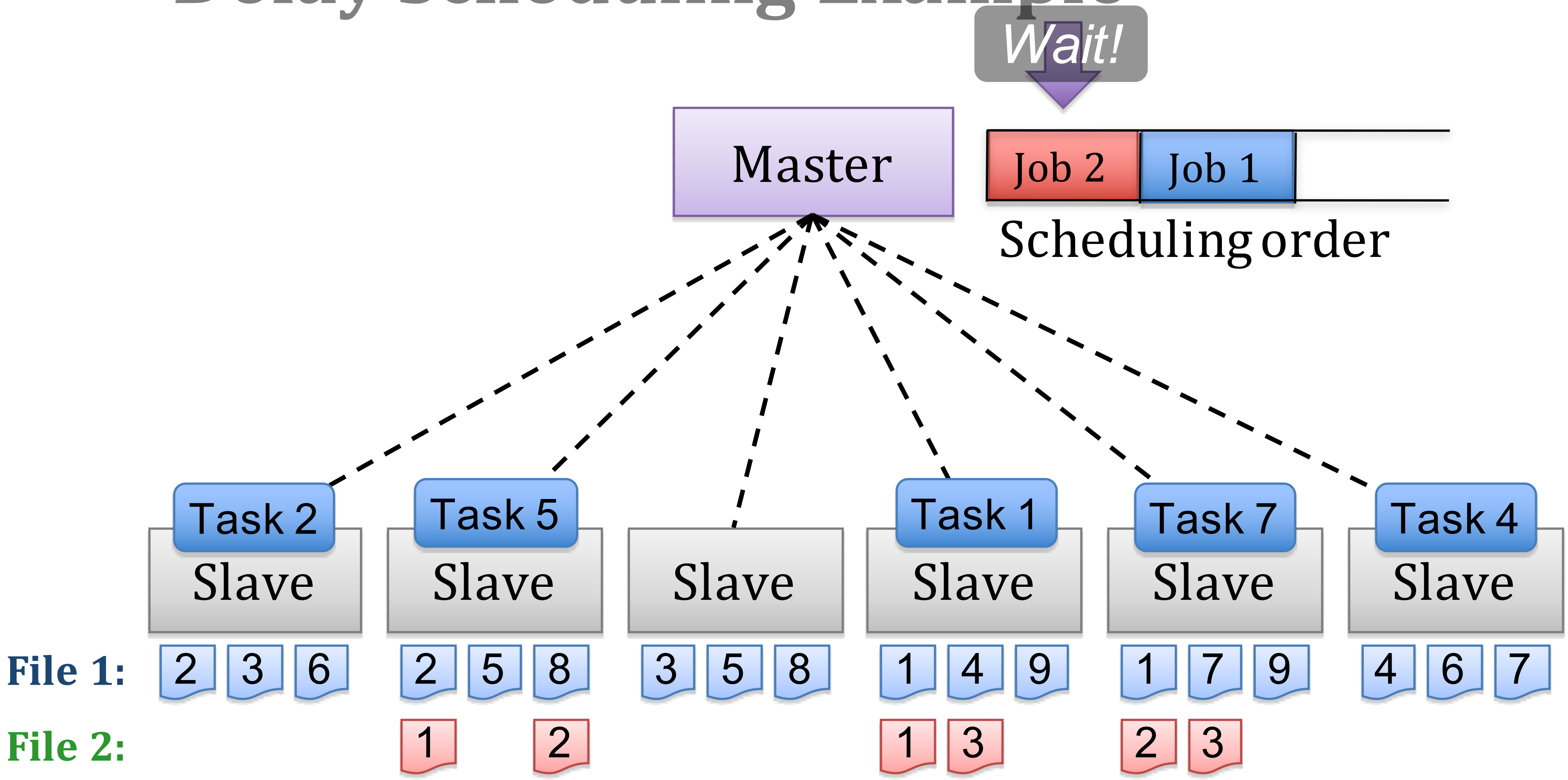
Delay Scheduling Example



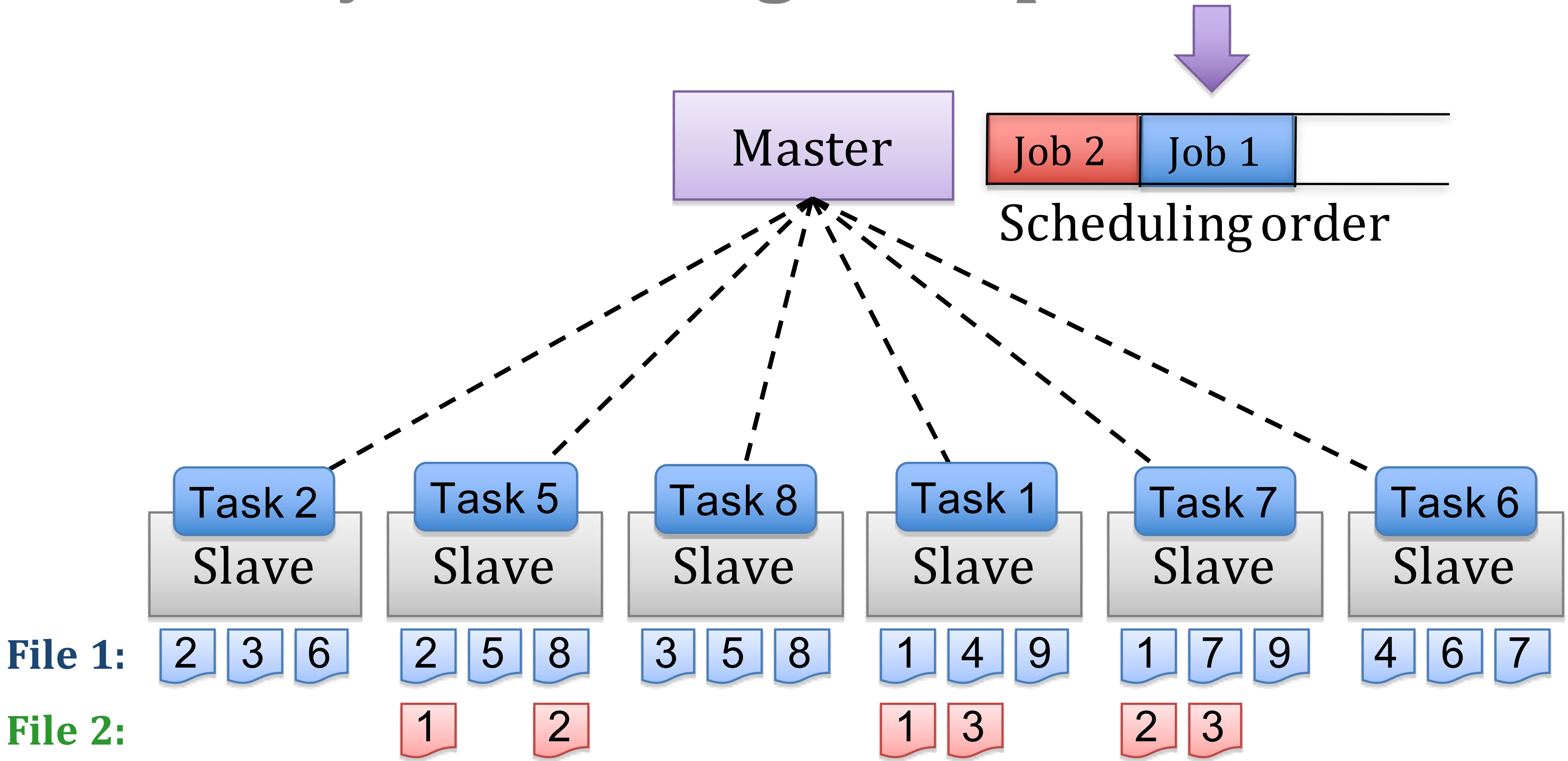
Delay Scheduling Example



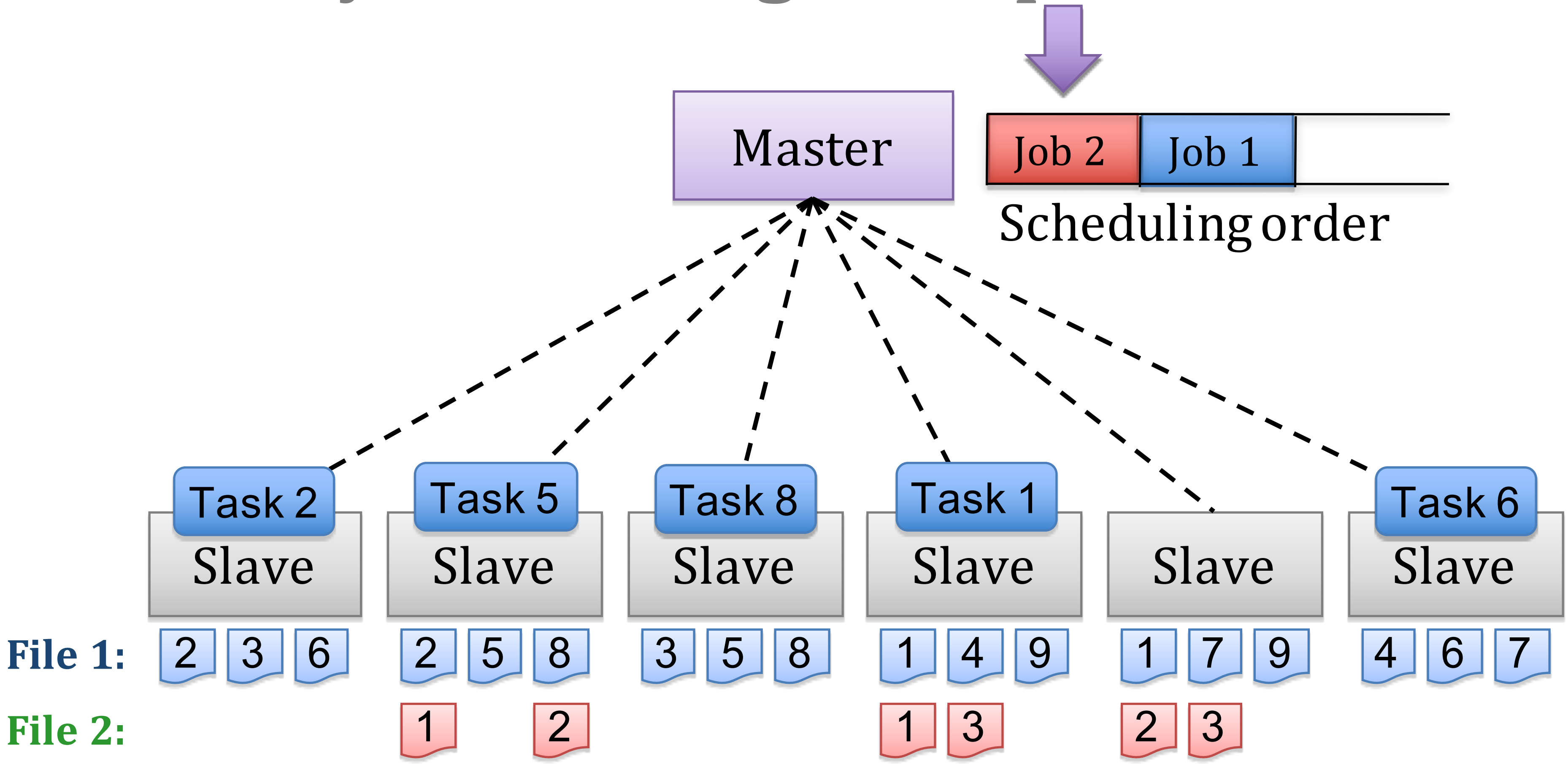
Delay Scheduling Example



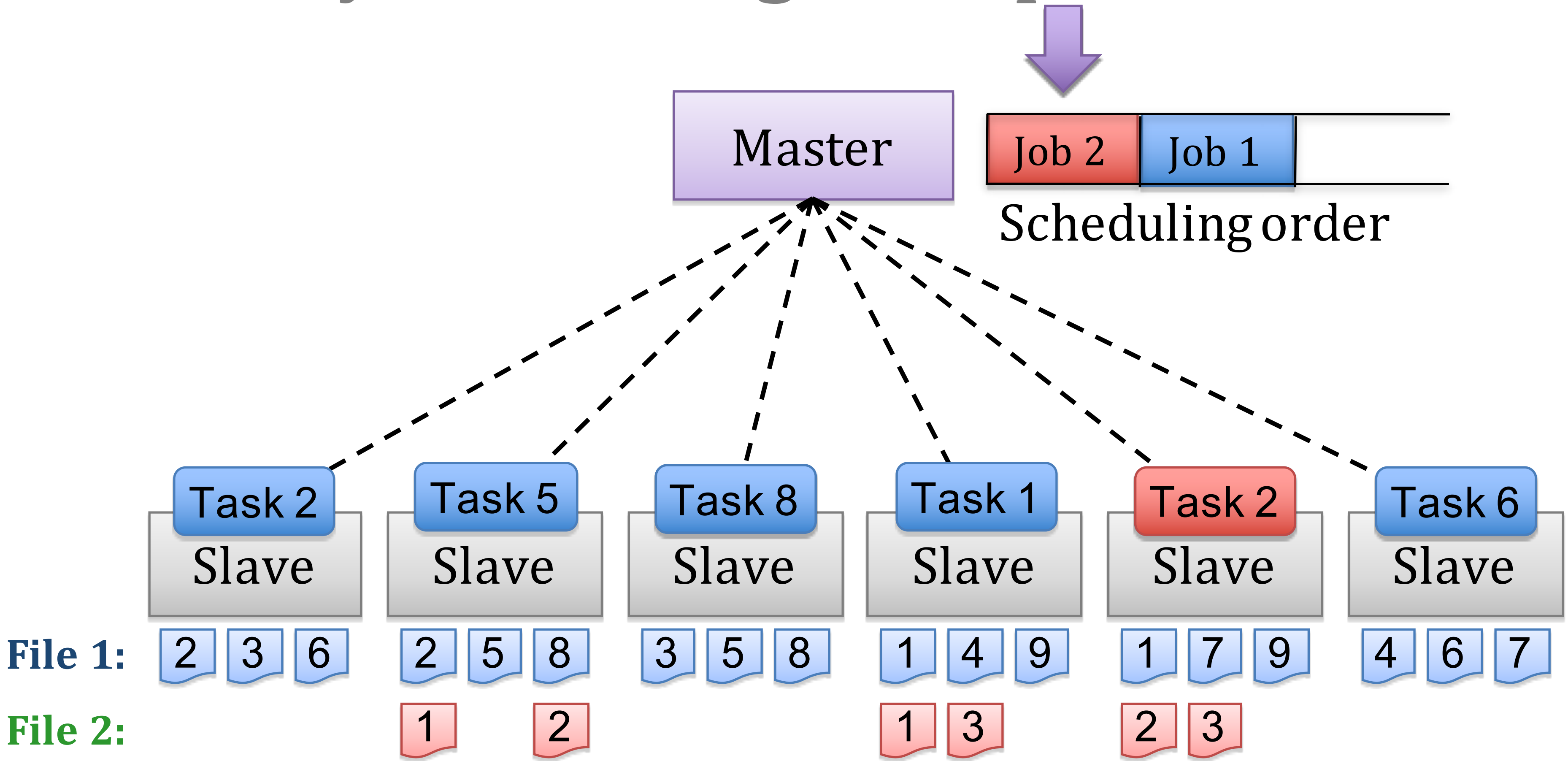
Delay Scheduling Example



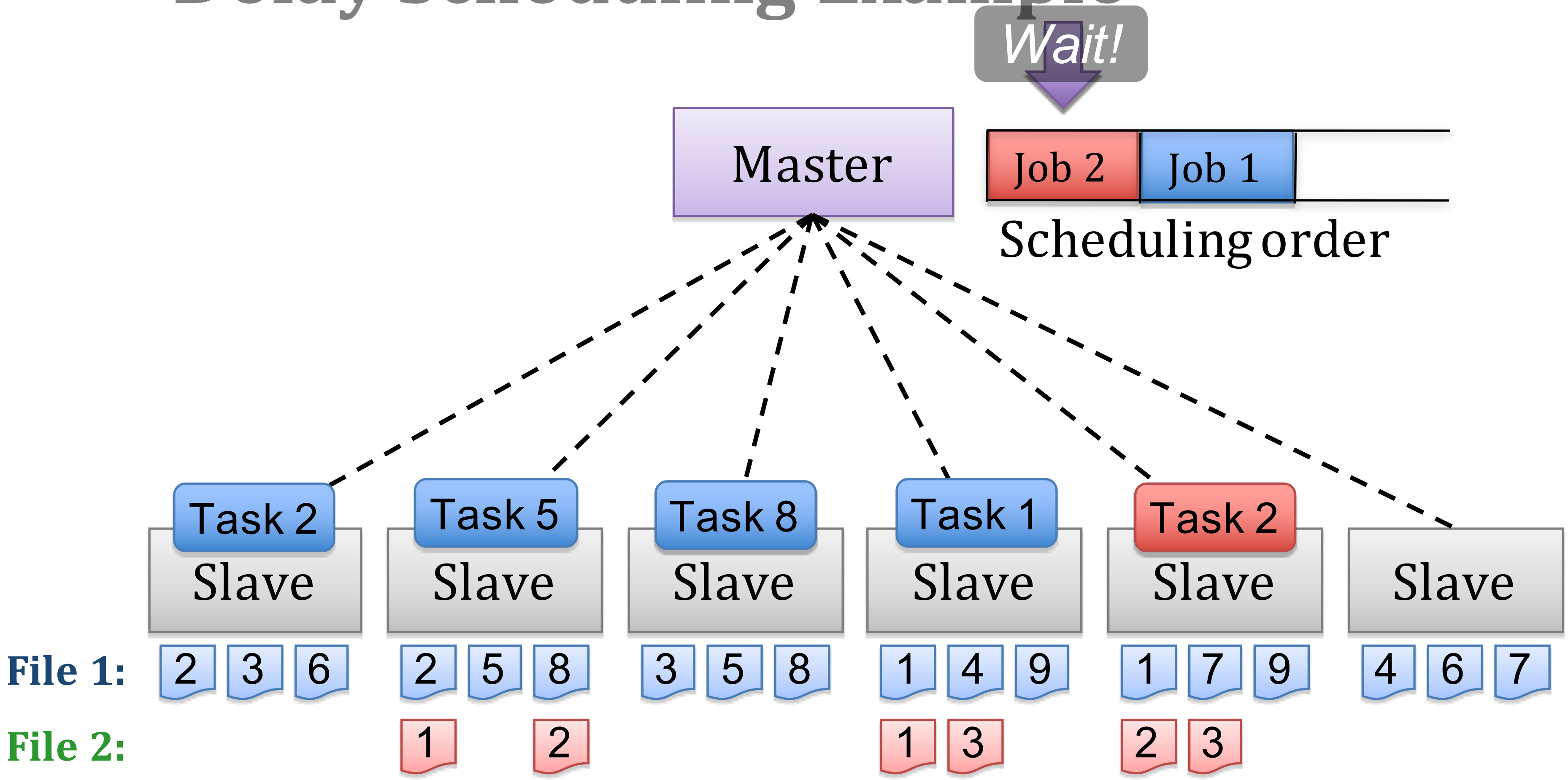
Delay Scheduling Example



