Job Scheduling for MapReduce

Matei Zaharia, Dhruba Borthakur*,
Joydeep Sen Sarma*, Scott Shenker, Ion Stoica

RAD Lab, *Facebook Inc

Motivation

- Hadoop was designed for large batch jobs
  - FIFO queue + locality optimization
- At Facebook, we saw a different workload:
  - Many users want to share a cluster
  - Many jobs are small (10-100 tasks)
    - Sampling, ad-hoc queries, periodic reports, etc

➢ How should we schedule tasks in a shared MapReduce cluster?

Benefits of Sharing

- Higher utilization due to statistical multiplexing
- Data consolidation (ability to query disjoint data sets together)

Why is it Interesting?

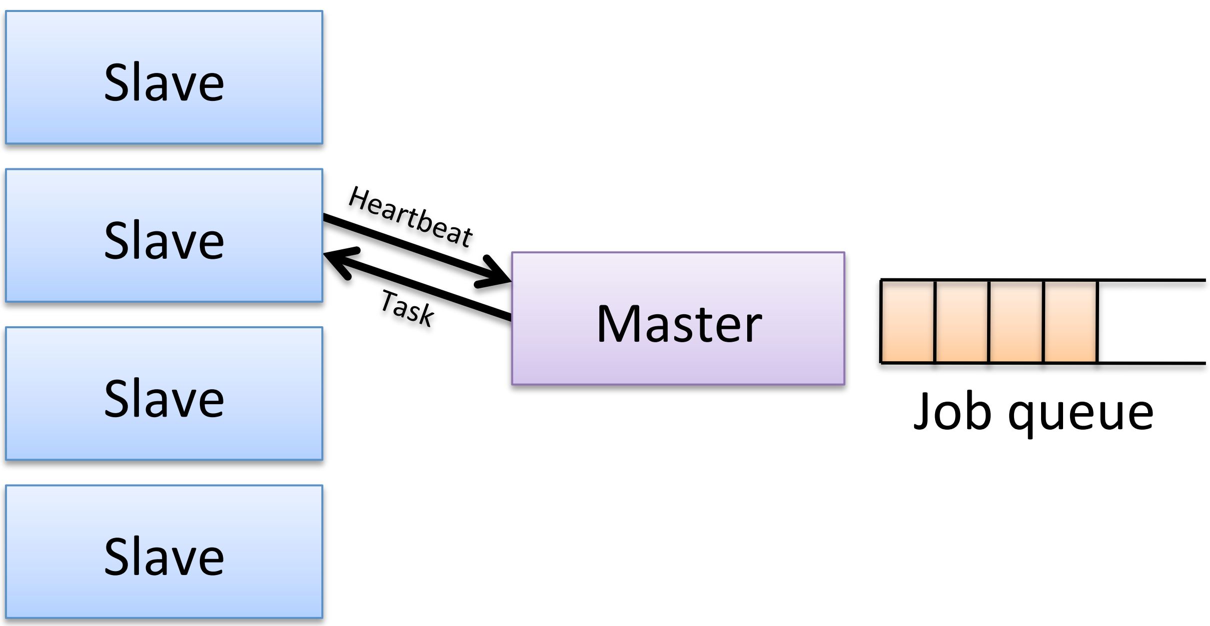
- Data locality is crucial for performance
- Conflict between locality and fairness
- 70% gain from simple algorithm

Outline

- Task scheduling in Hadoop
- Two problems
  - Head-of-line scheduling
  - Slot stickiness
- A solution (global scheduling)
- Lots more problems (future work)

Task Scheduling in Hadoop

- Slaves send heartbeats periodically
- Master responds with task if a slot is free, picking task with data closest to the node

