A Convenient Framework for Efficient Parallel Multipass Algorithms

Markus Weimer

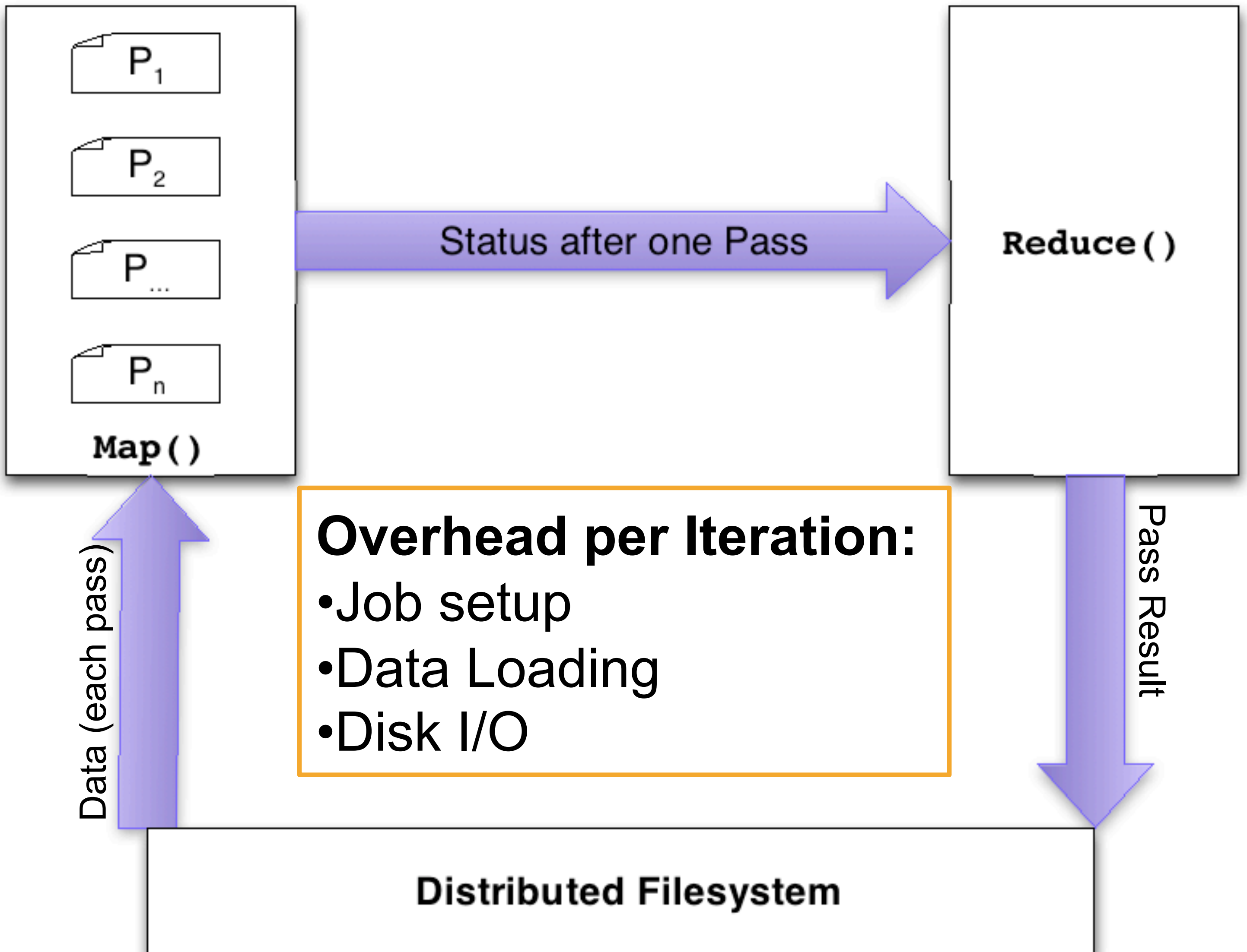
Joint Work with Sriram Rao and Martin Zinkevich