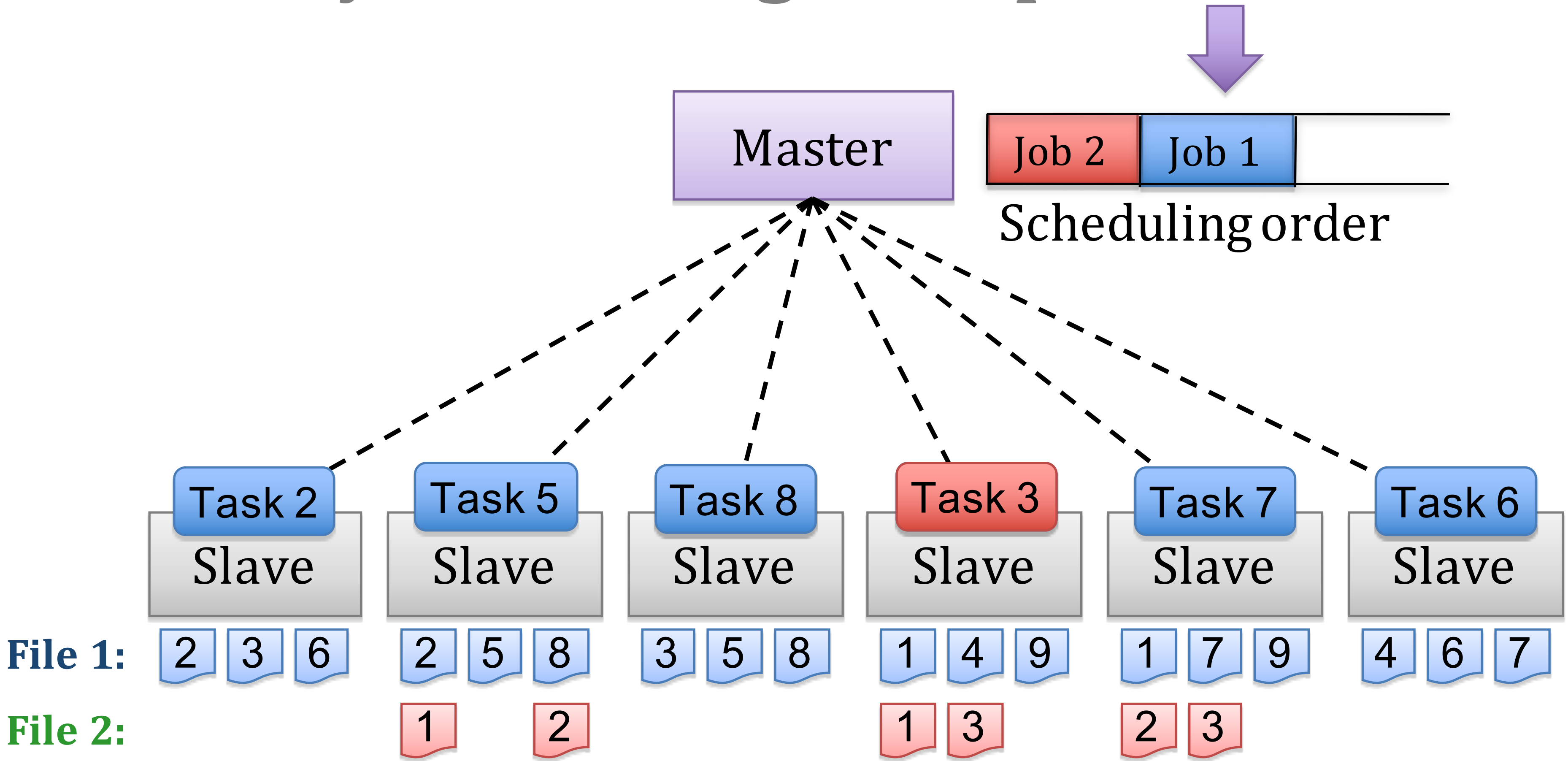
Delay Scheduling Example



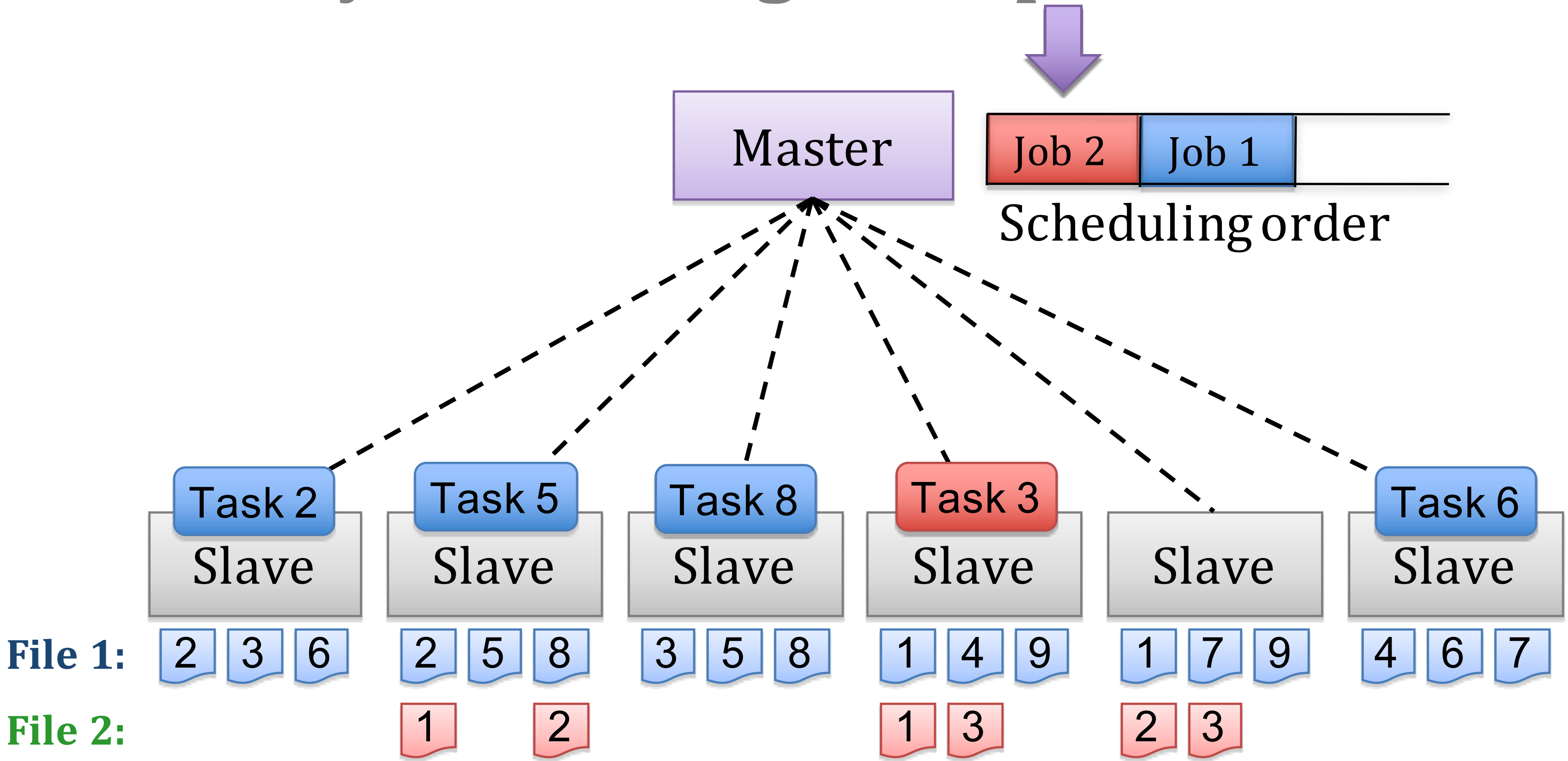
Delay Scheduling Example



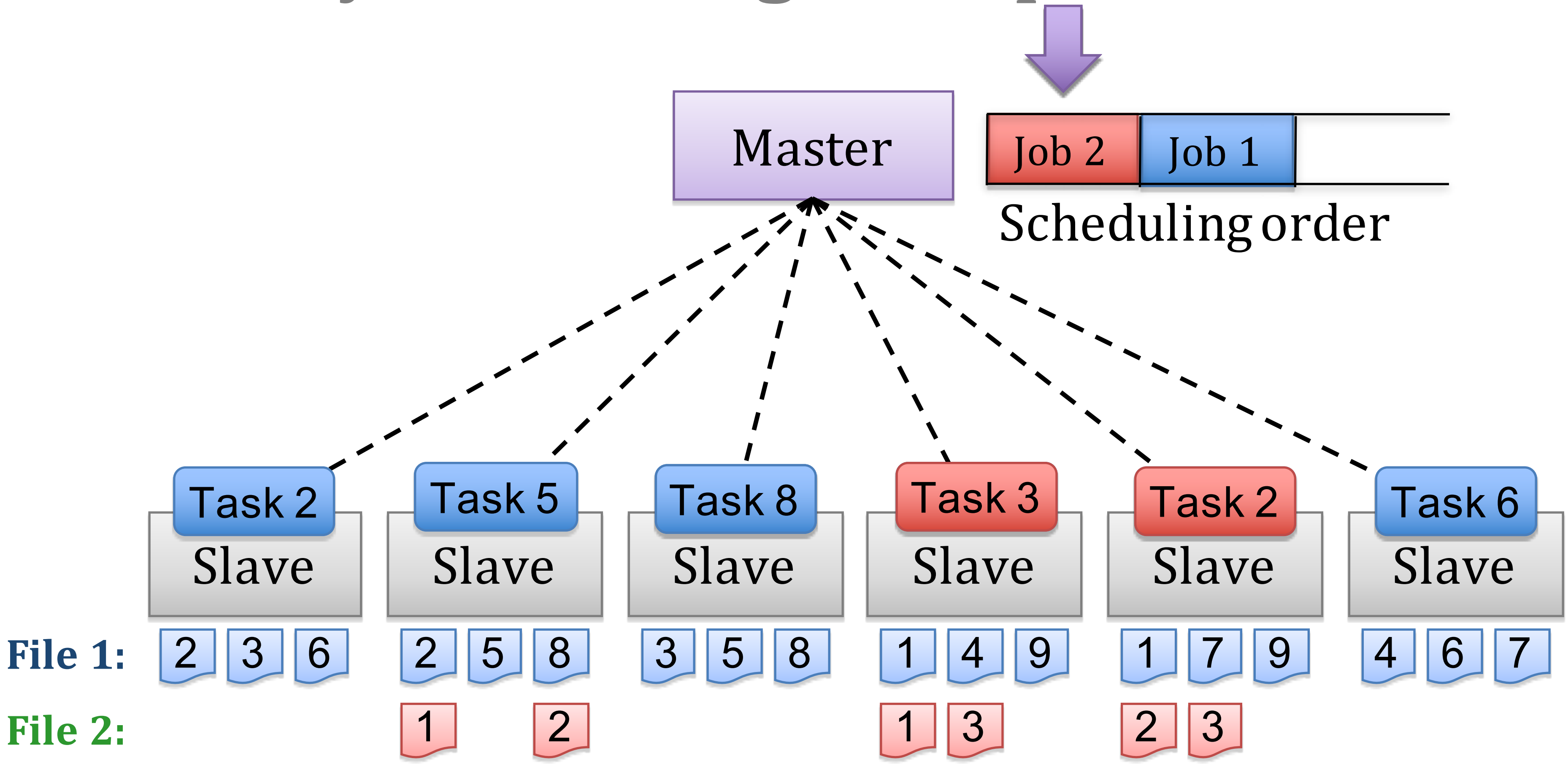
Delay Scheduling Example



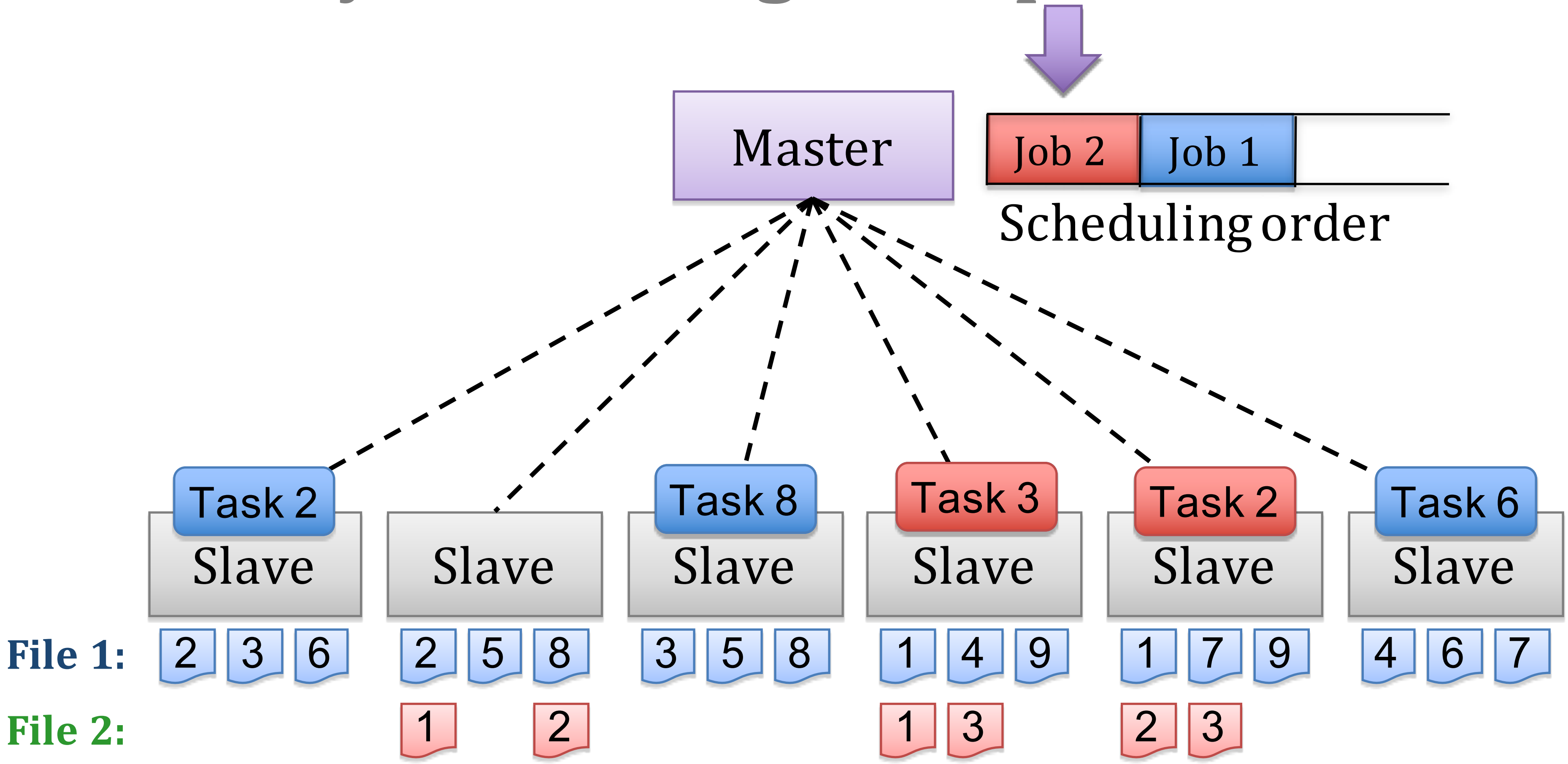
Delay Scheduling Example



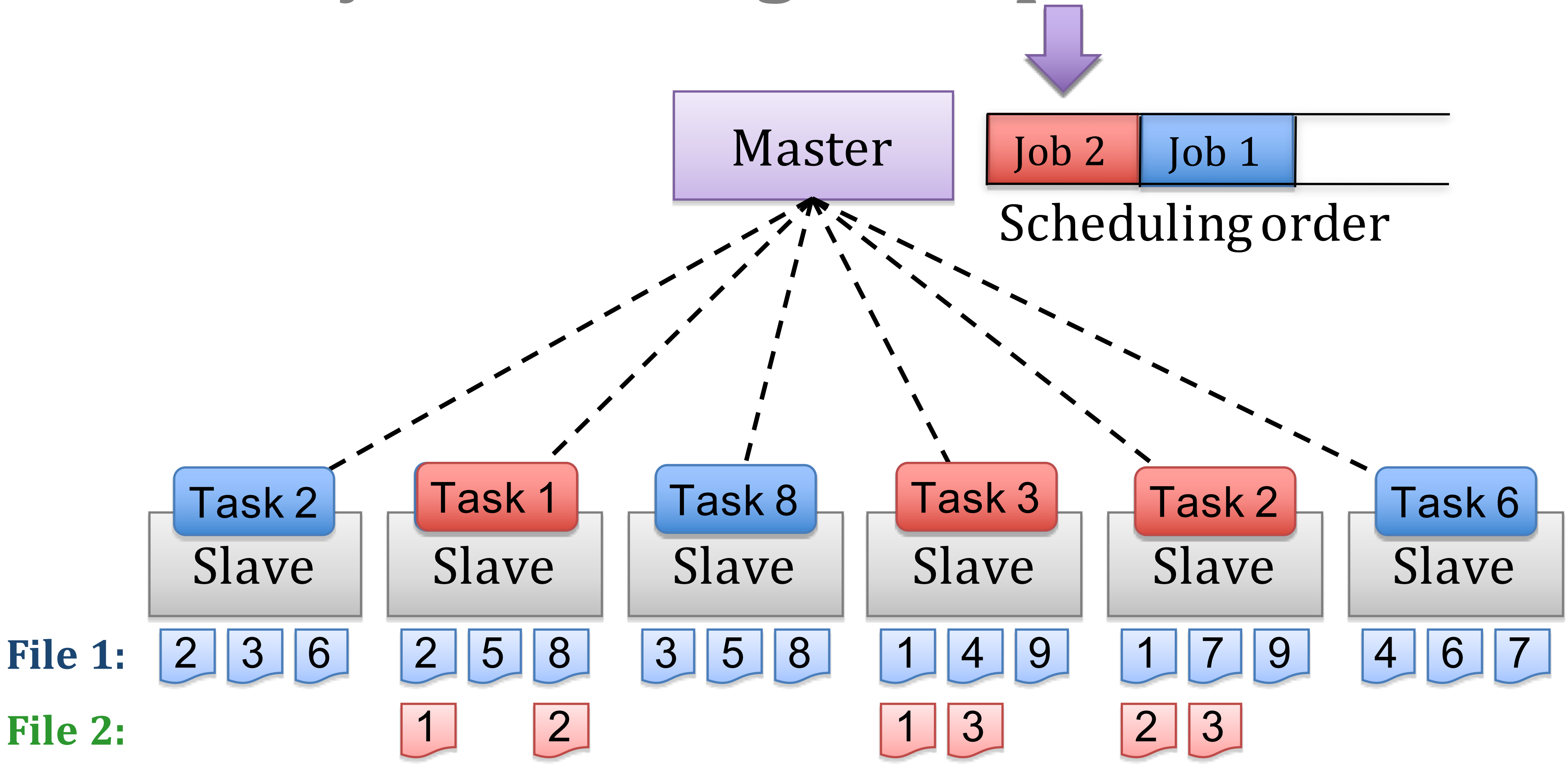
Delay Scheduling Example



Delay Scheduling Example