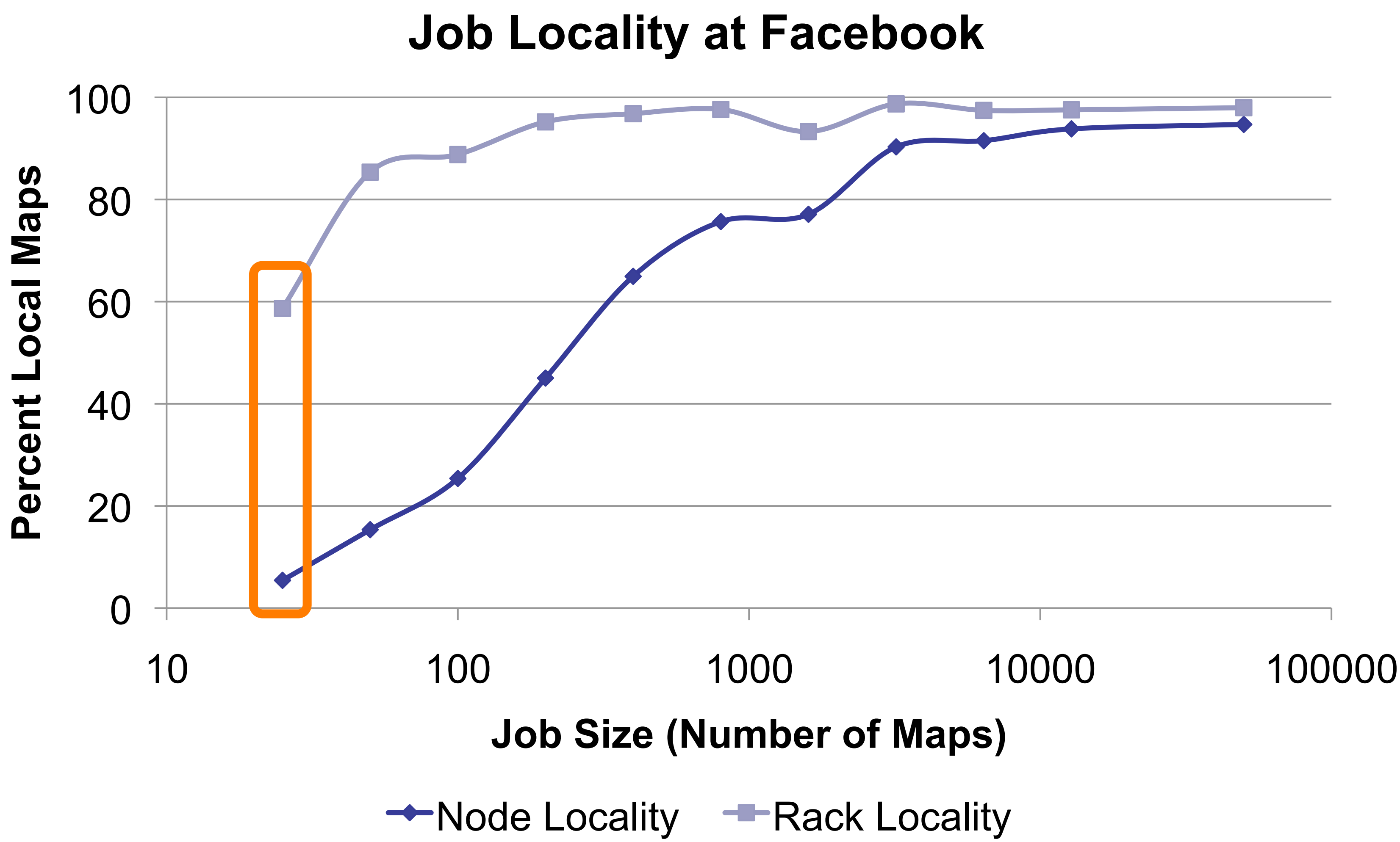


Problem 1: Poor Locality for Small Jobs

Job Sizes at Facebook

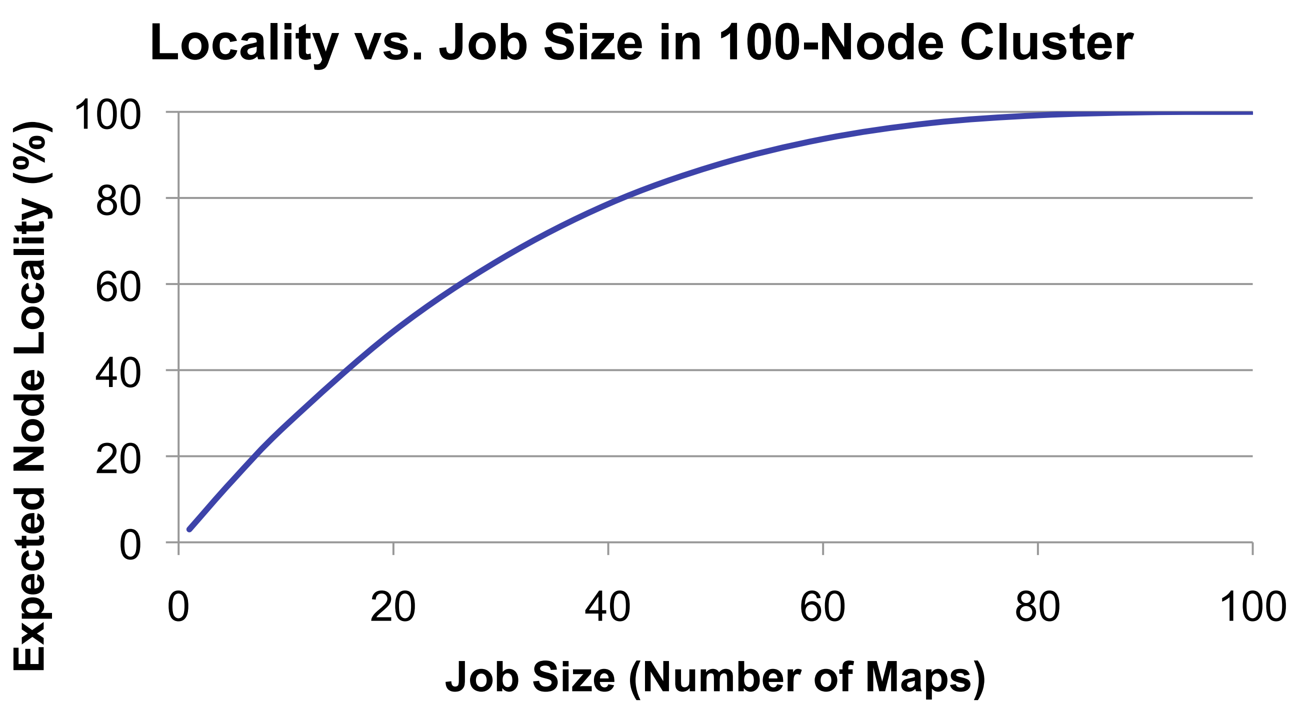
# of Maps	Percent of Jobs
< 25	58%
25-100	18%
100-400	14%
400-1600	7%
1600-6400	3%
> 6400	0.26%

Problem 1: Poor Locality for Small Jobs



Cause

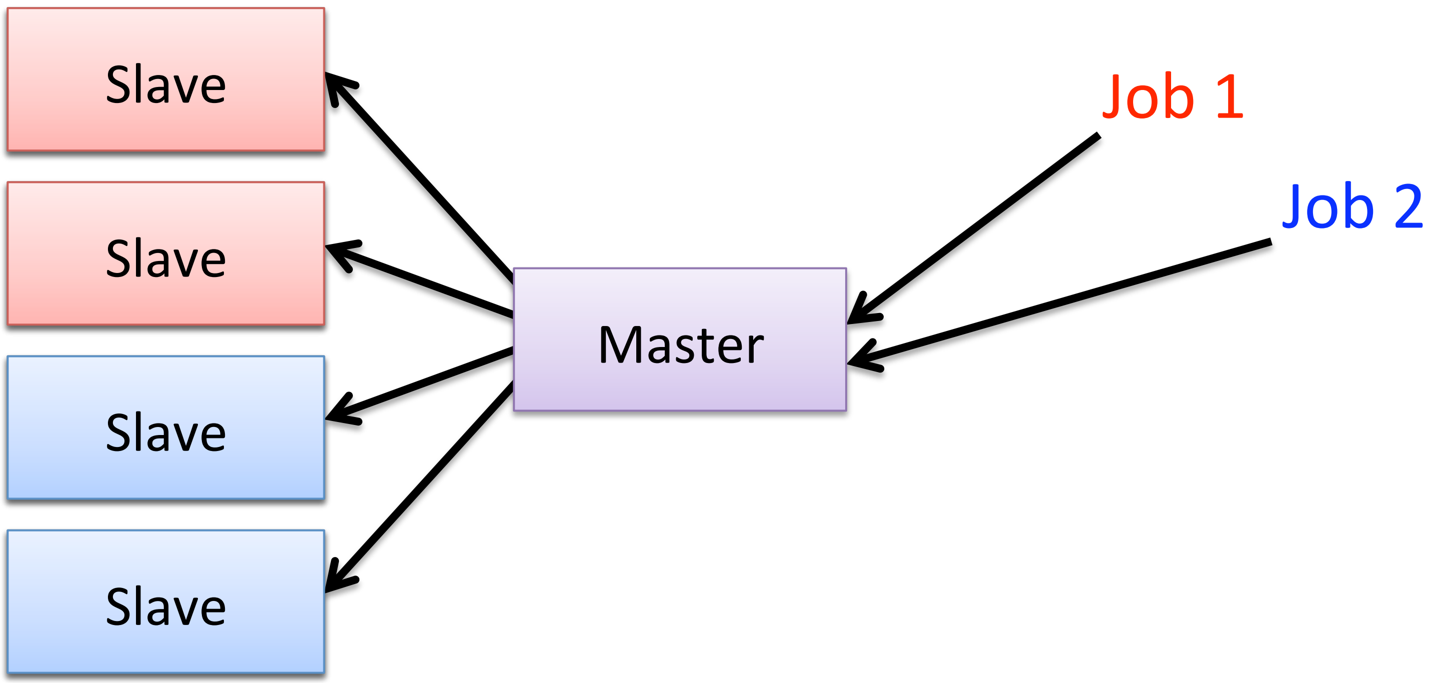
- Only head-of-queue job is schedulable on each heartbeat
- Chance of heartbeat node having local data is low
- Jobs with blocks on X% of nodes get X% locality