Intro / Point of view taken

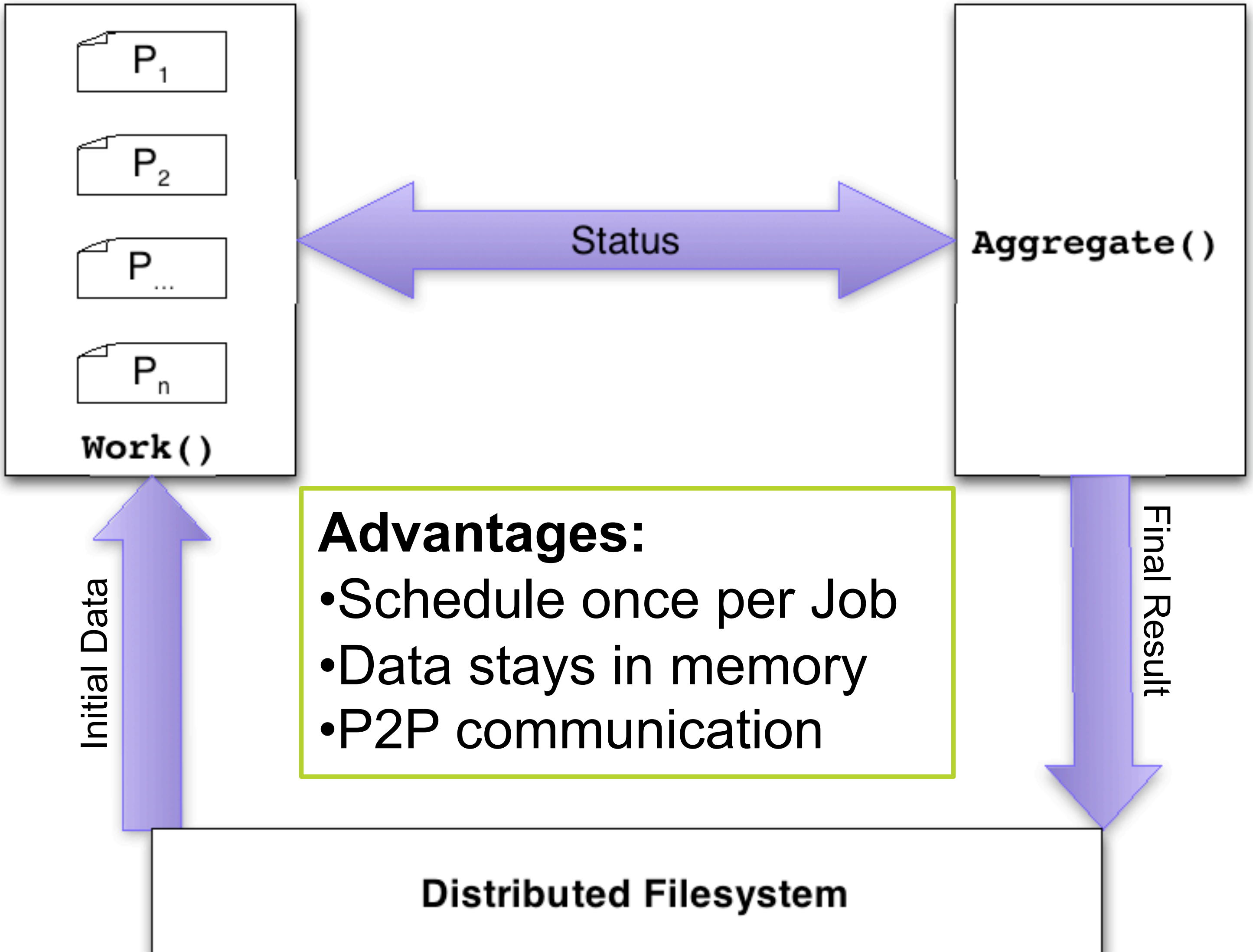
- ML is data compression: from large training data to a small model
- We typically *iterate* over the training data
- The state shared between iterations is relatively small O(model)

→Many algorithms can be expressed as data-parallel loops with synchronization

In MapReduce



Worker/Aggregator



Advantages:

- Schedule once per Job
- Data stays in memory
- P2P communication

Worker

1. Load data
2. Iterate:
   1. Iterates over data
   2. Communicates state
   3. Waits for input state of next pass

Worker

1. Load data
2. Iterate:
   1. **Iterates over data ← user supplied function**
   2. Communicates state
   3. Waits for input state of next pass