Delay Scheduling Example



Idea: Wait a short time to get data-local scheduling opportunities

Evaluation

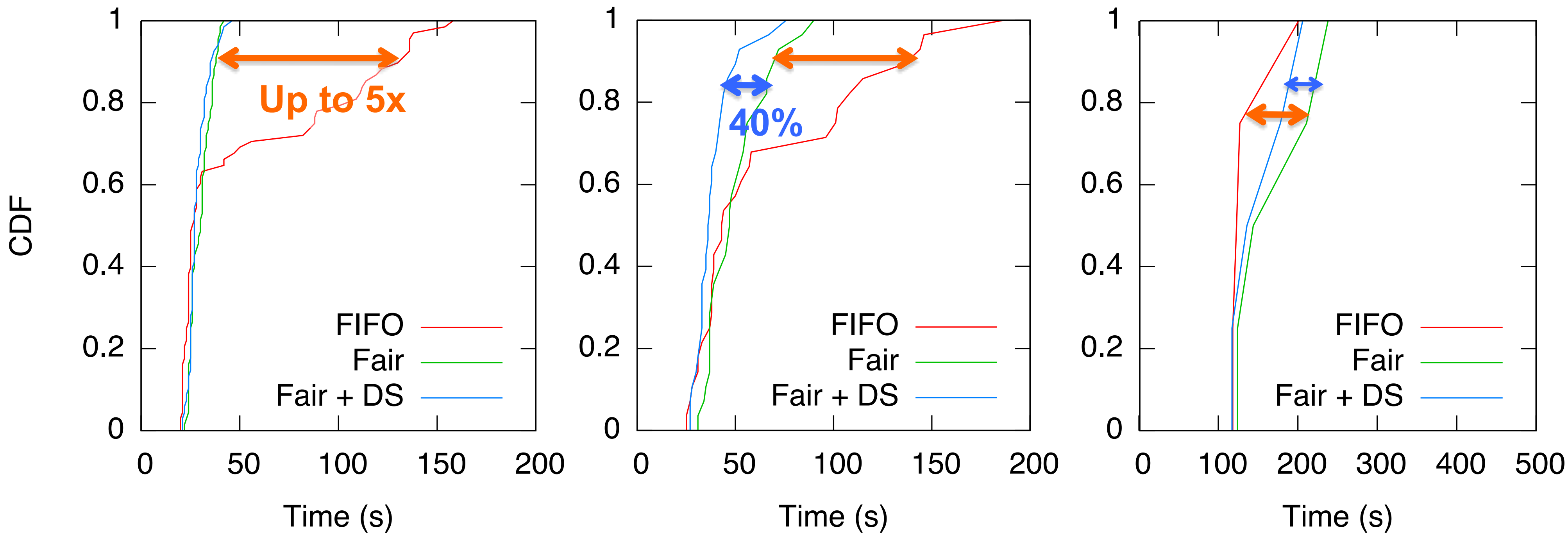
- Macrobenchmark
  - IO-heavy workload
  - CPU-heavy workload
  - Mixed workload
- Microbenchmarks
  - Sticky slots
  - Small jobs
  - Hierarchical fair scheduling
- Sensitivity analysis
- Scheduler overhead

