Experiences with MapReduce, an Abstraction for Large-Scale Computation

Jeff Dean
Google, Inc.

Problem: lots of data

- Example: 20+ billion web pages x 20KB = 400+ terabytes
- One computer can read 30-35 MB/sec from disk
  - ~four months to read the web
- ~1,000 hard drives just to store the web
- Even more to *do* something with the data

Solution: spread the work over many machines

- Good news: same problem with 1000 machines, < 3 hours
- Bad news: programming work
  - communication and coordination
  - recovering from machine failure
  - status reporting
  - debugging
  - optimization
  - locality
- Bad news II: repeat for every problem you want to solve

MapReduce

- A simple programming model that applies to many large-scale computing problems

- Hide messy details in MapReduce runtime library:
  - automatic parallelization
  - load balancing
  - network and disk transfer optimization
  - handling of machine failures
  - robustness
  - **improvements to core library benefit all users of library!**

Typical problem solved by MapReduce

- Read a lot of data
- Map: extract something you care about from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, or transform
- Write the results

Outline stays the same,
map and reduce change to fit the problem

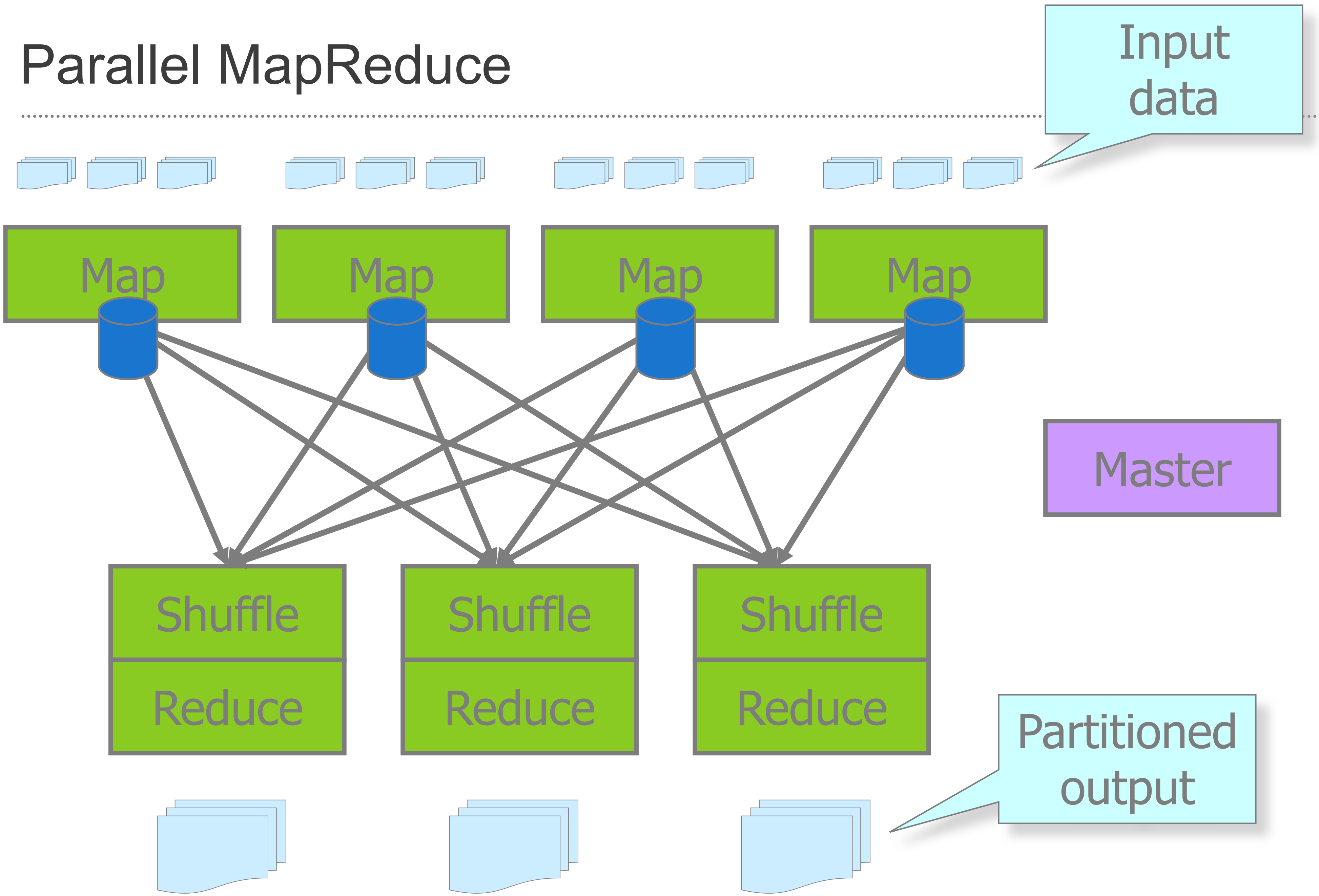
More specifically…

- Programmer specifies two primary methods:
  - map(k, v) → <k', v'>*
  - reduce(k', <v'>*) → <k', v''>*
- All v' with same k' are reduced together, in order.
- Usually also specify:
  - partition(k', total partitions) -> partition for k'
    - often a simple hash of the key
