MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

Presenter: Aditya Devarakonda

September 1, 2015

Programming Model

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Execution Model

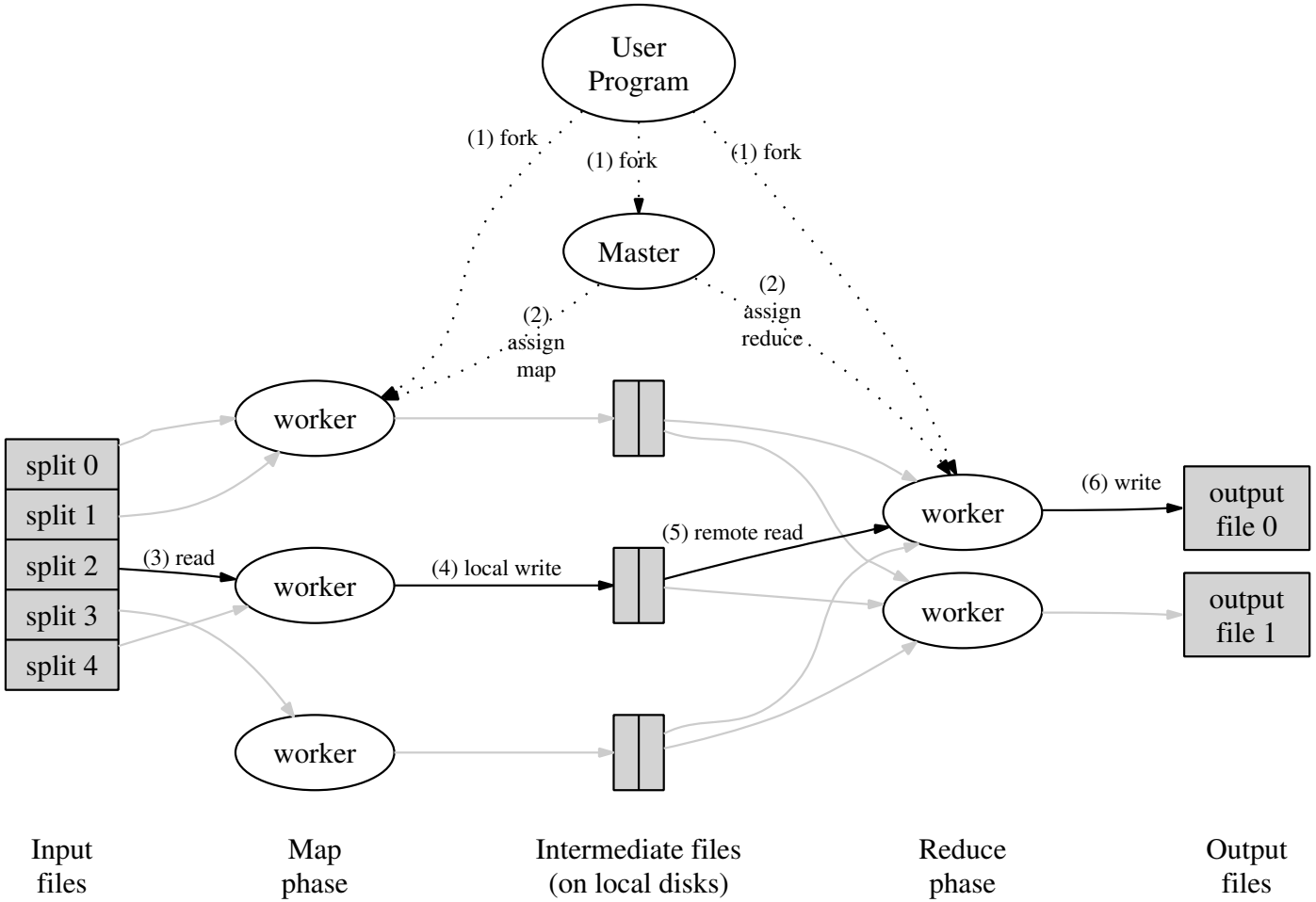


Figure 1: Execution overview

Prior Art

- Programming models: OpenMP, MPI, UPC, Charm++.

Prior Art

- Programming models: OpenMP, MPI, UPC, Charm++.
- Resource management: SLURM, HTCondor.

Prior Art

- Programming models: OpenMP, MPI, UPC, Charm++.
- Resource management: SLURM, HTCondor.
- Machine setup: Supercomputer, network of computers.