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

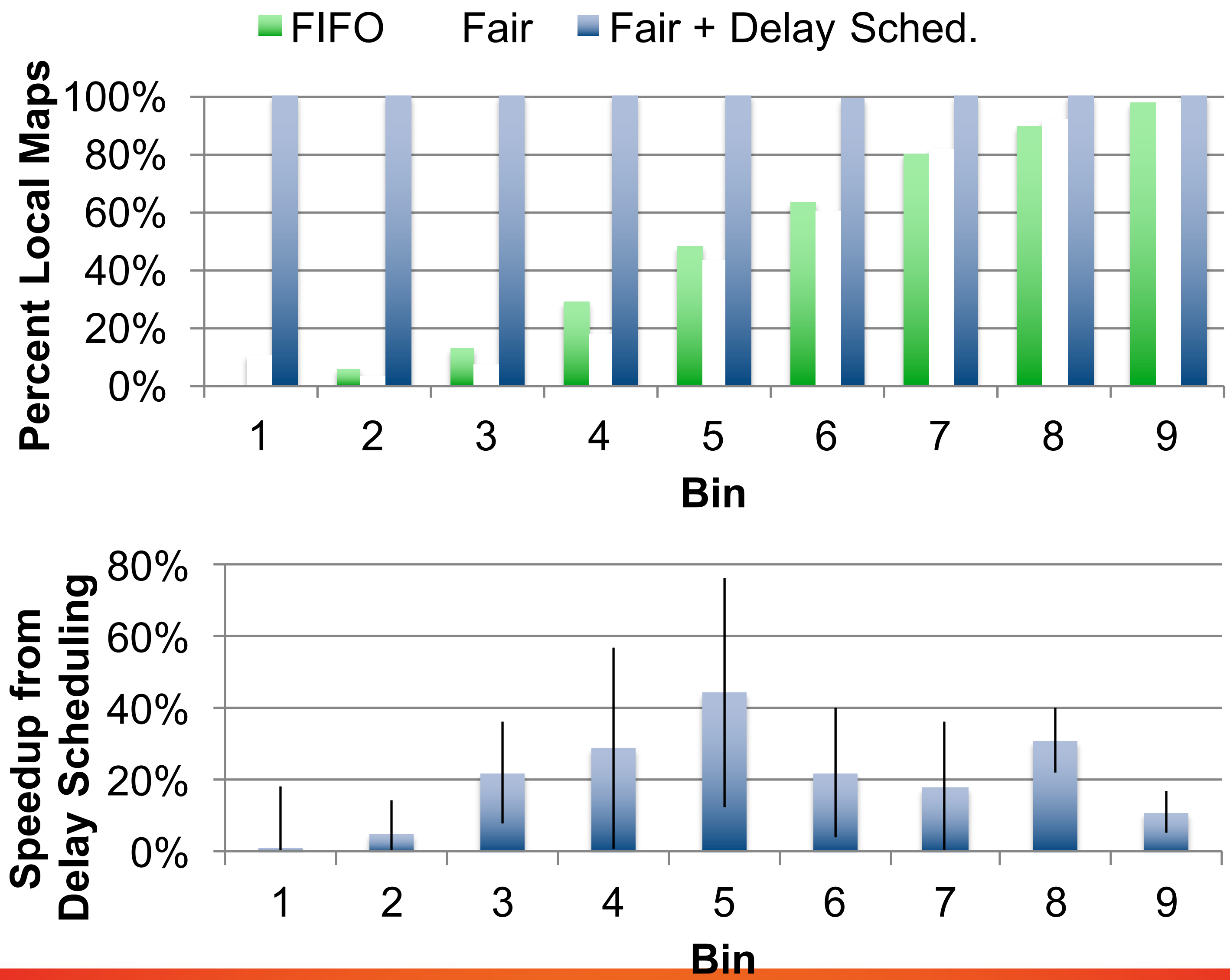


Small Jobs
(1-10 input blocks)

Medium Jobs
(50-800 input blocks)

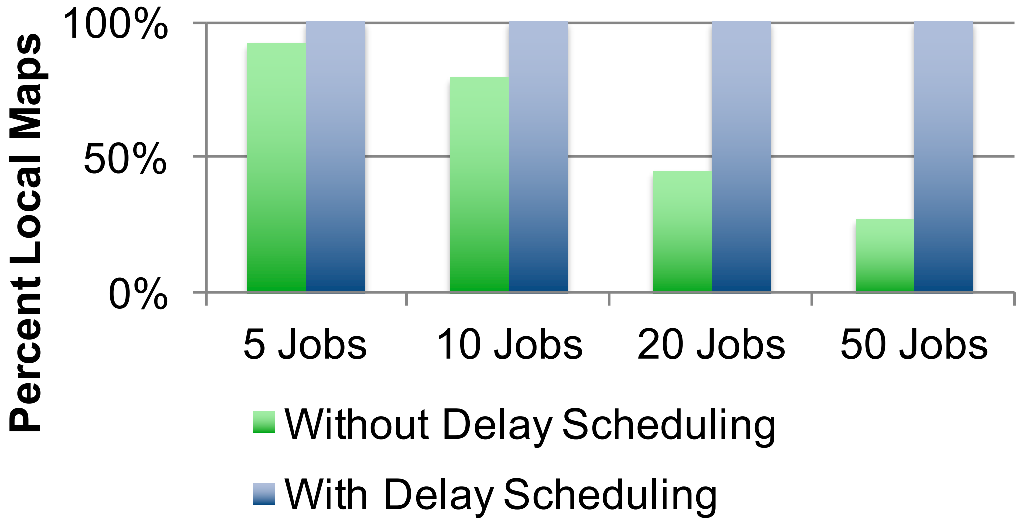
Large Jobs
(4800 input blocks)

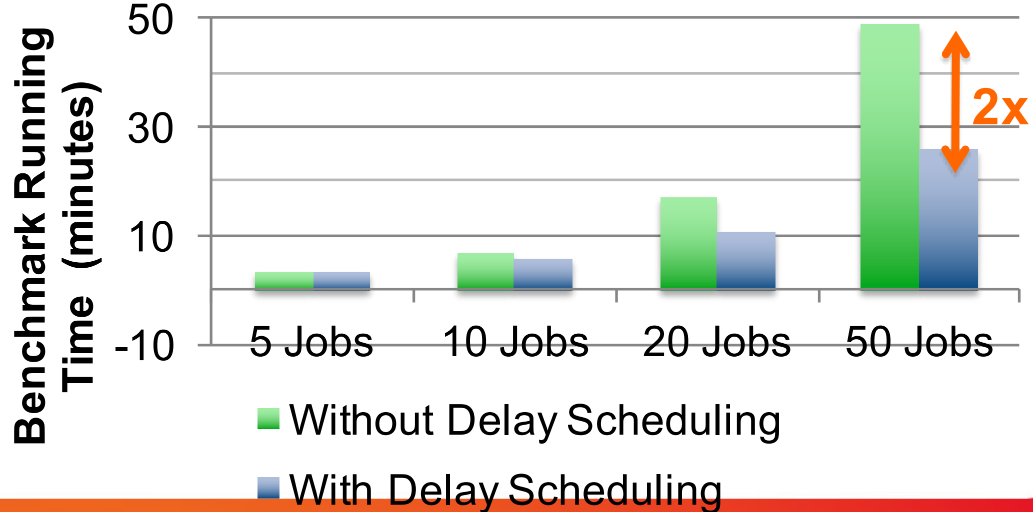
Results for IO-Heavy Workload



Sticky Slots Microbenchmark

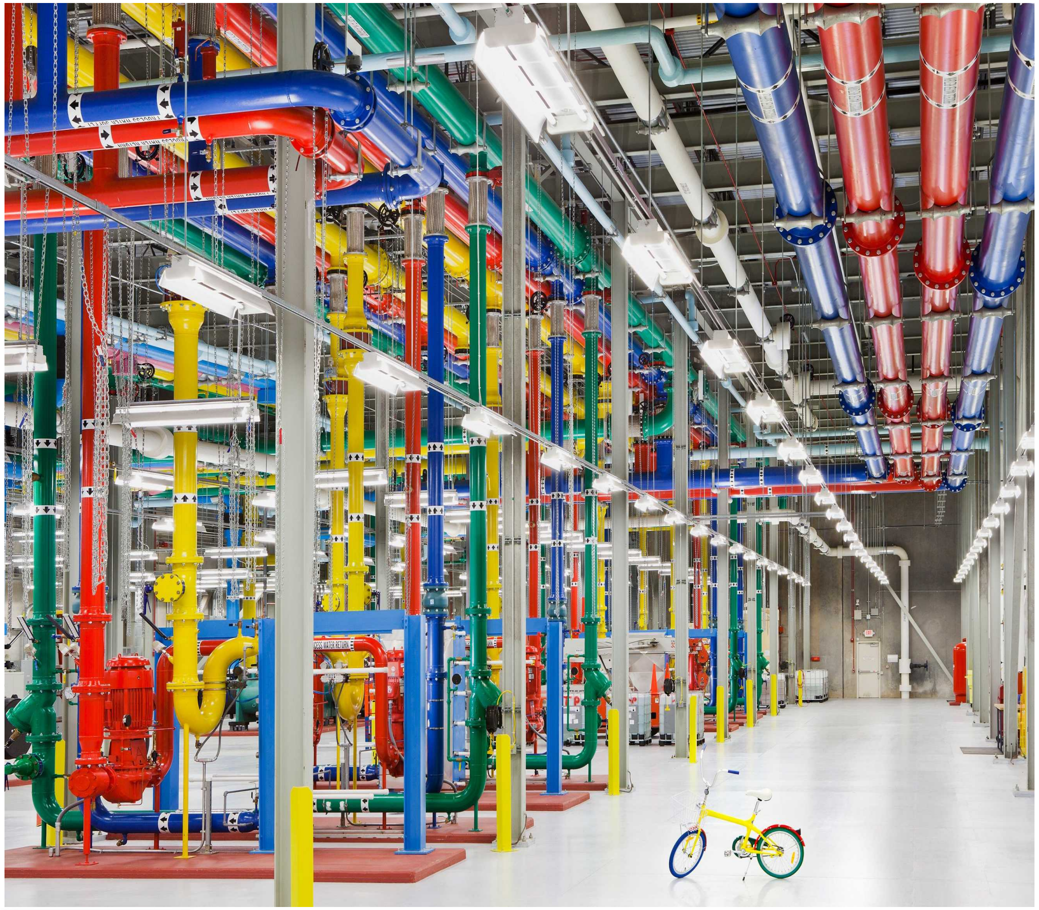
- 5-50 jobs on EC2
- 100-node cluster
- 4 cores / node
- 5s delay scheduling

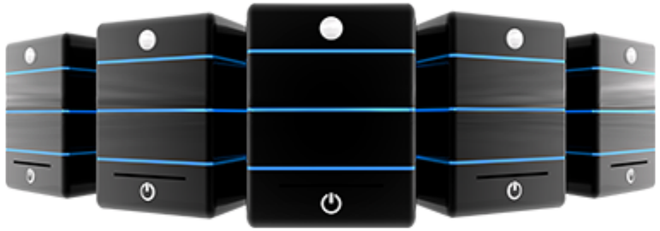




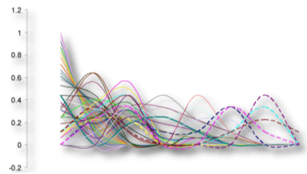
Job scheduling under Failures

Hadoop at large-scale clouds





Failures



Performance Variability

Towards Failure-aware scheduling

O. Yildiz, S. Ibrahim, G. Antoniu . *Enabling fast failure recovery in shared Hadoop clusters: Towards failure-aware scheduling.* *FGCS 2016*

In large-scale cloud, node failures are inevitable

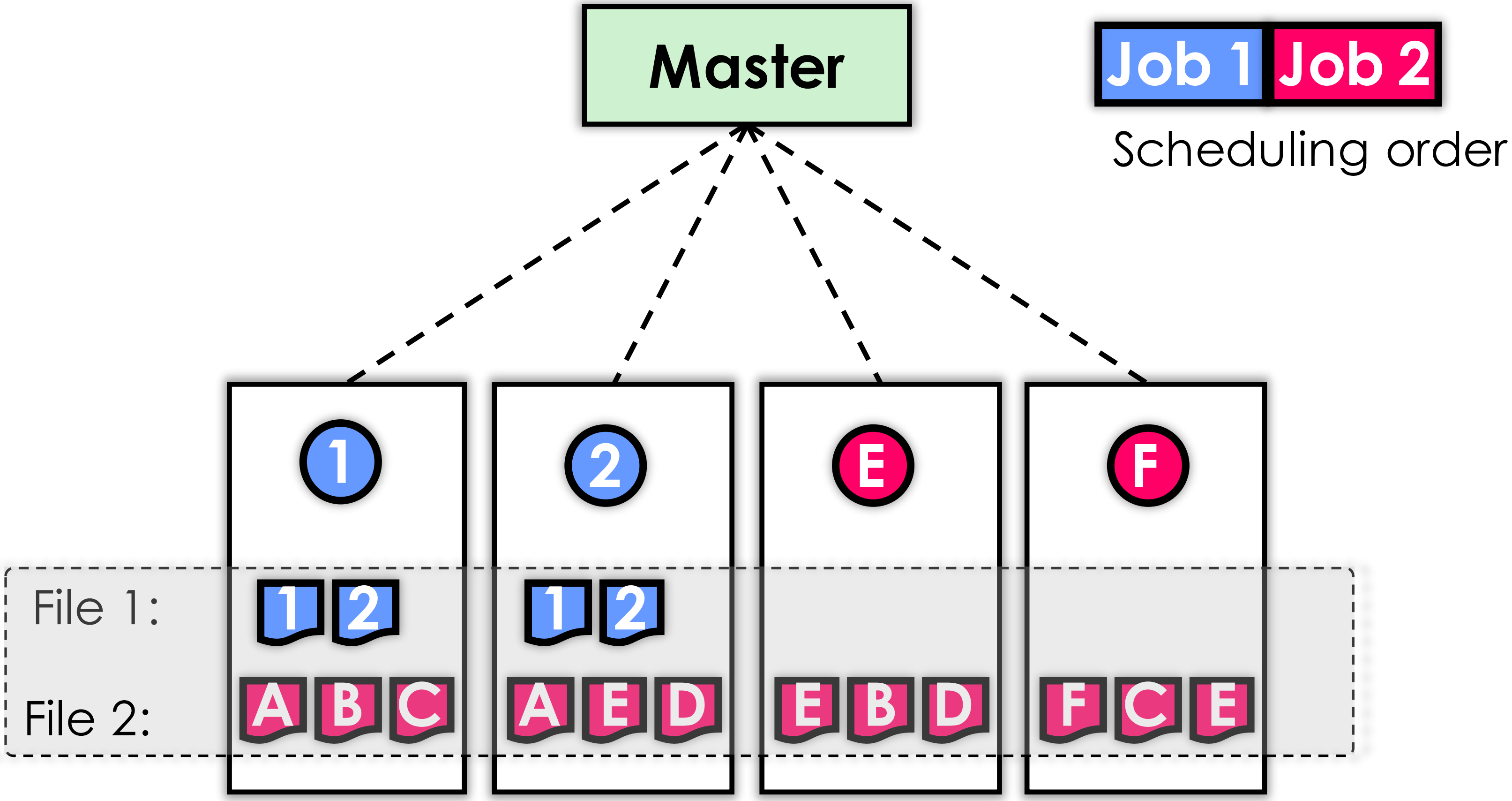
- 1000 machine failures in the 1st year of Google cluster[1]
- 10% -15% job failure rate in a CMU cluster

