

MapReduce: Simplified Data Processing on Large Clusters

Naser AlDuaij

MapReduce: Benefits

- Programming model for processing/generating large data sets
- Easy to use
- Highly scalable (order of TBs of data)

MapReduce: Features

- Parallelization
- Fault-tolerance
- Locality optimization (avoid network overhead)
- Load balancing

MapReduce: Real-life applications

- Google's production web search service
- Sorting
- Data mining
- Machine learning
- Graph computations

MapReduce: Map & Reduce

- Distributed divide & conquer approach
- Input/output key/value pairs
- Map: Master node partitions input
- Reduce: Master node collects & combines data

MapReduce: Example

Word frequency

```
void map(String name, String document):  
    // name: document name  
    // document: document contents  
    for each word w in document:  
        EmitIntermediate(w, "1");  
  
void reduce(String word, Iterator partialCounts):  
    // word: a word  
    // partialCounts: a list of aggregated partial counts  
    int sum = 0;  
    for each pc in partialCounts:  
        sum += ParseInt(pc);  
    Emit(word, AsString(sum));
```

MapReduce: Types

Map (k1, v1) → list (k2, v2)

Reduce (k2, list (v2)) → list (v2)

- Input k,v ≠ Output k,v data domain
- Word Frequency example:

Map (string, string) → list (partial_str, int)

Reduce (partial_str, Iterator (list (int))) → list (int)