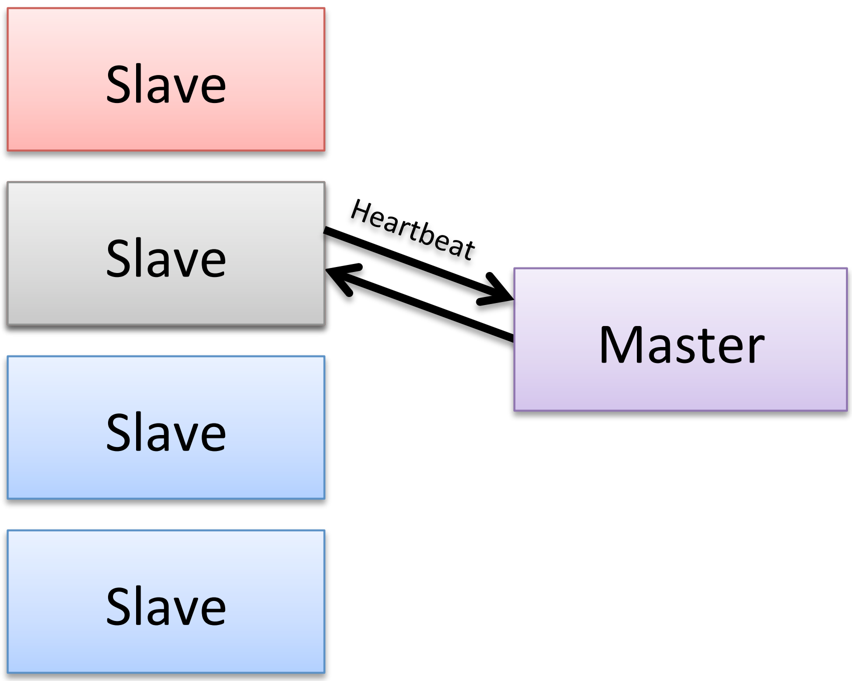
Problem 2: Sticky Slots

- Suppose we do fair sharing as follows:
  - Divide task slots equally between jobs
  - When a slot becomes free, give it to the job that is farthest below its fair share

Problem 2: Sticky Slots



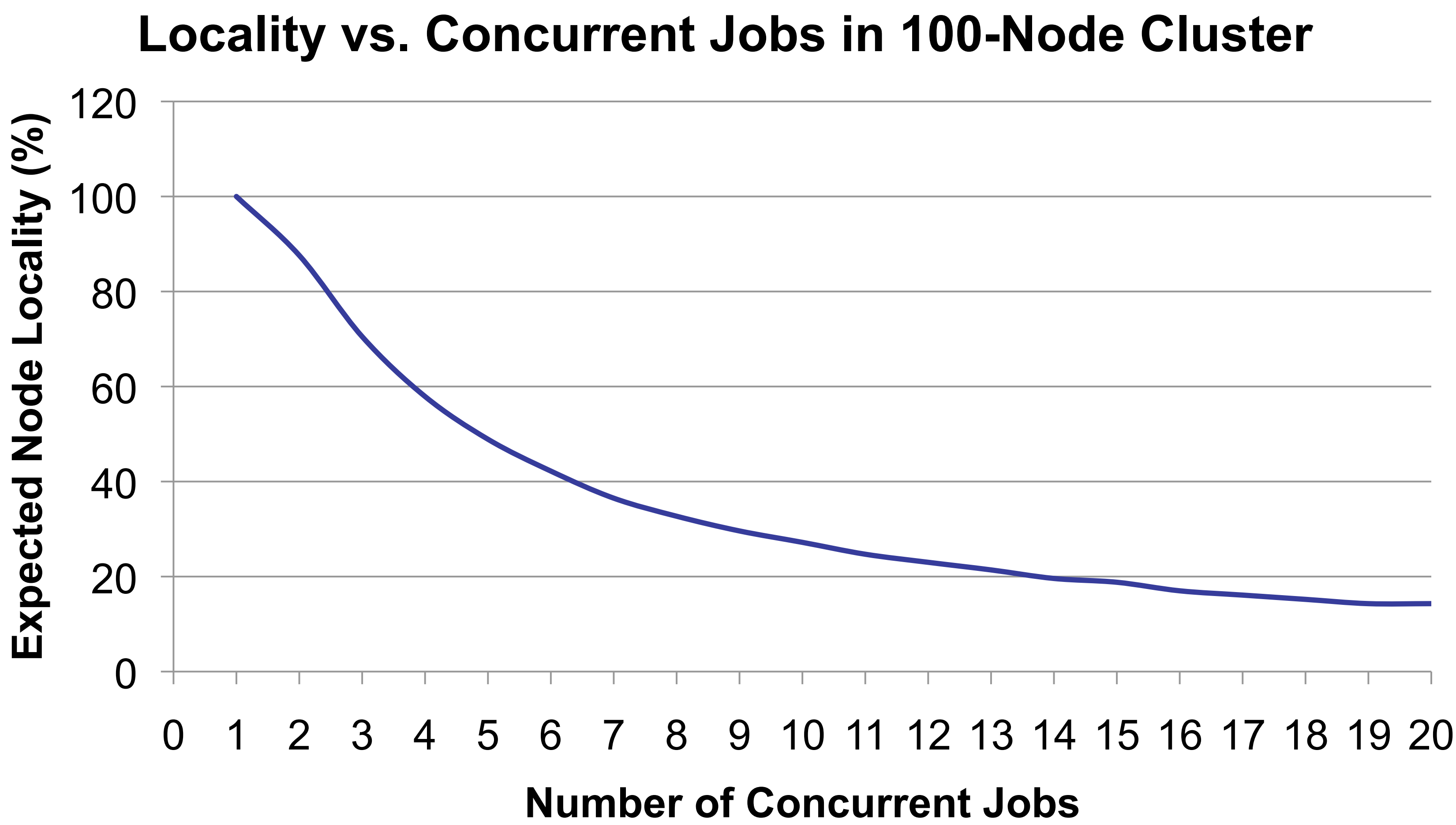
Problem 2: Sticky Slots



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

Problem: Jobs never leave their original slots

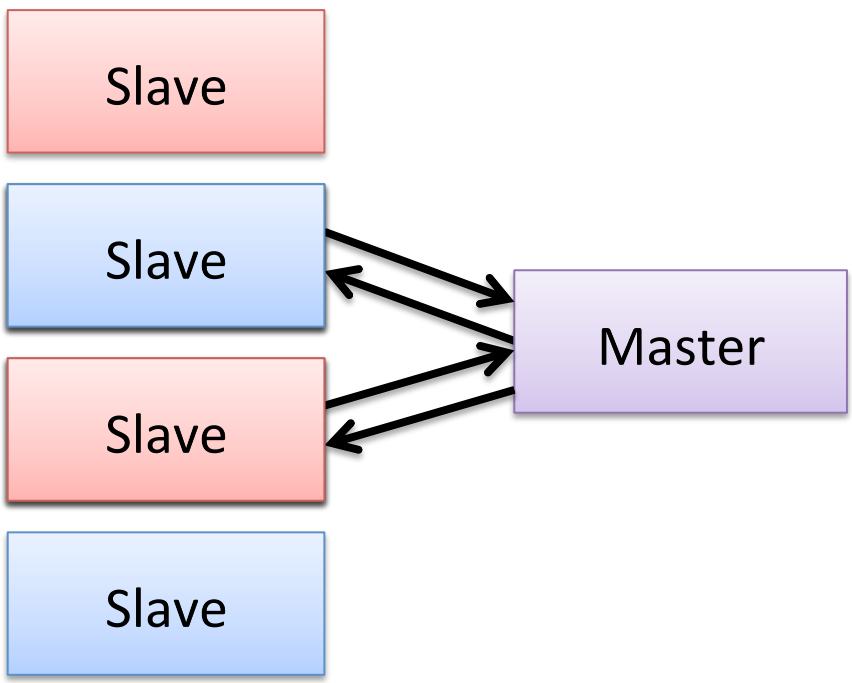
Calculations



Solution: Locality Wait

- Scan through job queue in order of priority
- Jobs must wait before they are allowed to run 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