    - allows reduce operations for different k' to be parallelized

MapReduce: Scheduling

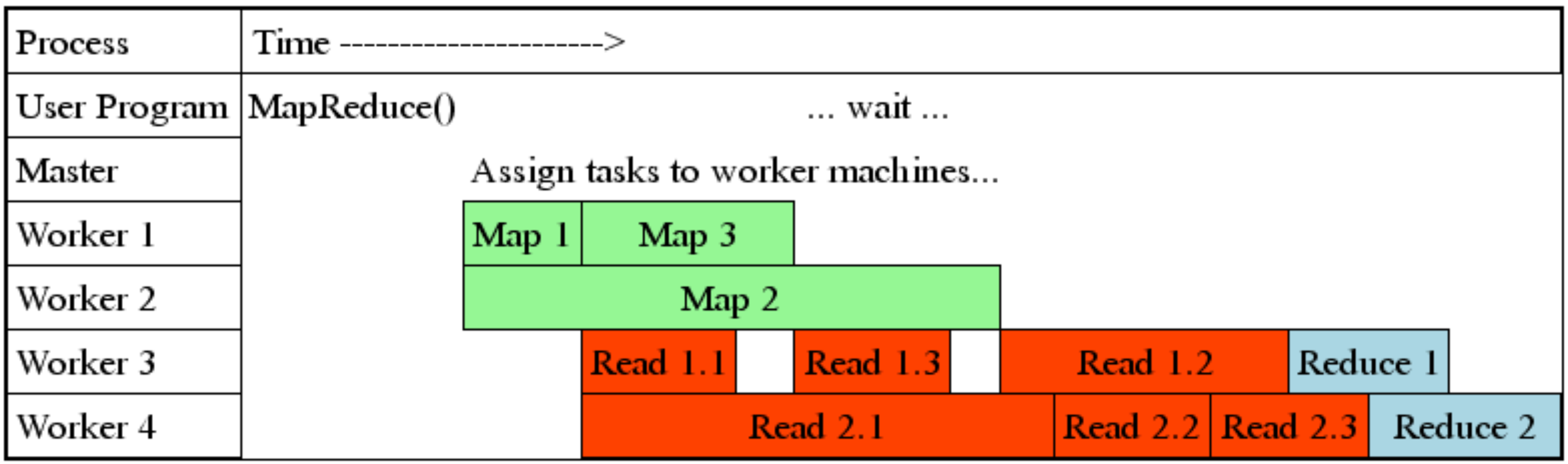
- **One master, many workers**
  - Input data split into *M* map tasks (typically 64 MB in size)
  - Reduce phase partitioned into *R* reduce tasks
  - Tasks are assigned to workers dynamically
  - Often: *M*=200,000; *R*=4,000; workers=2,000
- **Master assigns each map task to a free worker**
  - Considers locality of data to worker when assigning task
  - Worker reads task input (often from local disk!)
  - Worker produces R **local files** containing intermediate k/v pairs
- **Master assigns each reduce task to a free worker**
  - Worker reads intermediate k/v pairs from map workers
  - Worker sorts & applies user's *Reduce* op to produce the output

Parallel MapReduce



Task Granularity and Pipelining

- Fine granularity tasks: many more map tasks than machines
  - Minimizes time for fault recovery
  - Can pipeline shuffling with map execution
  - Better dynamic load balancing
- Often use 200,000 map/5000 reduce tasks w/ 2000 machines



Conclusion

- MapReduce has proven to be a remarkably-useful abstraction
- Greatly simplifies large-scale computations at Google
- Fun to use: focus on problem, let library deal with messy details
  - Many thousands of parallel programs written by hundreds of different programmers in last few years
  - Many had no prior parallel or distributed programming experience

Further info:

MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, OSDI' 04

http://labs.google.com/papers/mapreduce.html

(or search Google for [MapReduce])