Aggregator

- Receive state from the workers
- Aggregate state
- Send state to all workers

Aggregator

- Receive state from the workers
- **Aggregate state ← user supplied**
- Send state to all workers

Failure Handling in the Framework

- **Worker**
  - Meh (SGD)
  - Restart on different machine (else)
- **Aggregator**
  - Restart on different machine
  - Re-request data from workers

Experiments: Parallel Stochastic Gradient Descent

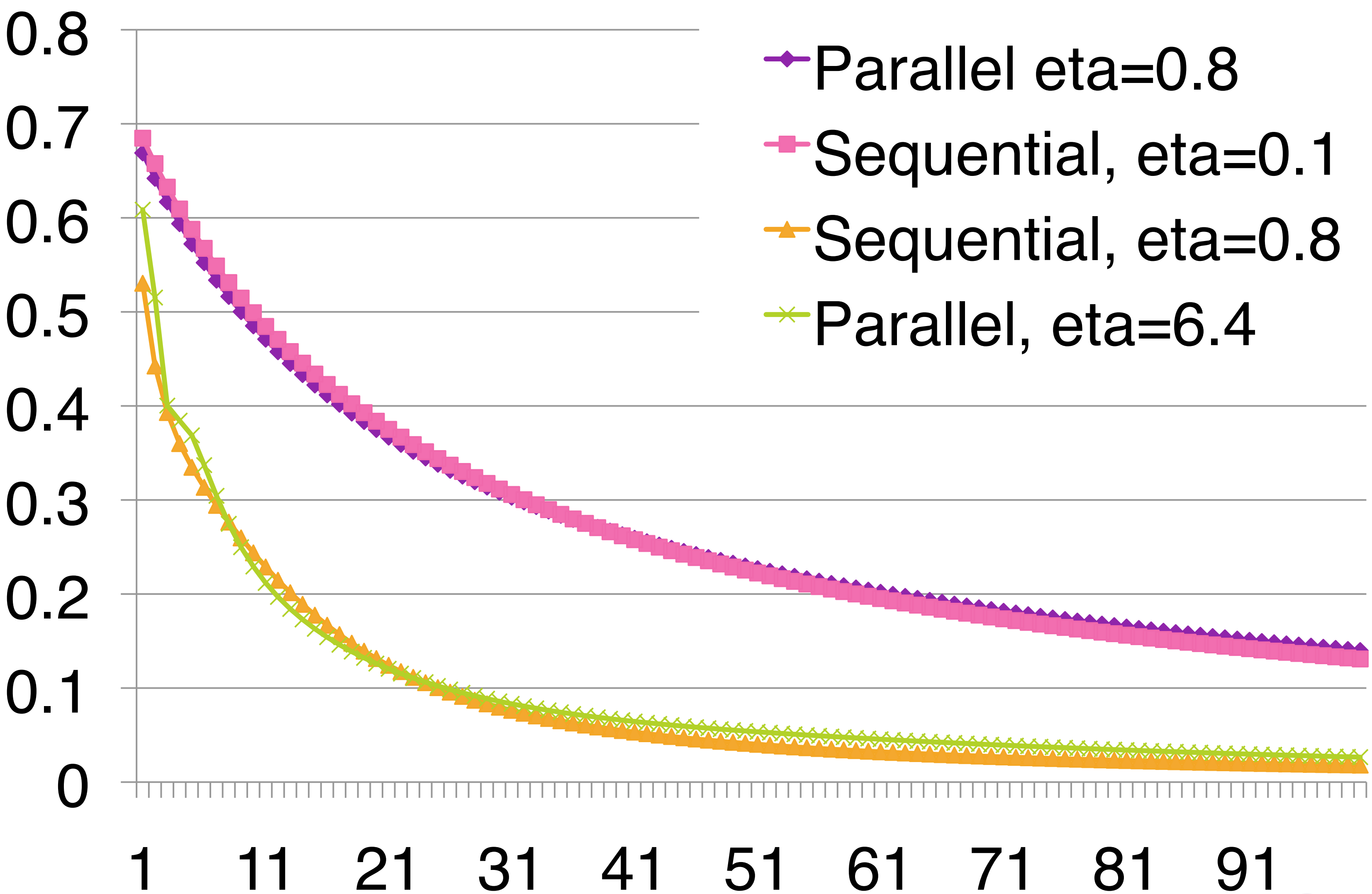
`Work()`

Stochastic Gradient Descent pass

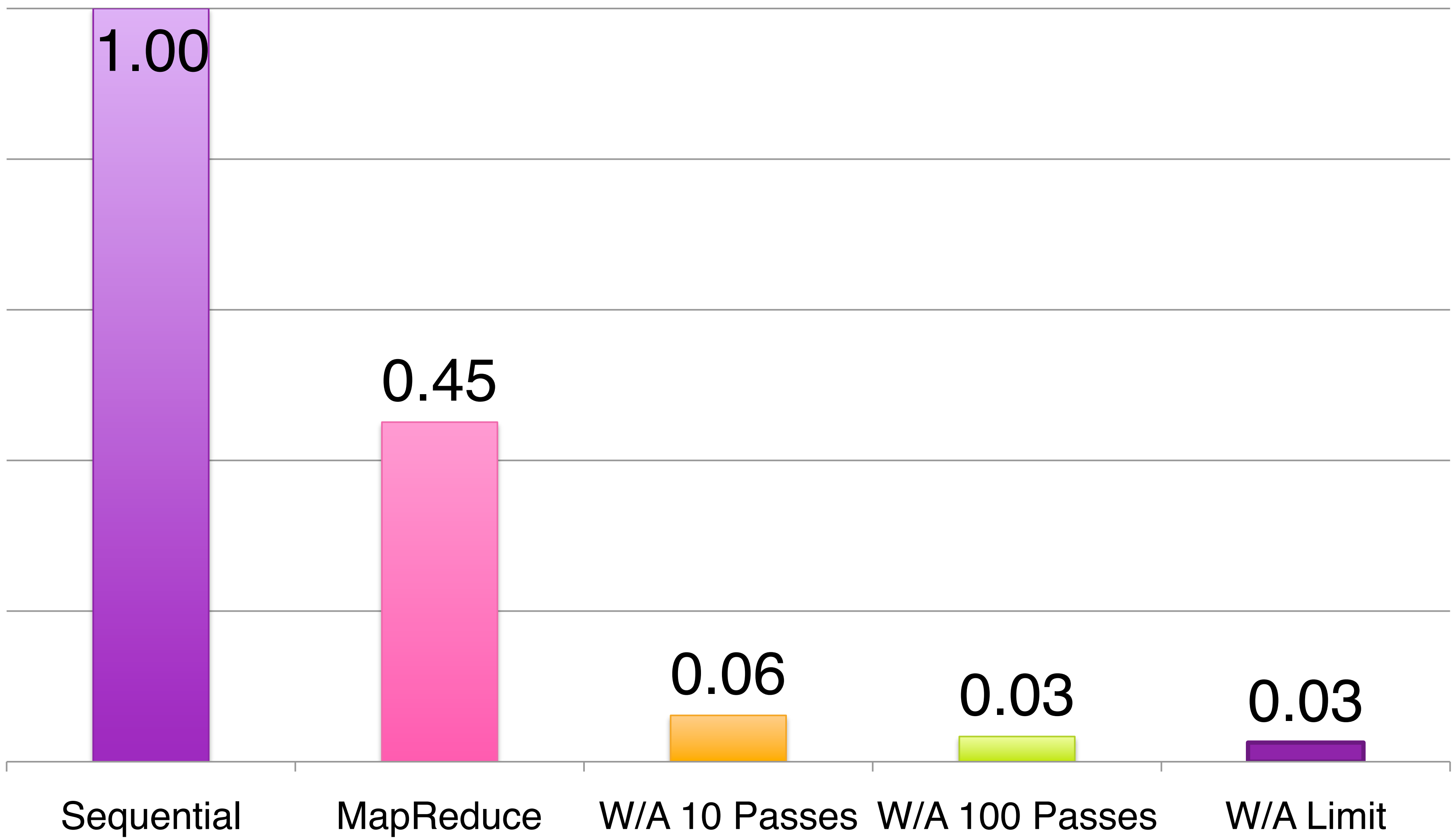
`Aggregate()`

Average Models

Does it work? – Objective over #Passes



Is it fast? – Time per pass (8 machines)



markus
weimer

Yahoo! Labs

weimer@yahoo-inc.com