Failure recovery in Hadoop

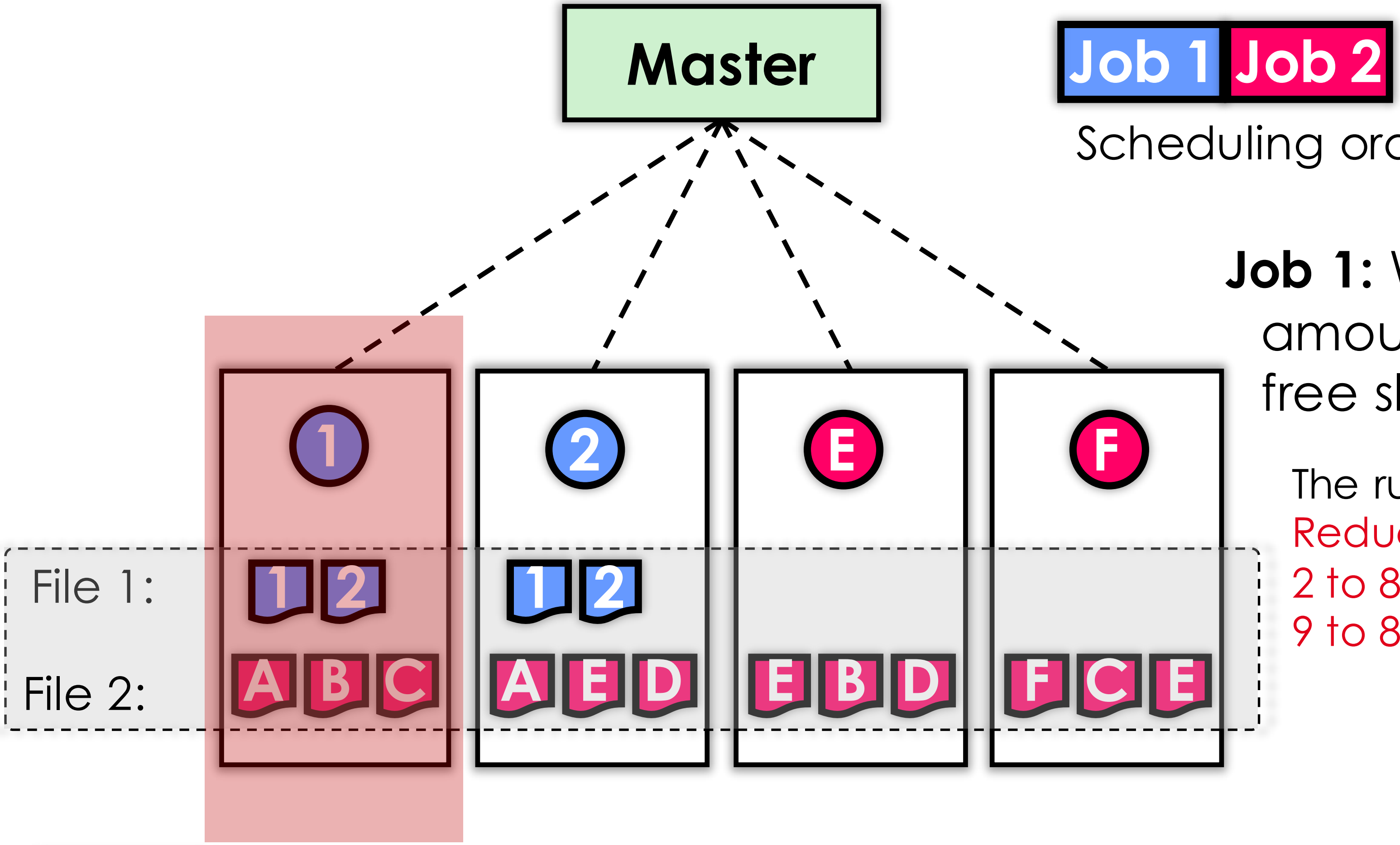
- Hadoop re-executes the tasks of failed machines

1 J. Dean, "Large-scale distributed systems at Google: Current systems and future directions" in keynote speech at the 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, 2009

In Shared Hadoop Cluster



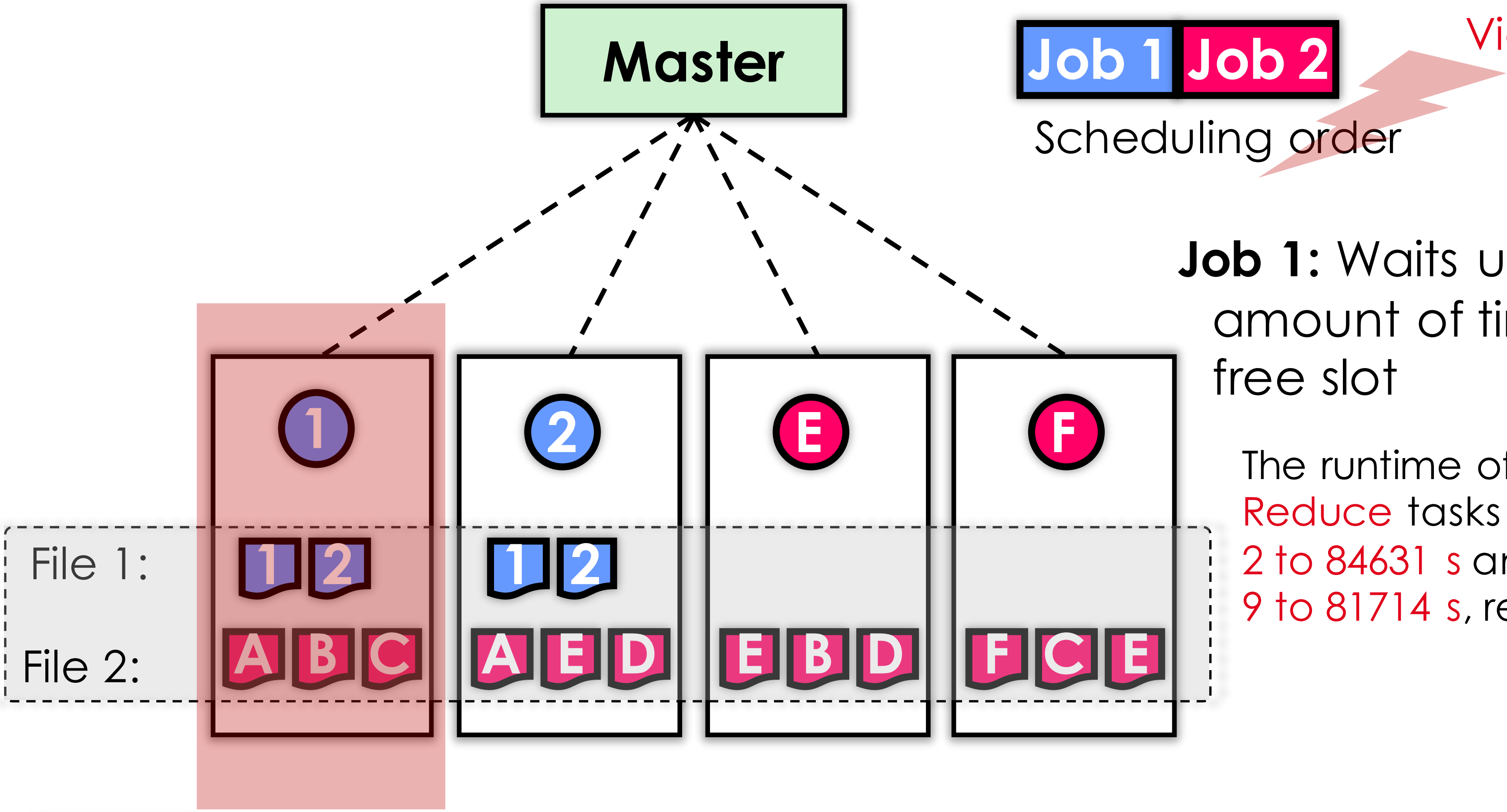
In Shared Hadoop Cluster



Job 1: Waits uncertain amount of time for a free slot

The runtime of Map and Reduce tasks varies from 2 to 84631 s and from 9 to 81714 s, respectively

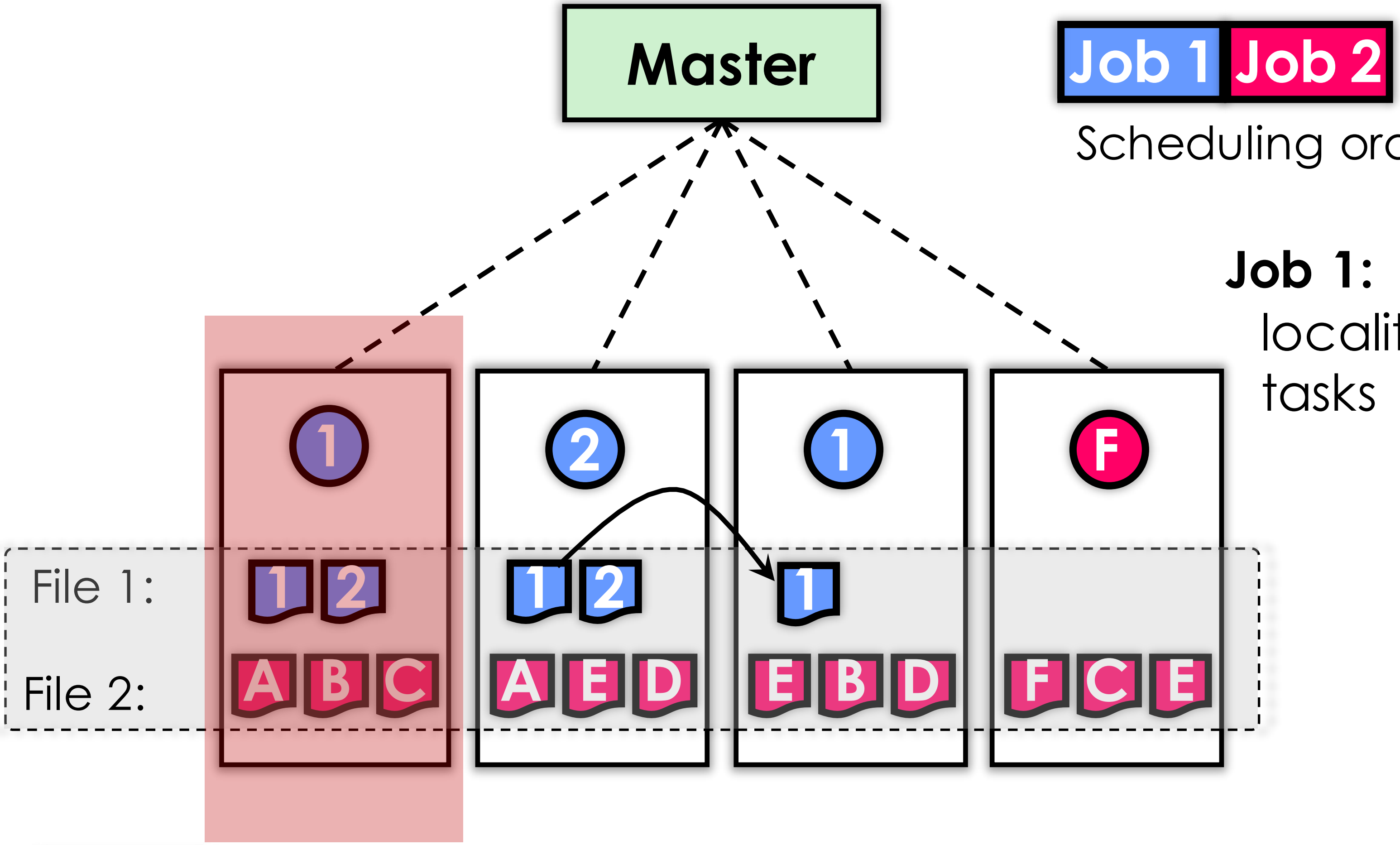
In Shared Hadoop Cluster



Job 1: Waits uncertain amount of time for a free slot

The runtime of Map and Reduce tasks varies from 2 to 84631 s and from 9 to 81714 s, respectively