Locality Wait Example



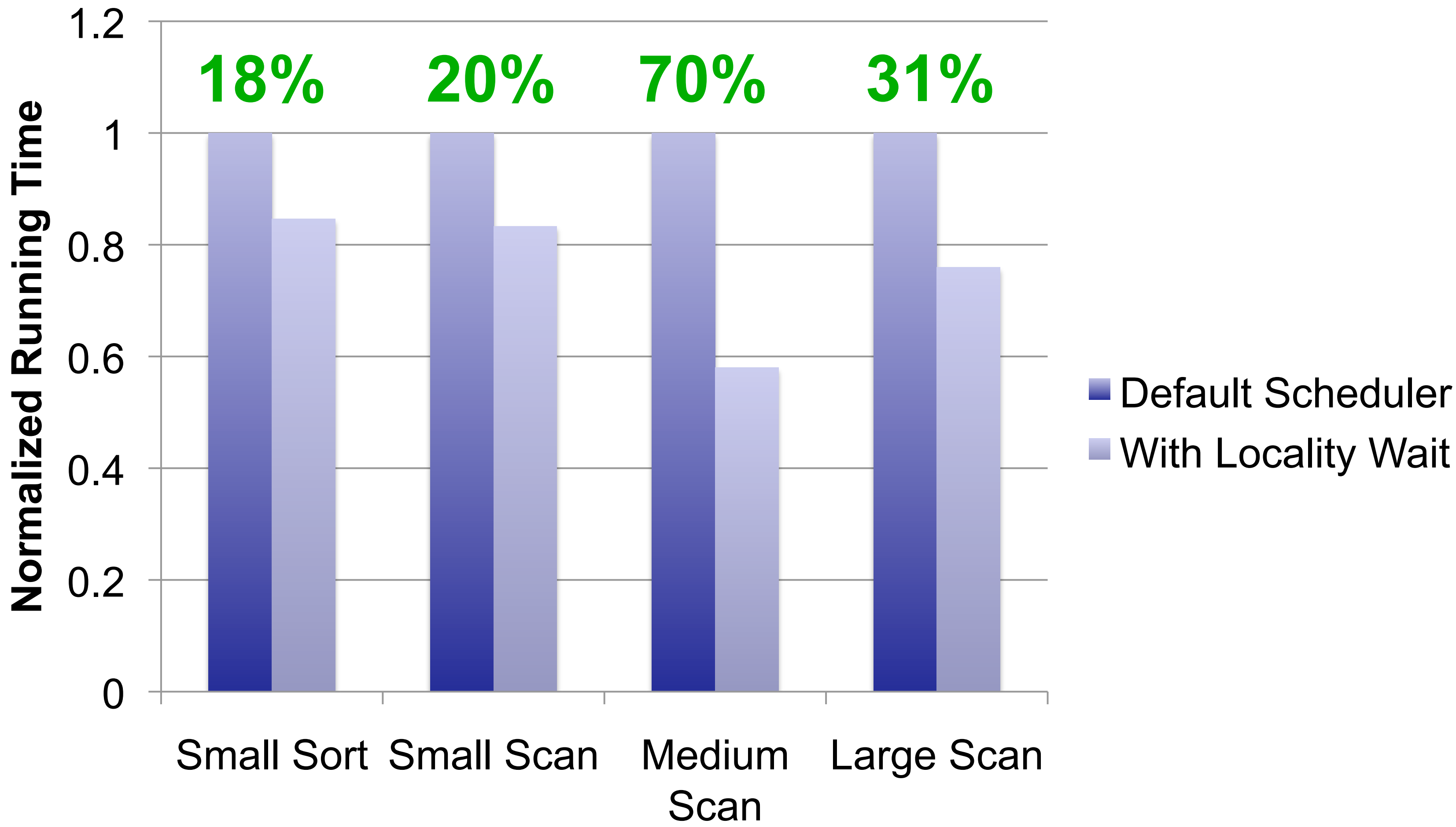
Job	Fair Share	Running Tasks
Job 1	2	**1**
Job 2	2	**3**

Jobs can now shift between slots

Evaluation – Locality Gains

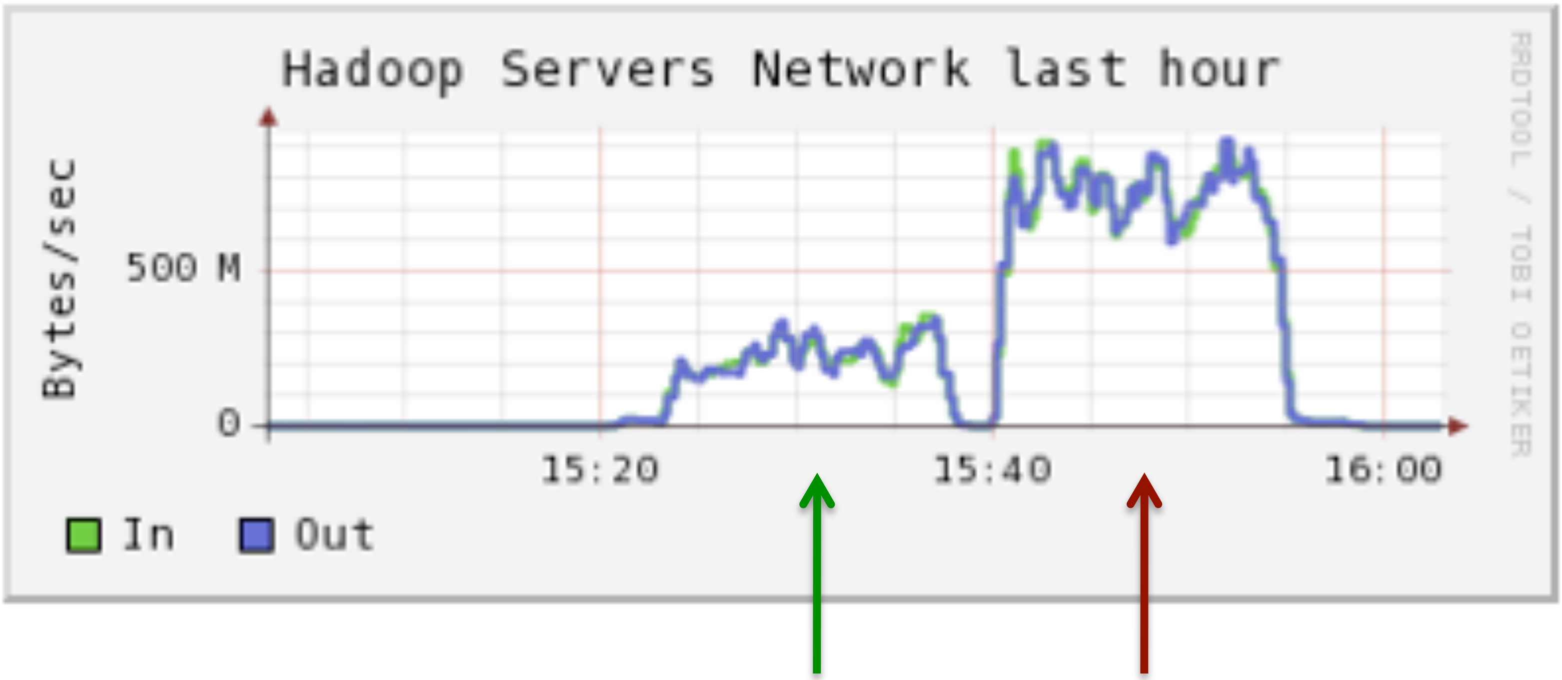
Job Type	Default Scheduler		With Locality Wait	
	Node Loc.	Rack Loc.	Node Loc.	Rack Loc.
Small Sort	2%	50%	81%	96%
Small Scan	2%	50%	75%	94%
Medium Scan	37%	98%	99%	99%
Large Scan	84%	99%	94%	99%

Throughput Gains



Network Traffic Reduction

Network Traffic in Sort Workload



With locality wait

Without locality wait

Further Analysis

- When is it worthwhile to wait, and how long?