Key Players in MapReduce

- Master

Key Players in MapReduce

- Master
- Workers (Mappers and Reducers)

Key Players in MapReduce

- Master
- Workers (Mappers and Reducers)
- Combiners

Key Players in MapReduce

- Master
- Workers (Mappers and Reducers)
- Combiners
- Partitioners

Fault Tolerance

- Worker failure

Fault Tolerance

- Worker failure
- Master failure

Parallel Computing Issues

- Locality

Parallel Computing Issues

- Locality
- Load balancing

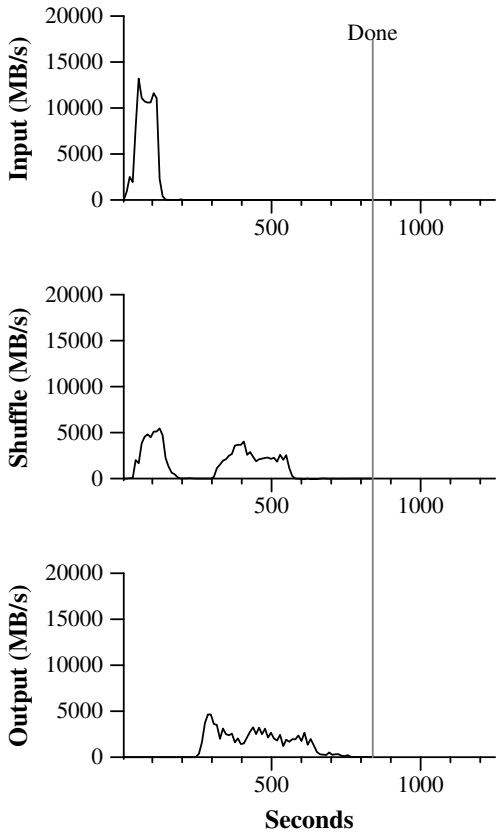
Parallel Computing Issues

- Locality
- Load balancing
- Communication

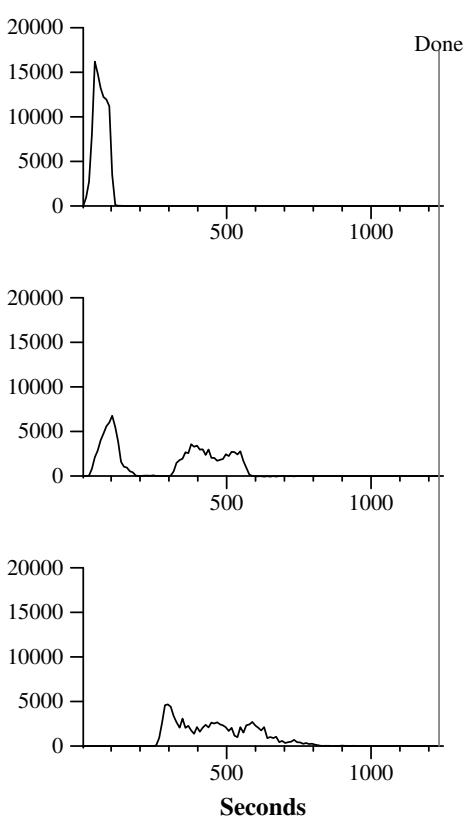
Parallel Computing Issues

- Locality
- Load balancing
- Communication
- Invariants (What assumptions does MR make).

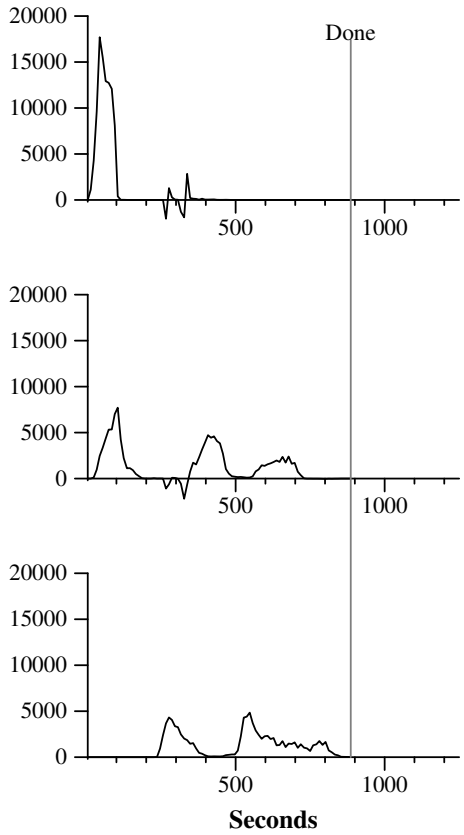
Performance on 1TB Sort



(a) Normal execution



(b) No backup tasks



(c) 200 tasks killed

Usage at Google

Number of jobs	29,423
Average job completion time	634 secs
Machine days used	79,186 days
Input data read	3,288 TB
Intermediate data produced	758 TB
Output data written	193 TB
Average worker machines per job	157
Average worker deaths per job	1.2
Average map tasks per job	3,351
Average reduce tasks per job	55
Unique *map* implementations	395
Unique *reduce* implementations	269
Unique *map/reduce* combinations	426

Conclusion

"[MapReduce] allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library."[Introduction, DG04]

My take away: Simplicity at the cost of generality and performance. However, it is a novel, easy-to-use, quick-to-production system for *simple* data parallel programs and applications on *commodity* clusters where fault tolerance is a primary goal.