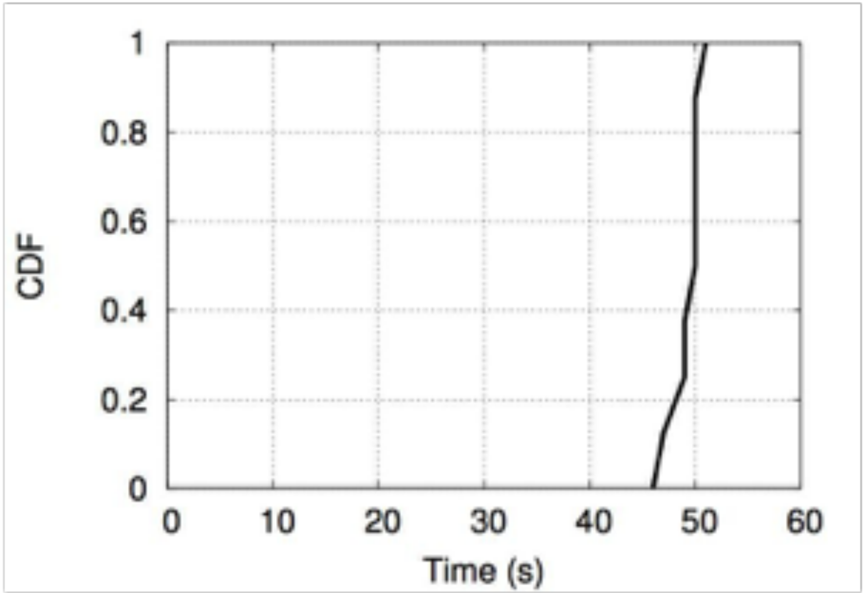
In Shared Hadoop Cluster



Job 1: Ignores the data locality of the recovery tasks

Hadoop Under Failures: Experimental Analysis

- Increase in job execution times by 30% to 70% due failures



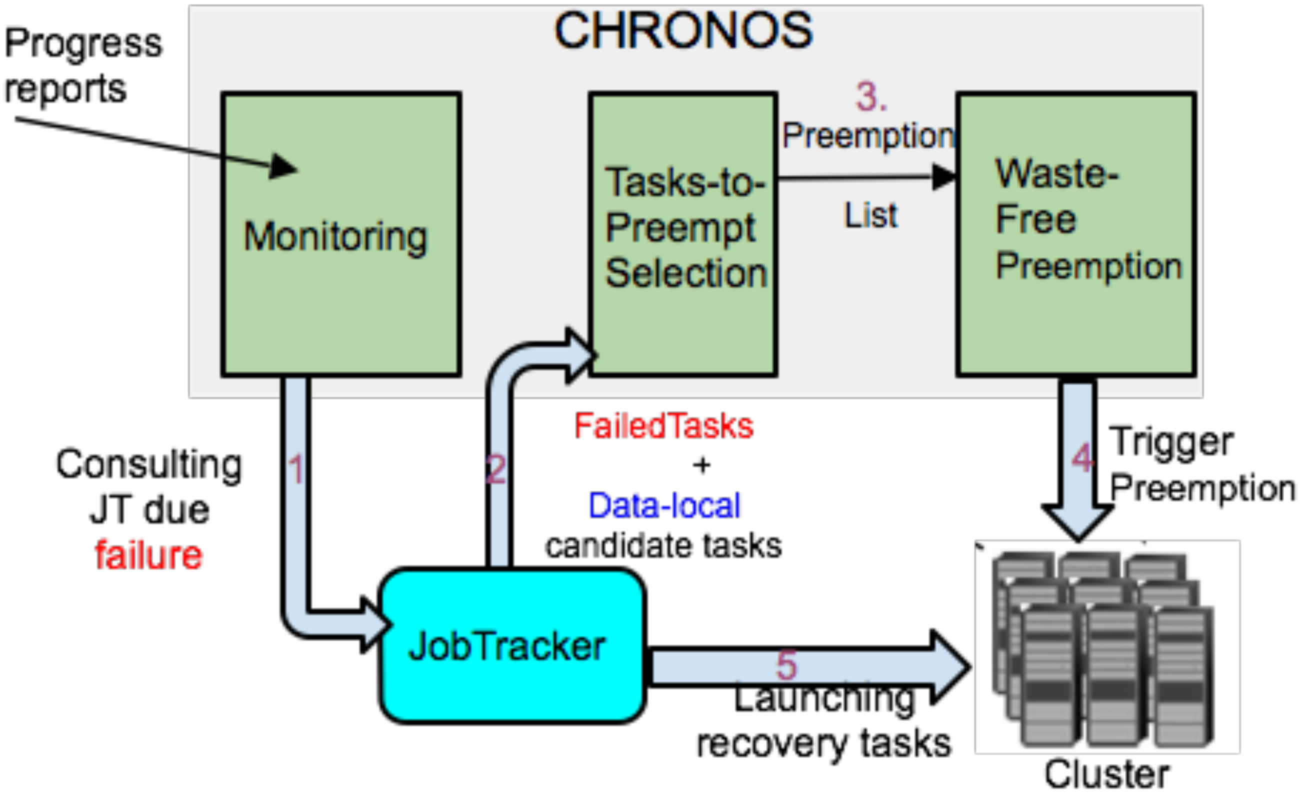
- Long waiting time for the recovery tasks: up to 51 seconds

Chronos

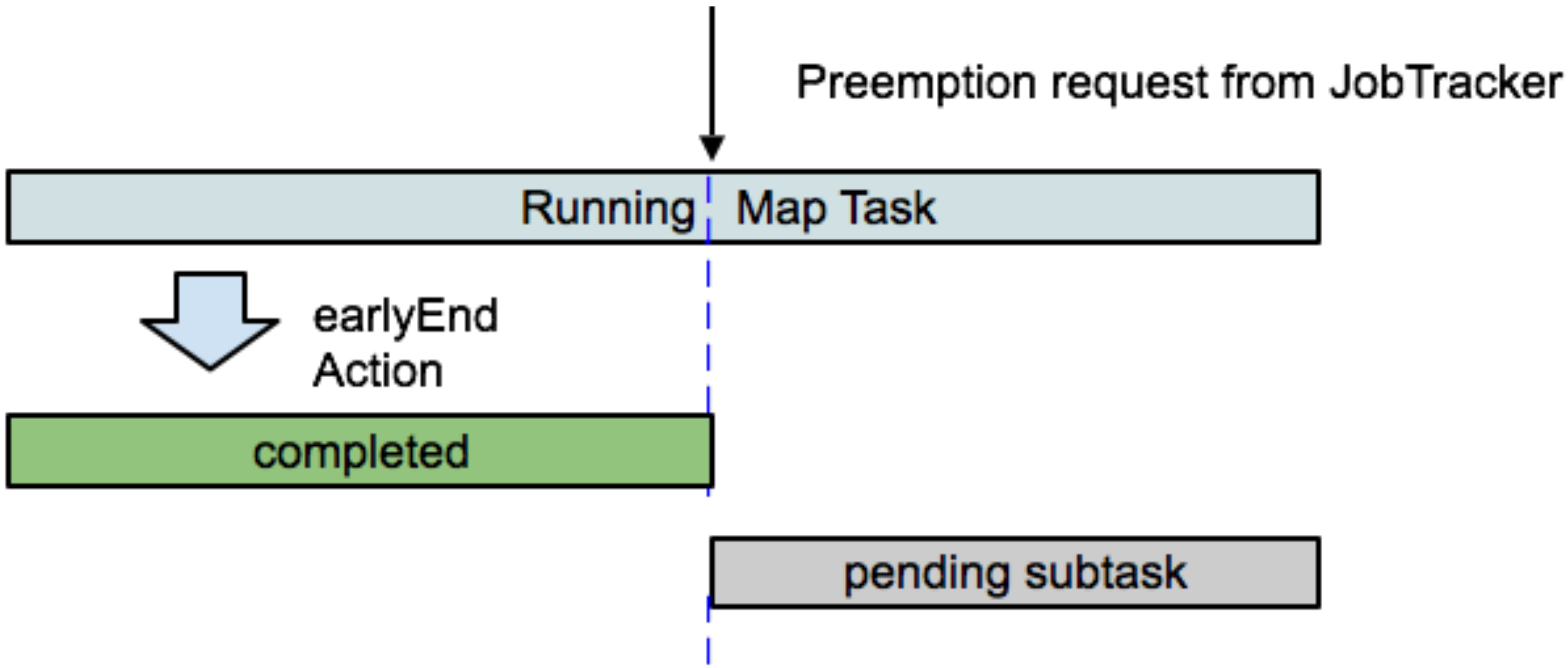
Chronos is a failure-aware scheduling strategy:

- Takes early action upon failure
  - Employs light-weight preemption technique
- Embraces a smart selection algorithm
  - Considers three criteria: the progress scores of running tasks, the scheduling objectives, and the recovery tasks input data locations.