- **For throughput:**
  - Always worth it, unless there's a hotspot
  - If hotspot, prefer to run IO-bound tasks on the hotspot node and CPU-bound tasks remotely (rationale: maximize rate of local IO)

Further Analysis

- When is it worthwhile to wait, and how long?

- **For response time:**

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

Wait amount

Expected time to get local heartbeat

Delay from running non-locally

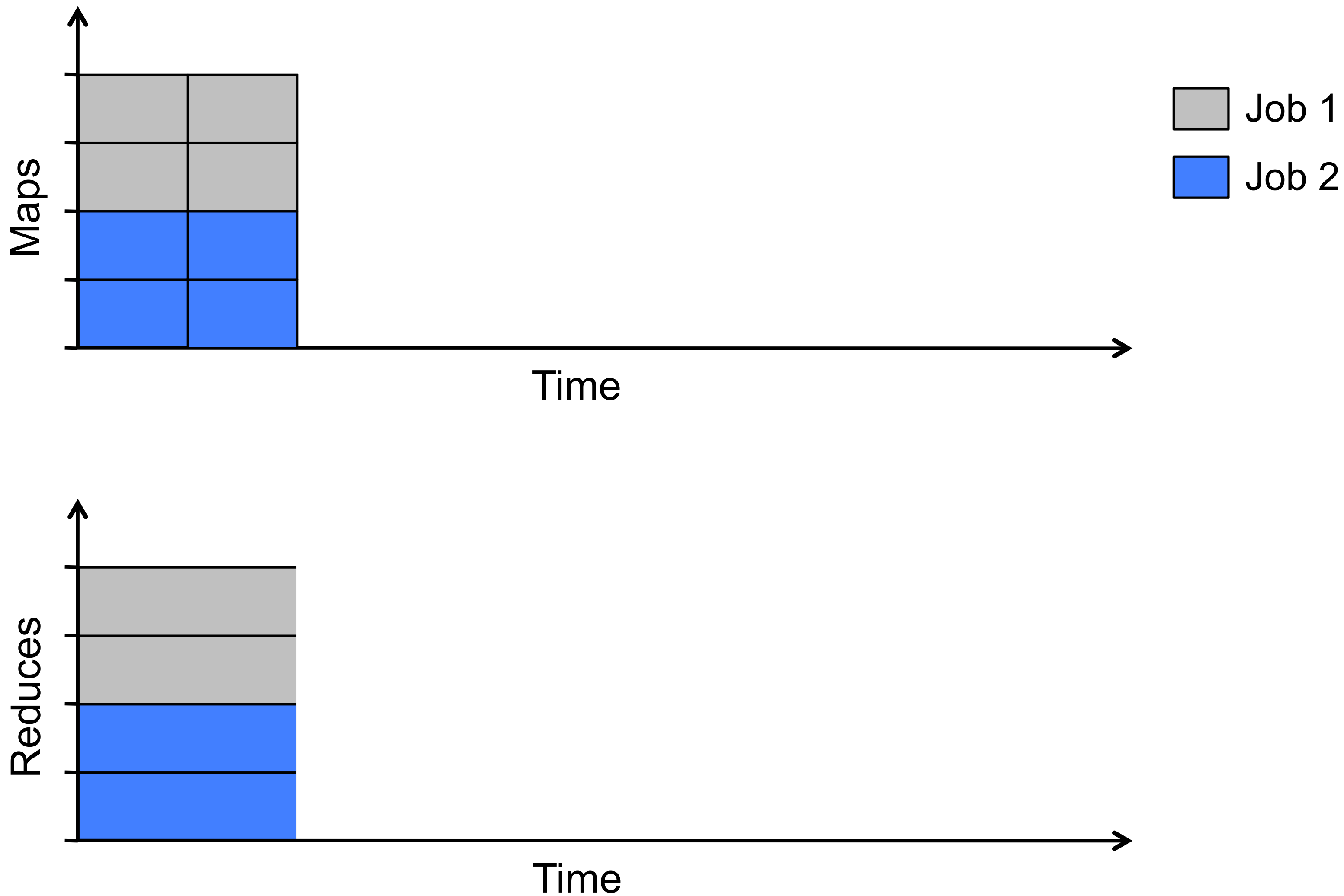
  - Worth it if E(wait) < cost of running non-locally
  - Optimal wait time is infinity

Problem 3: Memory-Aware Scheduling

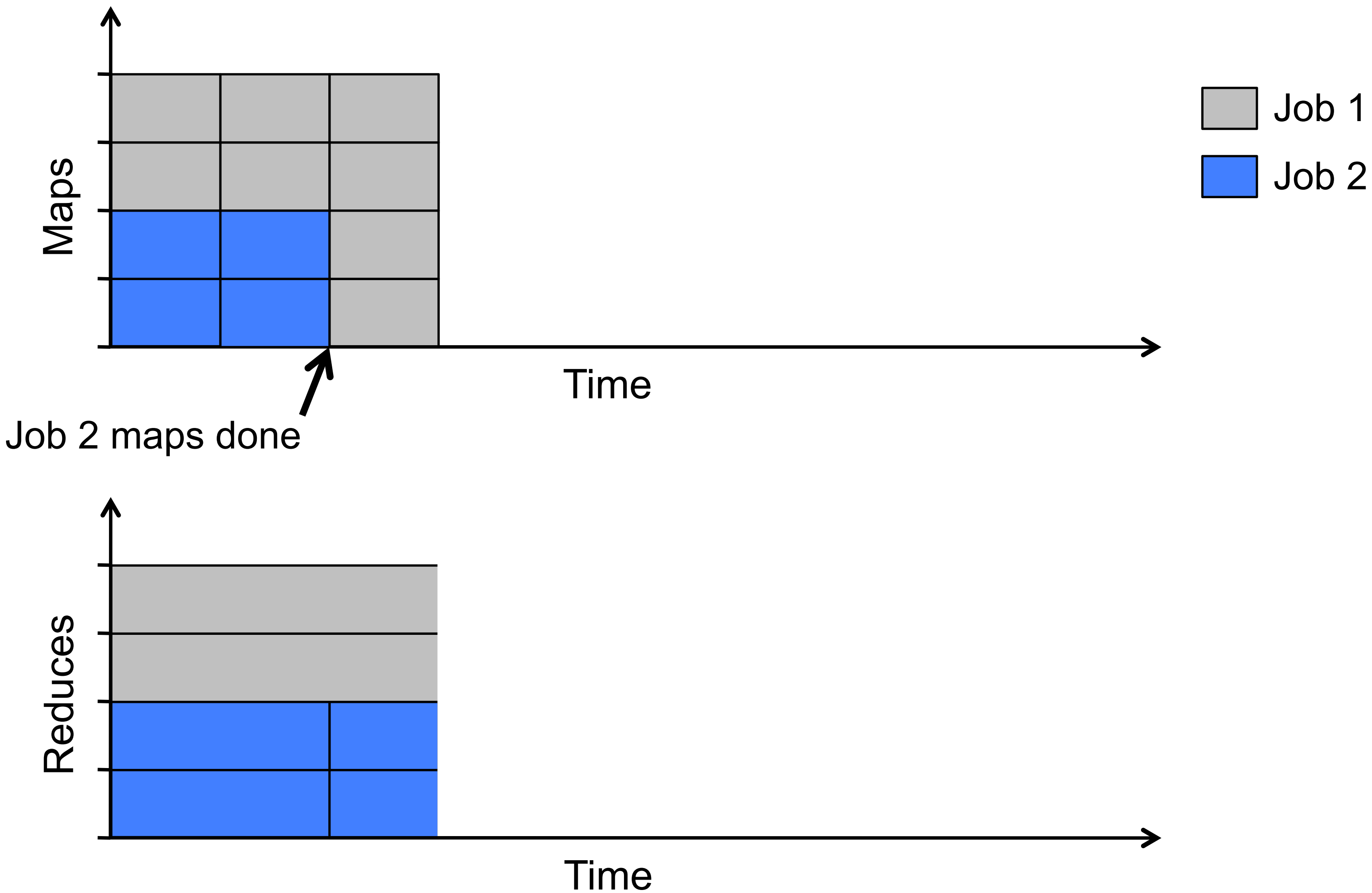
a) How much memory does each job need?
   - Asking users for per-job memory limits leads to overestimation
   - Use historic data about working set size?