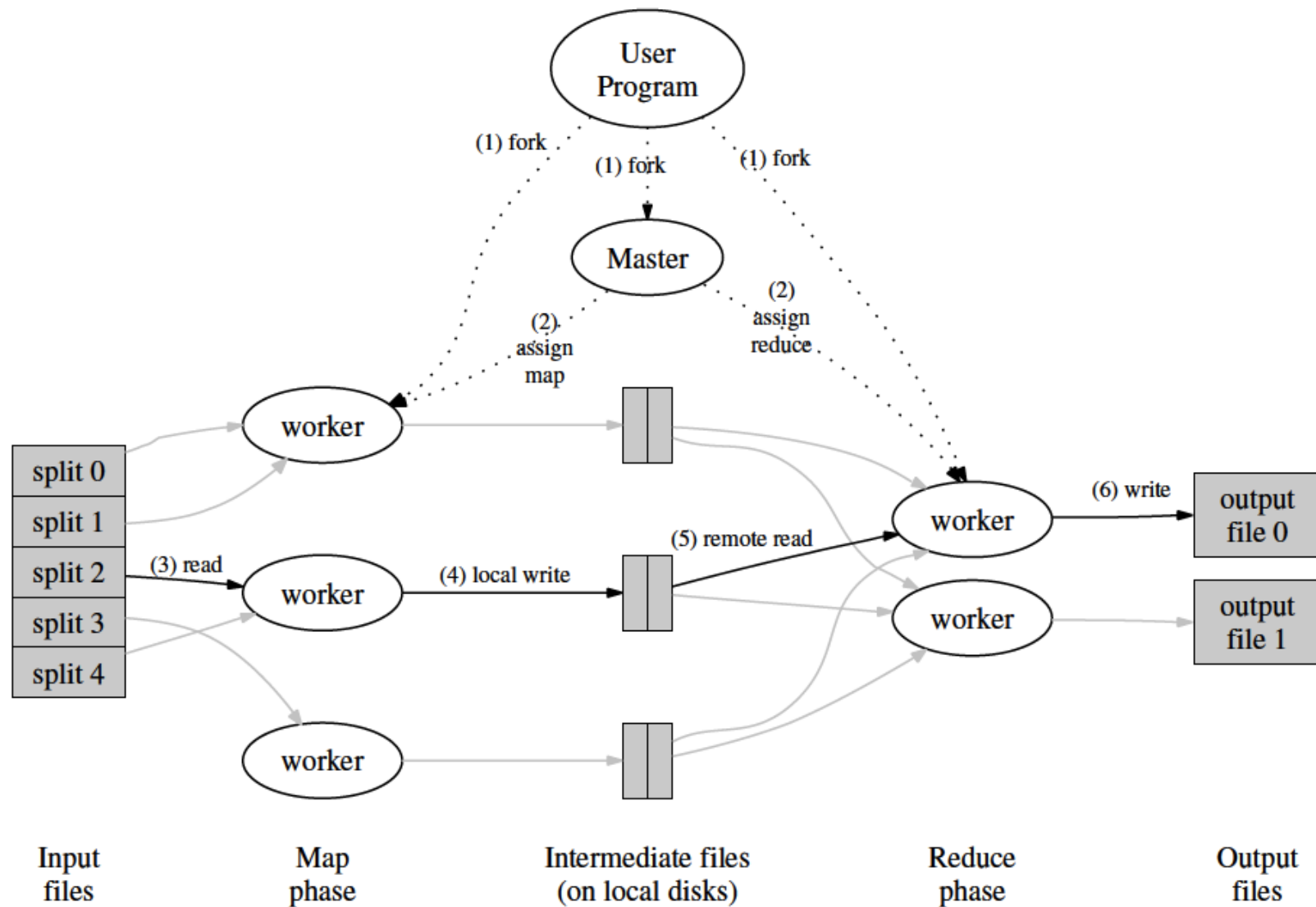
MapReduce: More Examples

- Distributed Grep: Map <pattern> – Reduce <collect output>
- URL Access Frequency: Map <URL, 1> – Reduce <URL, total count>
- Reverse Web-Link Graph: Map <target, src> - Reduce <target, list(src)>

MapReduce: More Examples

- Term-Vector per Host: Map <hostname, term vec> - Reduce <hostname, term vec>
- Inverted Index: Map <word, doc ID> - Reduce <word, list(doc ID)>
- Distributed Sort: Map <key, rec> - Reduce <key, rec>

MapReduce: Execution Overview



MapReduce: Execution Overview

- M map tasks, R reduce tasks / output files
- Input partitioned into M splits (~16-64 MB/piece)
- R pieces using tweak-able partitioning function
- Ideally, M && R >> # of worker machines
- Task states: <idle, in-progress, completed>

How is master node chosen?

MapReduce: Fault Tolerance

- “Master pings every worker periodically”
- Completed map tasks re-executed on failure (locality)
- Master redistributes tasks for failed workers
- MapReduce is aborted on master failure

MapReduce: Backup Tasks

“Stragglers”: Slow machines that hinder completion

- Solution:

Schedule backup tasks for in-progress tasks towards the end of MapReduce run

MapReduce: Refinements

- Partitioning function (default: $\text{Hash}(\text{key}) \bmod R$)
- Key/value pairs processed in increasing order (sorted output file per partition)
- Combiner function: Performed in map with results stored in intermediate file
- Implementable *reader* interface for new input types

MapReduce: Side-effects & Debugging

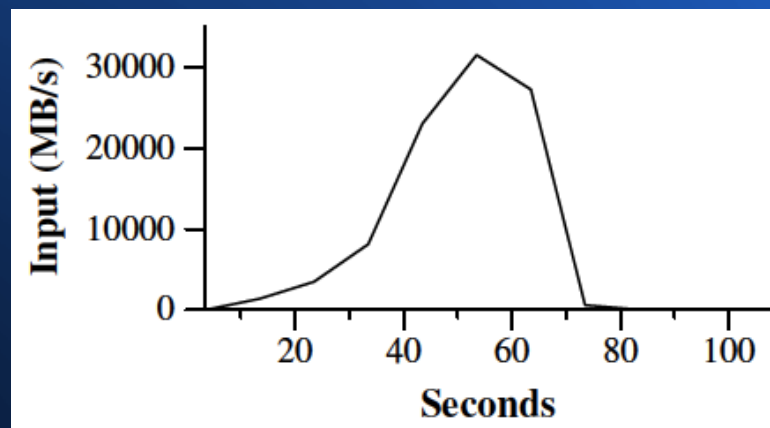
- Users are responsible for output files
- “No support for atomic two-phase commits of multiple output files”
- Signal handler for worker process (UDP packet)
- Option to run MapReduce on one local machine
- “Counter” facility for counting (e.g. # of words)
- Status information page with statistics