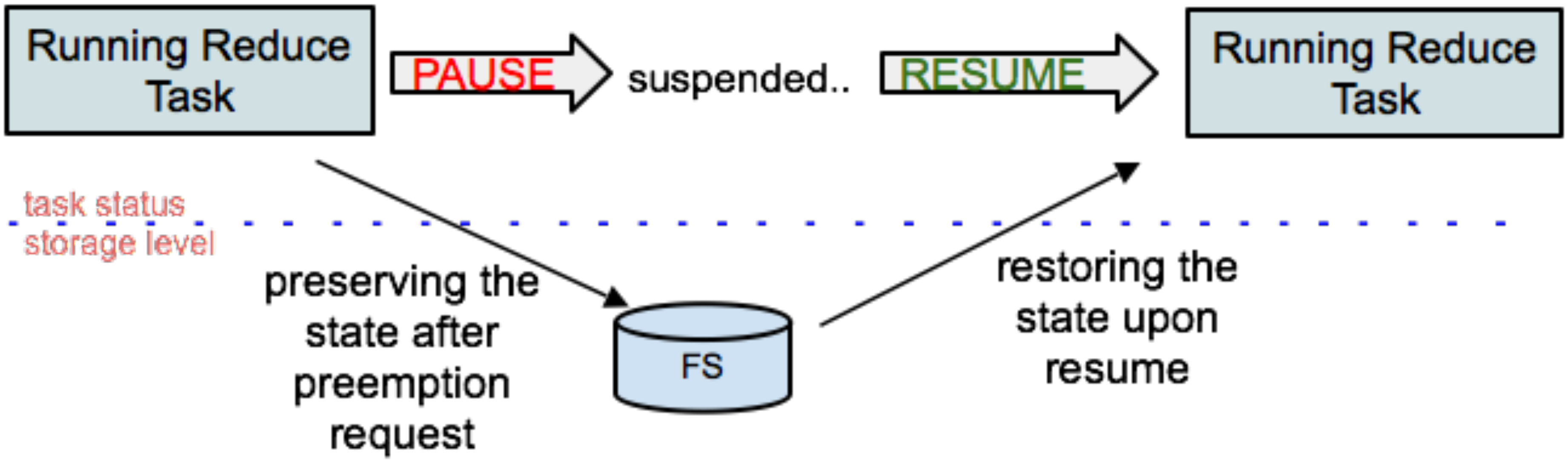
Chronos: Overview



Chronos: Overview

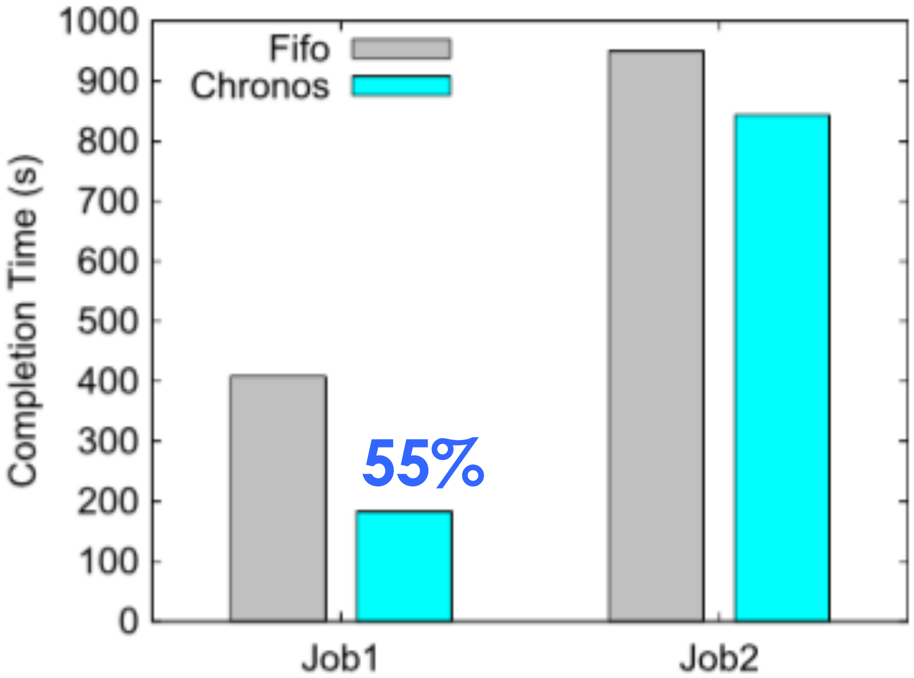


Chronos: Overview

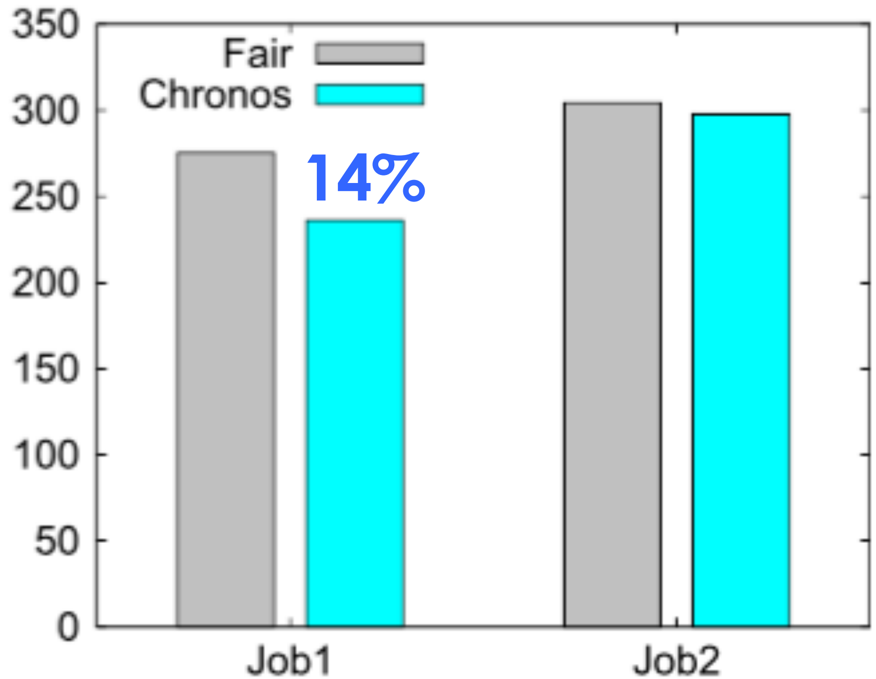


Evaluation

Chronos is independent of scheduling policy

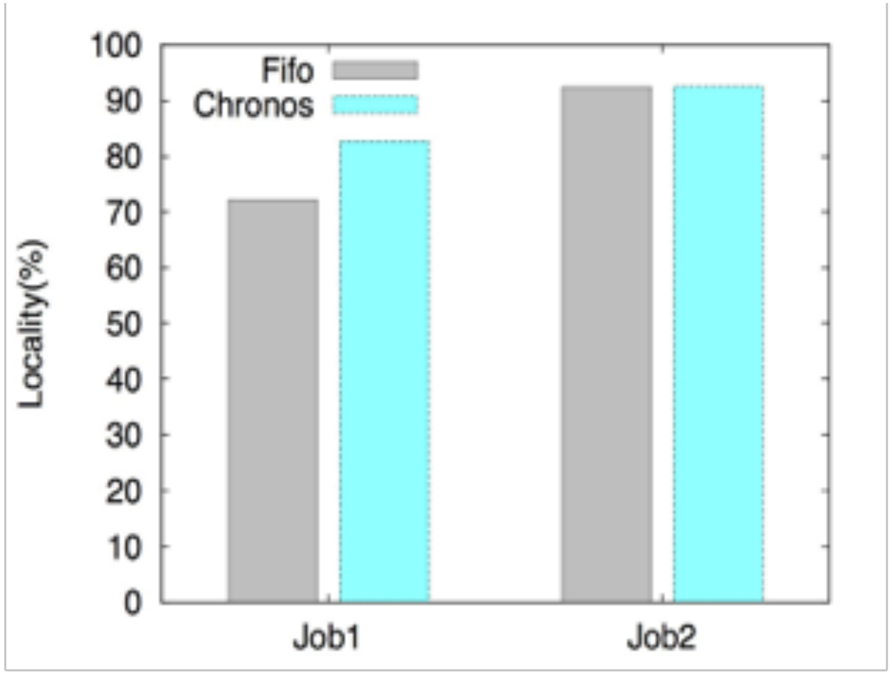


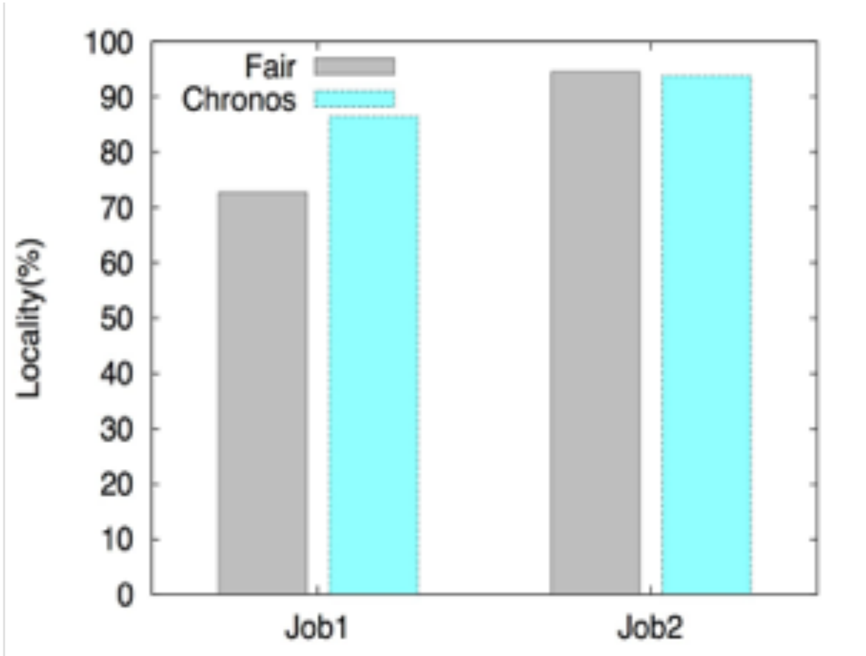
Sort: Chronos reduces the waiting time from 325 to 2 s



WordCount: Chronos reduces the waiting time from 48 s (16% of the total execution time) to 1.5 s

Evaluation





Chronos improves the data-locality by executing the recovery tasks locally

Job scheduling: Waiting Time

On the Usability of Shortest Remaining Time First Policy in Shared Hadoop Cluster, In the 31st ACM Symposium On Applied Computing **ACM SAC 2016**.

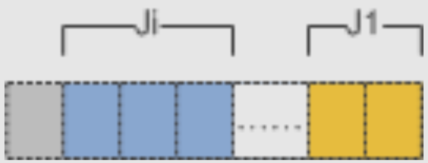
Motivation?

A practical problem is how to reduce job makespans (waiting time + execution time), especially for latency-sensitive small jobs

– 75% of the jobs in Facebook clusters are small jobs

Built-in Hadoop Schedulers

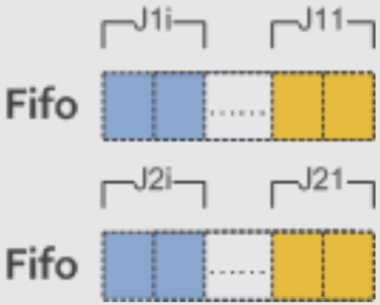
Fifo



Fair Scheduler

- Jobs are grouped into "pools"
- Resources are allocated across pools using weighted fair sharing
- **Delay technique to expose data locality**

Capacity Scheduler



- Jobs are grouped into queues
- Resources are partitioned among queues

Focus on improving job execution times by optimizing data locality

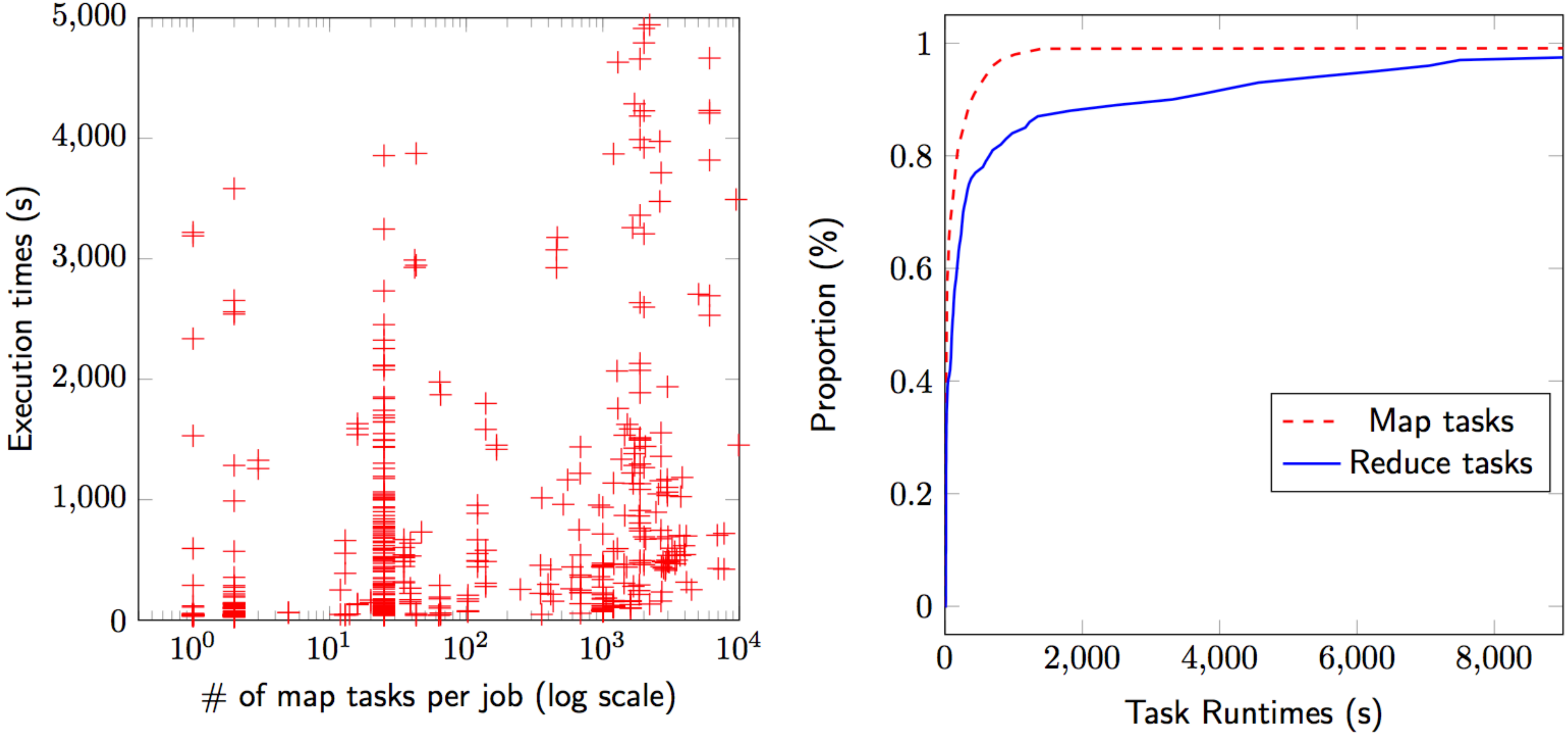
Problem definition

A few efforts have focused on reducing job waiting times, although waiting time is as important as execution time to improve job makespans.

Evaluating and prioritizing jobs according to their input data sizes may result in long waiting times

– Some jobs may have smaller input sizes but higher execution complexity.

CMU Hadoop research clusters

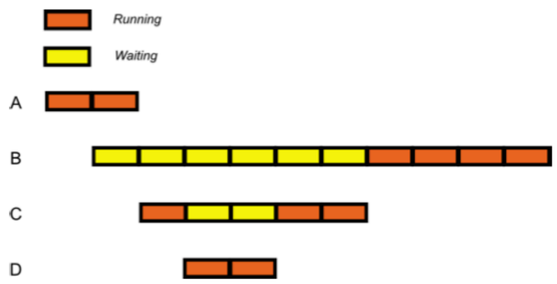


Why hSRTF?

An adaption of the Shortest Remaining Time First scheduler in shared Hadoop clusters.

- Prioritize short jobs
  - With critical response times
- Conceived to reduce waiting time
- Challenges:
  - Remaining time estimation
  - Multi-mode adoption
  - Fast allocation of resources

Job	Size
A	2
B	4
C	3
D	2



hSRTF in Hadoop

- Estimates remaining time of running jobs
  – Make full use of map slots and reduce slots

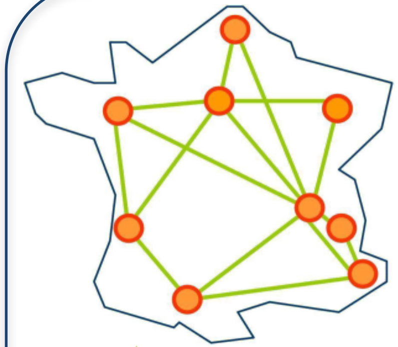
$$remaining\_time = \left\lceil \frac{map\_unfinished}{map\_capacity} \right\rceil * avg\_map\_time$$
$$+ \left\lceil \frac{reduce\_unfinished}{reduce\_capacity} \right\rceil * avg\_reduce\_time$$

  – Up-to-date time estimation
    - The remaining time is recomputed every 10 sec to cope with the dynamicity of currently running jobs and infrastructure.

hSRTF in Hadoop

- Provides fast allocation of resources to the job with shortest remaining time
  - Equipped with wait and kill primitives
- Multi mode
  - Pure hSRTF (hSRTF-Pu)
    - All the resources are allocated to the job with the shortest remaining time.
  - Time-based proportional sharing (hSRTF-Pr)
    - Allocates resources to jobs according to their remaining times.

Methodology: testbed and platform



Grid'5000

58 nodes on Toulouse Site

Each node has:

- Intel 4-core CPU
- 8GB memory
- Gigabit connection

Hadoop 1.0.4

4 Map slots, 2 Reduce slots per node
Replication factor: 3
Block size: 128MB

Methodology: benchmarks

We run a mixed workload consisting of sort and wordcount applications.

– Total of 31 jobs
– Each job is submitted 10 seconds after each other.