b) High-memory jobs may starve
   - Reservation scheme + finish time estimation?

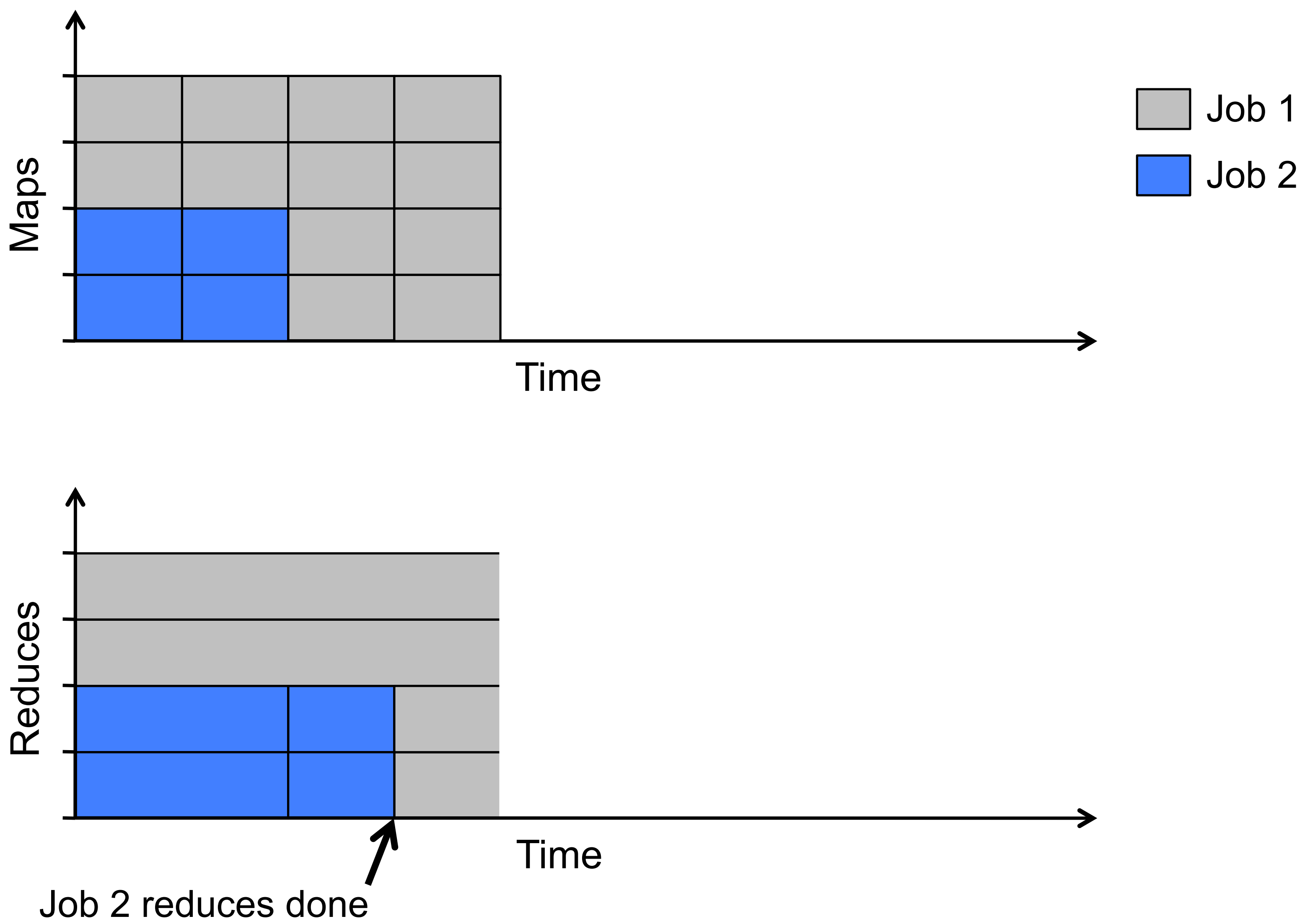
Problem 4: Reduce Scheduling



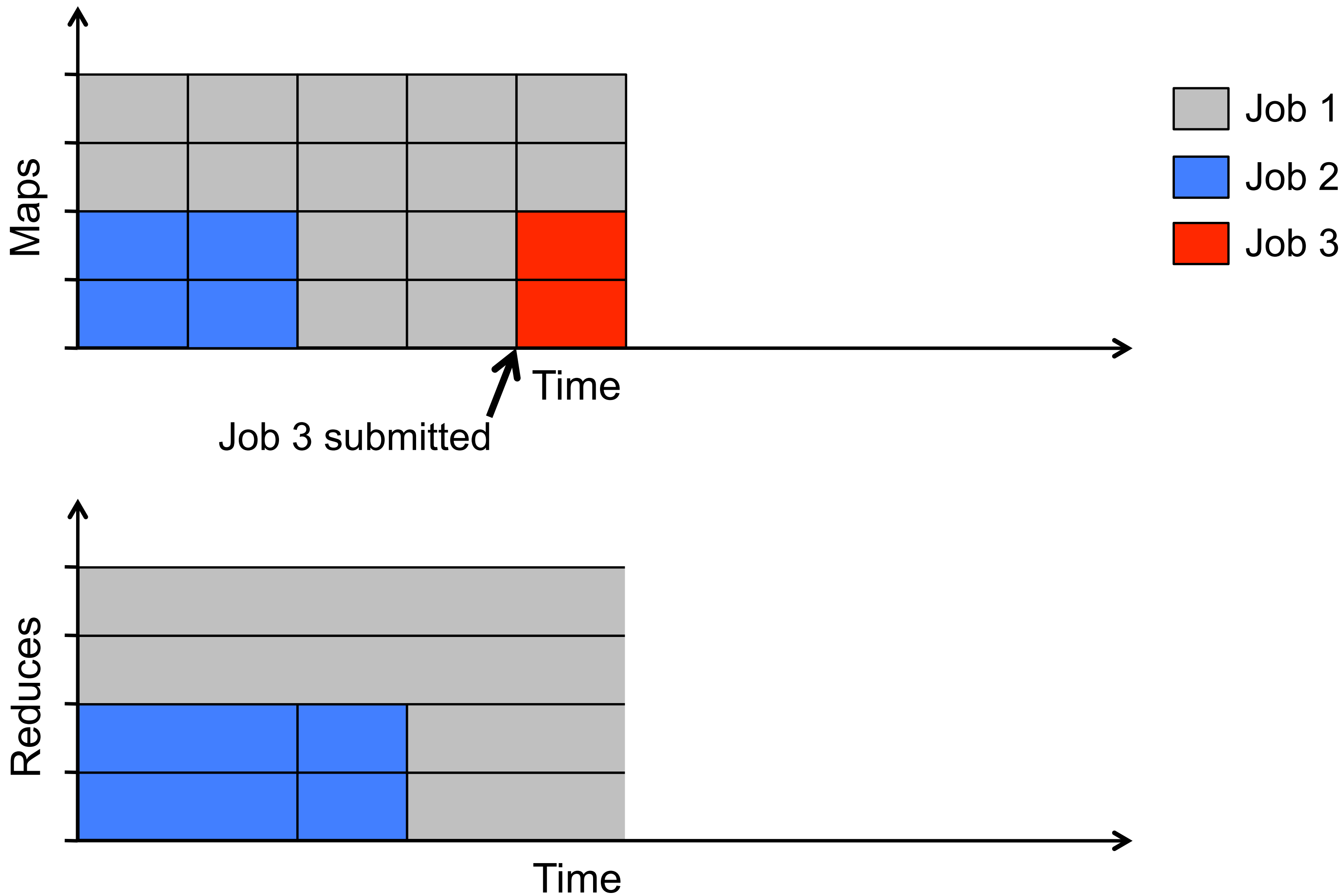
Problem 4: Reduce Scheduling



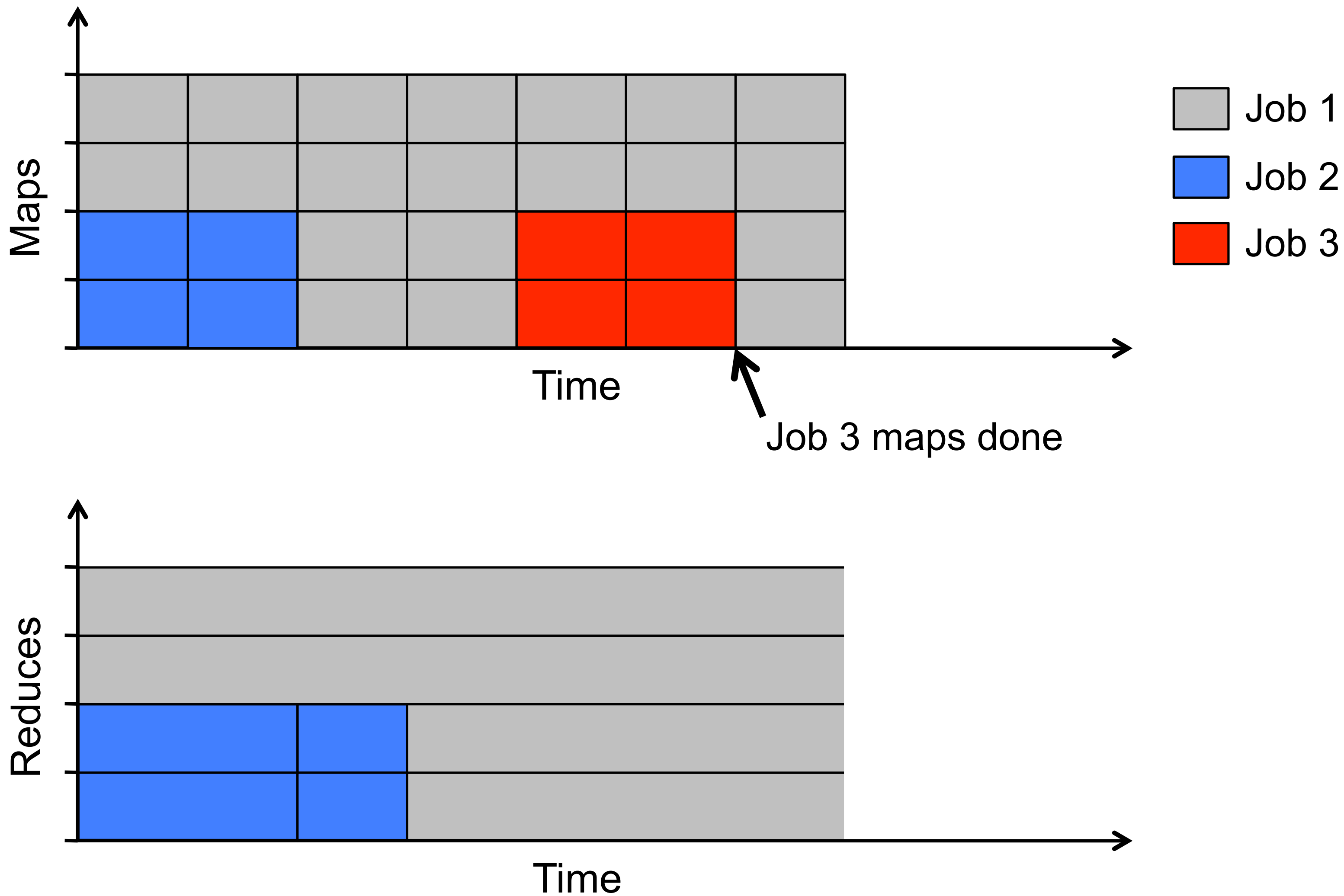
Problem 4: Reduce Scheduling



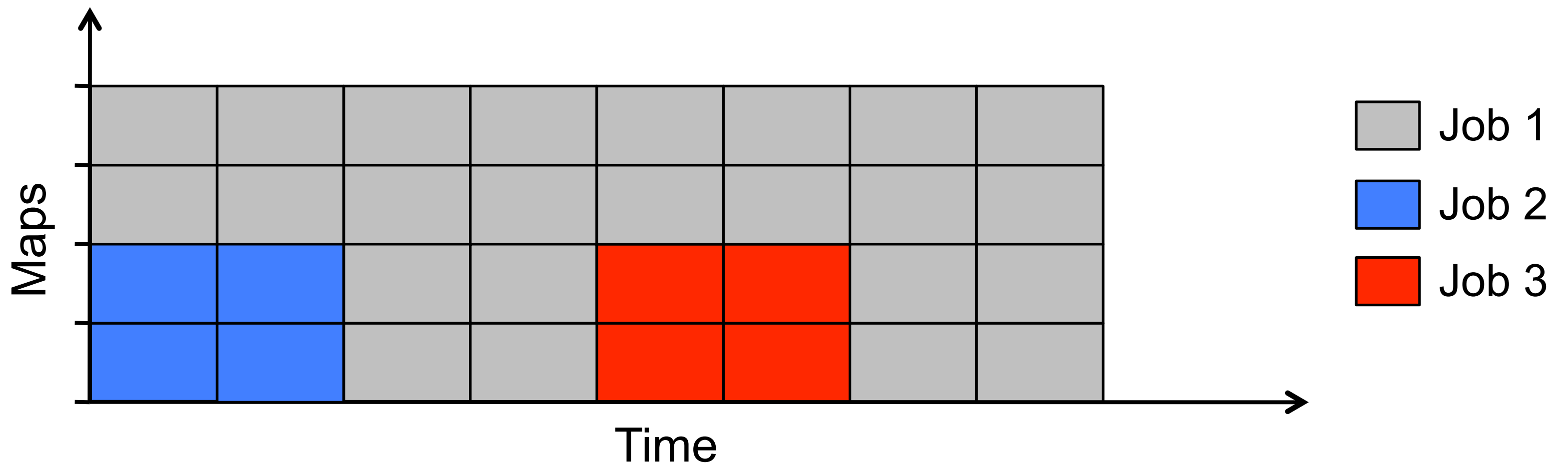
Problem 4: Reduce Scheduling

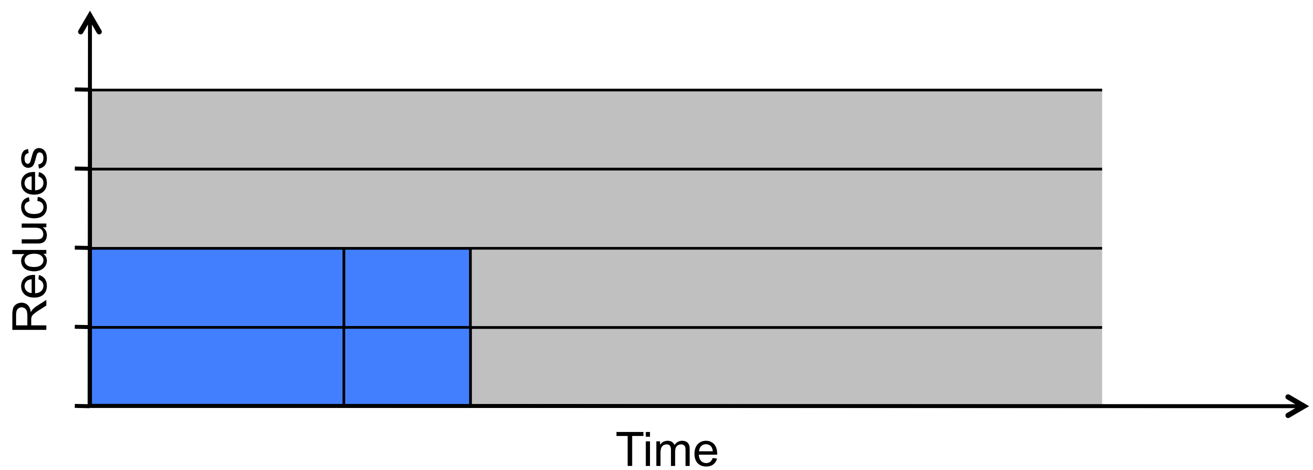


Problem 4: Reduce Scheduling

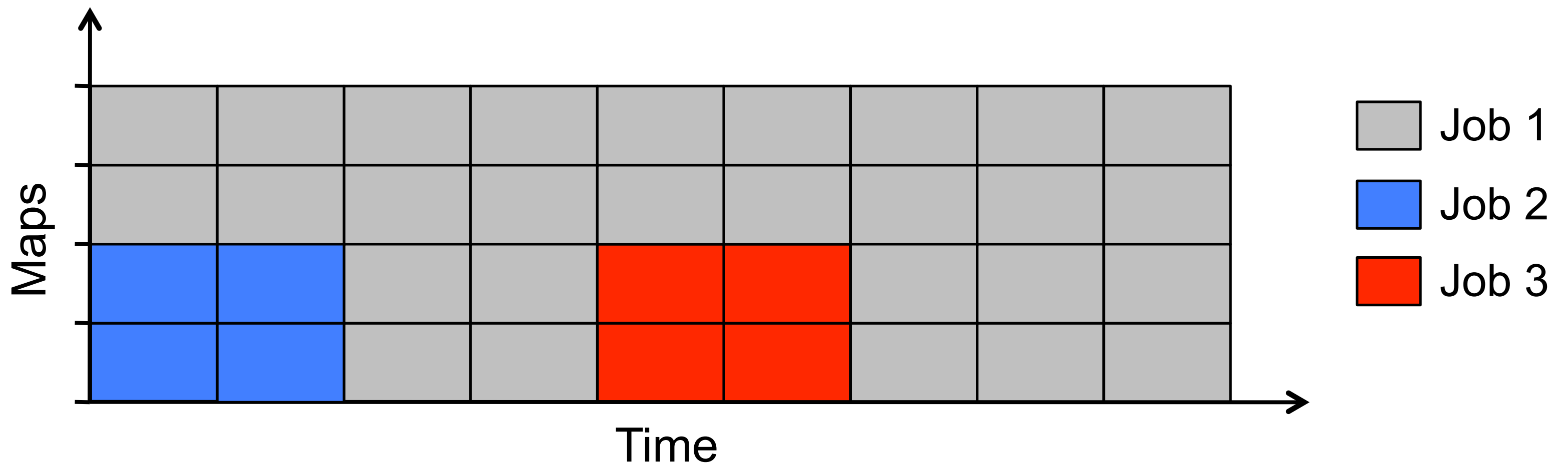


Problem 4: Reduce Scheduling

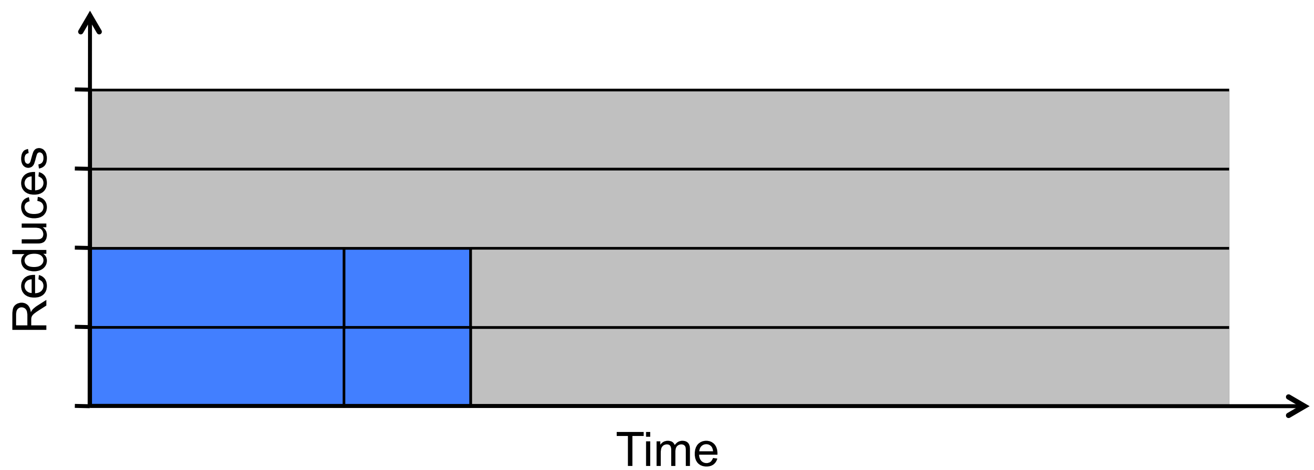




Problem 4: Reduce Scheduling



Problem: Job 3 can't launch reduces until Job 1 finishes



Conclusion

- Simple idea improves throughput by 70%
- Lots of future work:
  - Memory-aware scheduling
  - Reduce scheduling
  - Intermediate-data-aware scheduling
  - Using past history / learning job properties
  - Evaluation using richer benchmarks