	Application	# of maps	# of reduces	# of jobs
Large jobs	Sort	256	32	3
Medium Jobs	Sort	64	8	8
Small Jobs	Sort	1	1	10
	WordCount	1	1	10

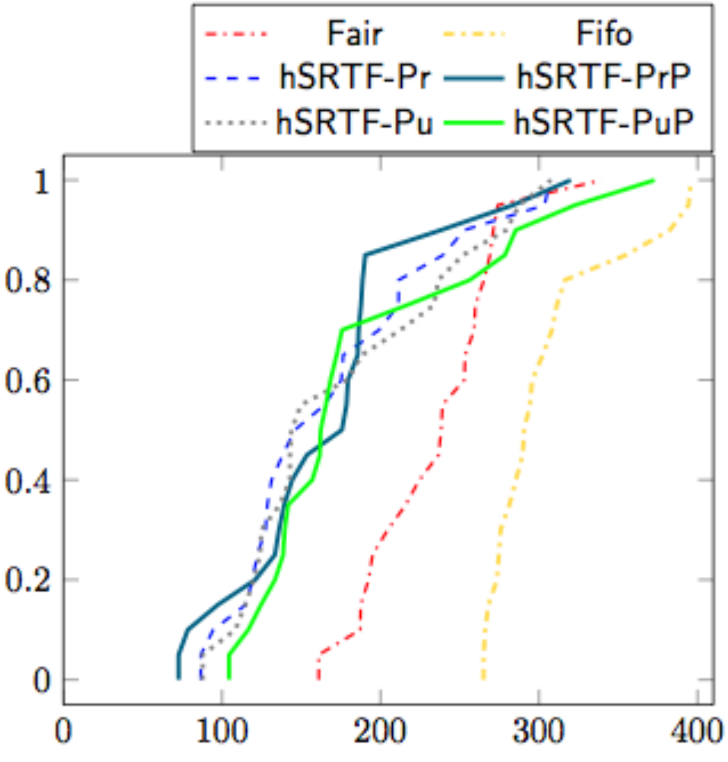
Methodology: List of schedulers

Scheduler	Description
Fifo	Priority scheduler with respect to job submission time
Fair	Provides fair allocation of resources between different jobs
hSRTF-Pu	Allocates all resources to the job with the shortest remaining time
hSRTF-PuP	Similar to *hSRTF-Pu*, but with the possibility of preempting (kill) running tasks which belong to a job with the longest remaining time to provide early allocation to a job with the shortest remaining time
hSRTF-Pr	Allocates resources to jobs according to their remaining time
hSRTF-PrP	Similar to *hSRTF-Pr*, but with the possibility of preemption

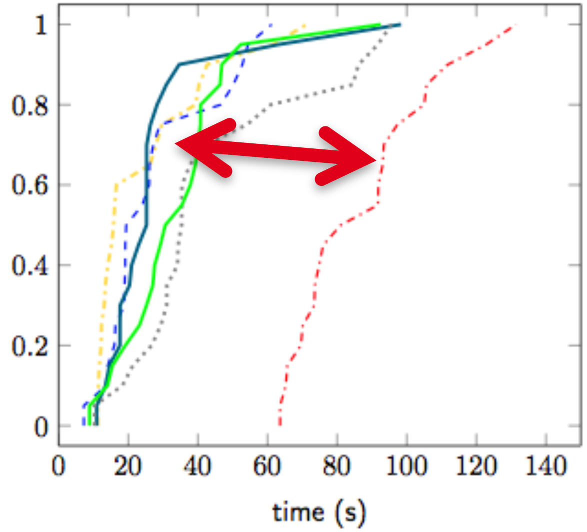
Small Jobs: Reducing the waiting time

hSRTF-Pr vs Fair

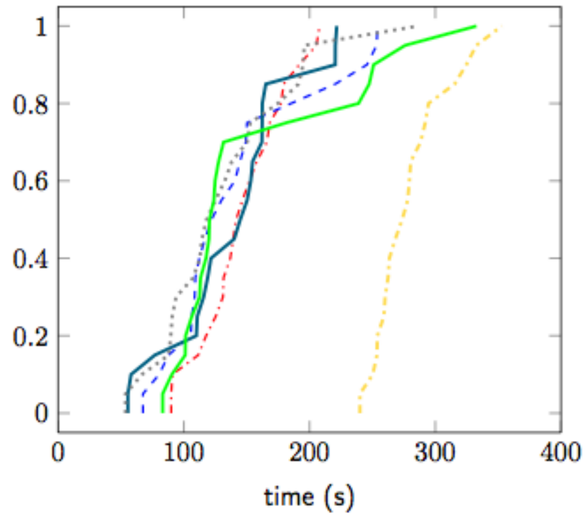
hSRTF-Pr reduces the makespans of small jobs with an average speedup of 26% compared to **Fair**.



(a) Makespan - small



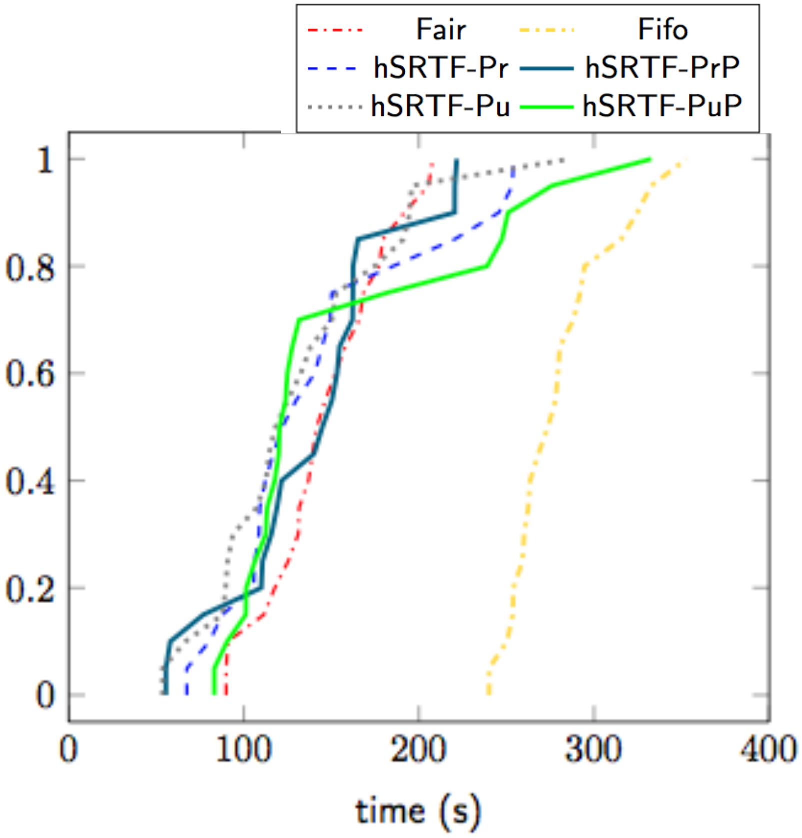
(d) Waiting time - small



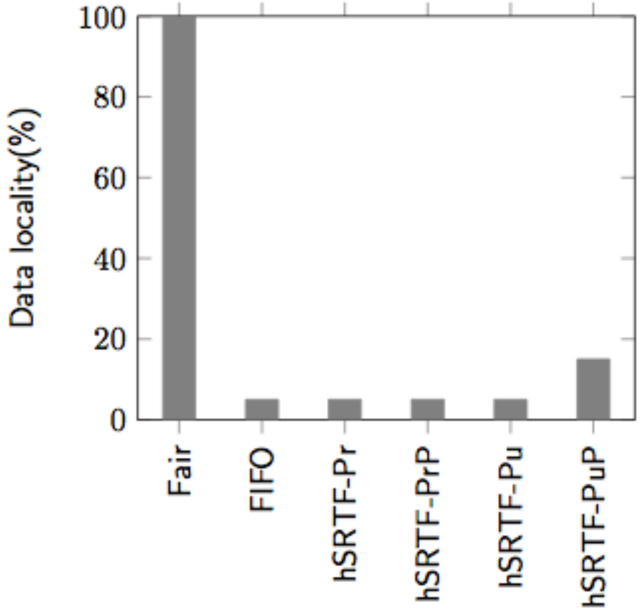
(g) Execution time - small

Small Jobs: Co-locating map and reduce tasks

hSRTF-Pr vs Fair



(g) Execution time - small



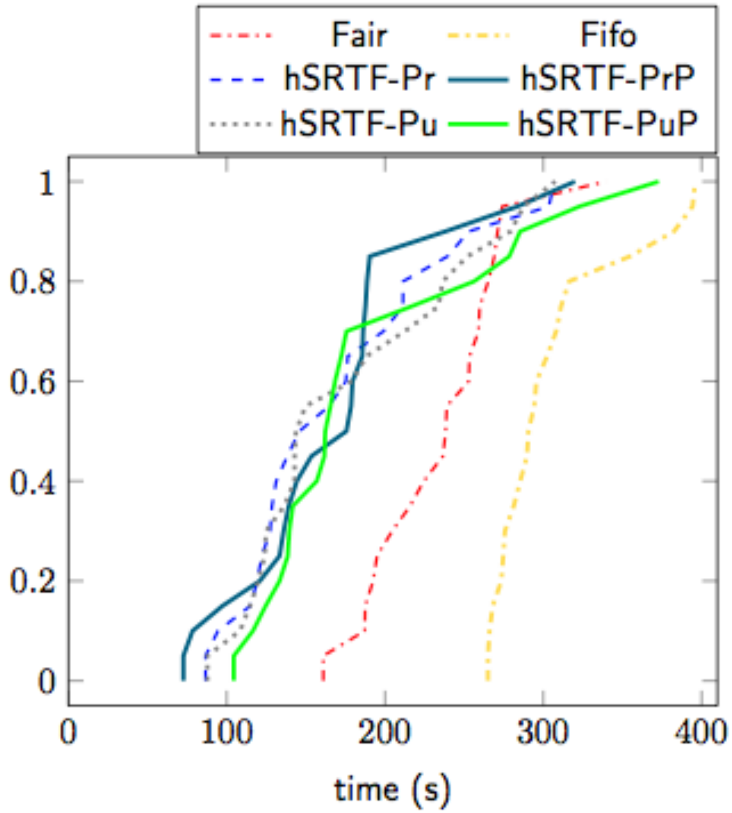
(a) Data locality - small

Fair achieves 100% data locality while hSRTF-Pr obtains only 5% data locality.

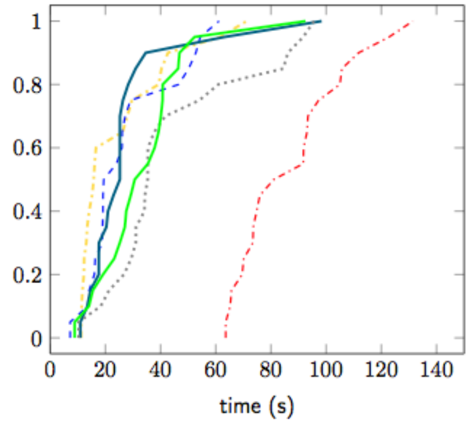
Due to the fine-grained co-location of map and reduce tasks which reduces data transfer during the shuffle phase

Small jobs: Avoid blocked jobs

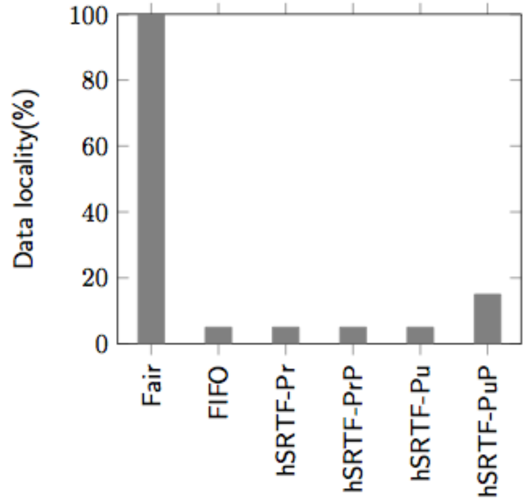
hSRTF-Pu vs Fifo



(a) Makespan - small



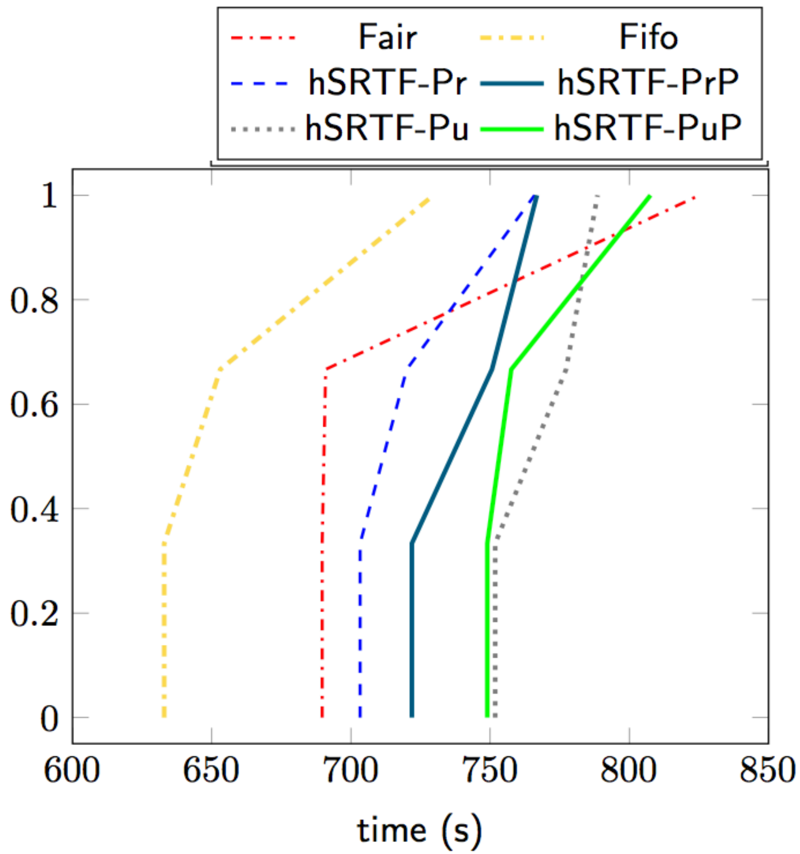
(d) Waiting time - small



(a) Data locality - small

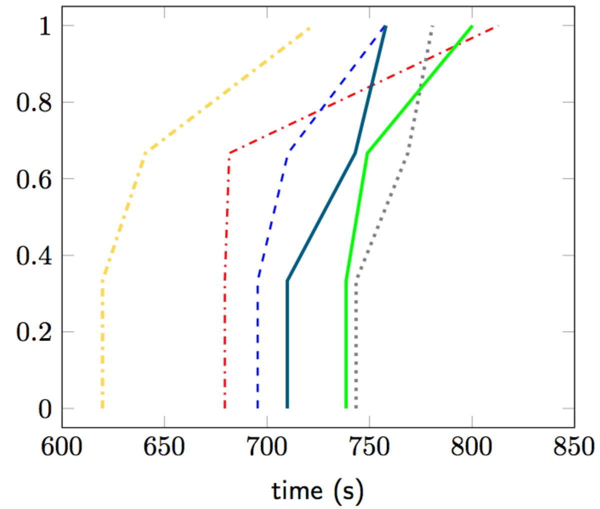
hSRTF-Pu reduces the makespans of small jobs with an average speedup of 43% compared to **Fifo**.

Large Jobs: Adversely impact



(c) Makespan - large

hSRTF introduces a performance degradation for large jobs by (on average) 10% and 0.2% compared to Fifo and Fair schedulers, respectively.



(i) Execution time - large

Preemption adversely impacts the performance of large jobs because theses jobs will los the work of killed tasks