MapReduce: Performance

- Two computations: Search and sort (~1TB)
- Cluster: ~1800 Machines
- Input split into 64 MB pieces
- $M = 15000$, $R = 1$ for Grep and $R = 4000$ for sort

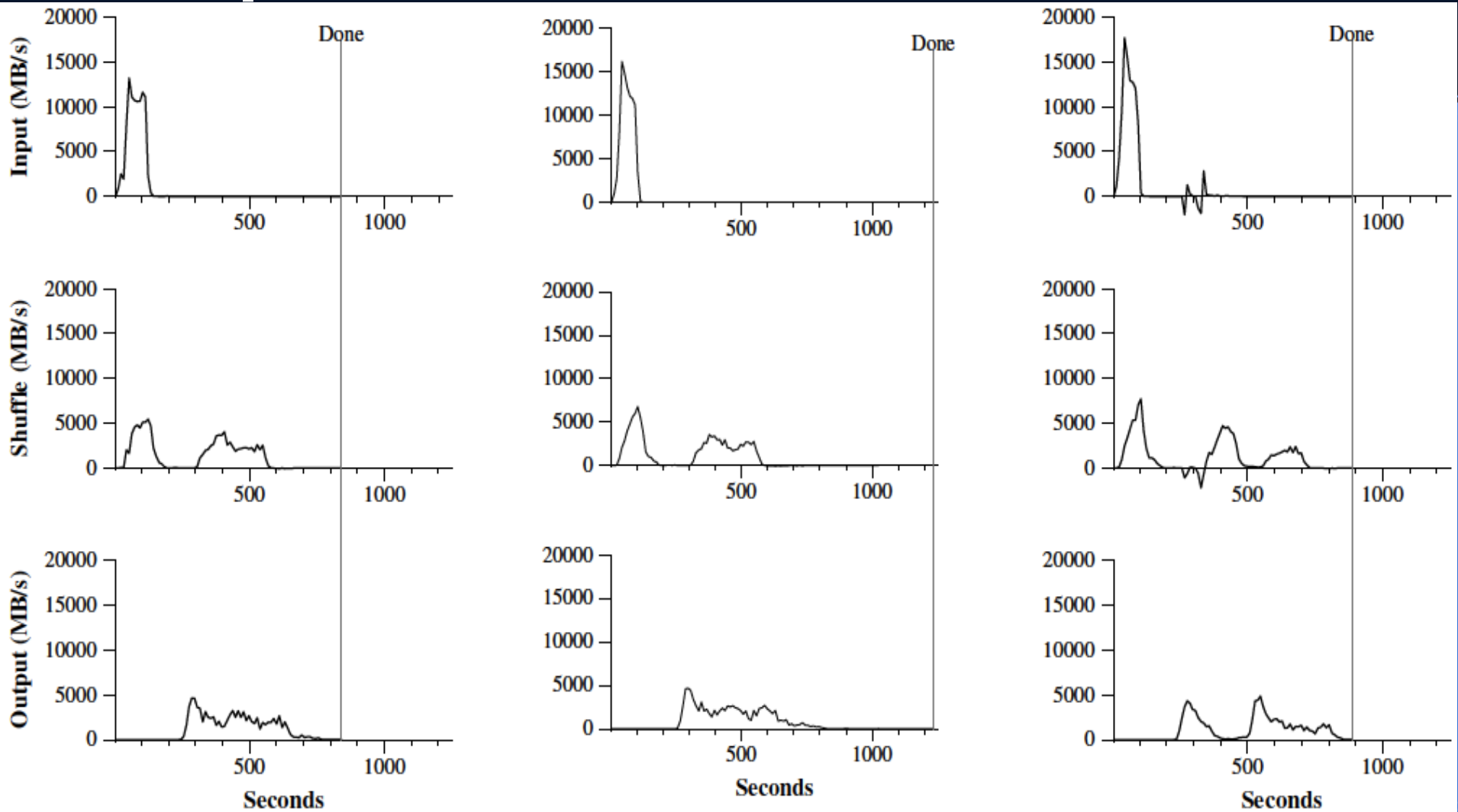
MapReduce: Performance - Grep

- Search for rare three-character pattern



- ~150 seconds to complete

MapReduce: Performance - Sort

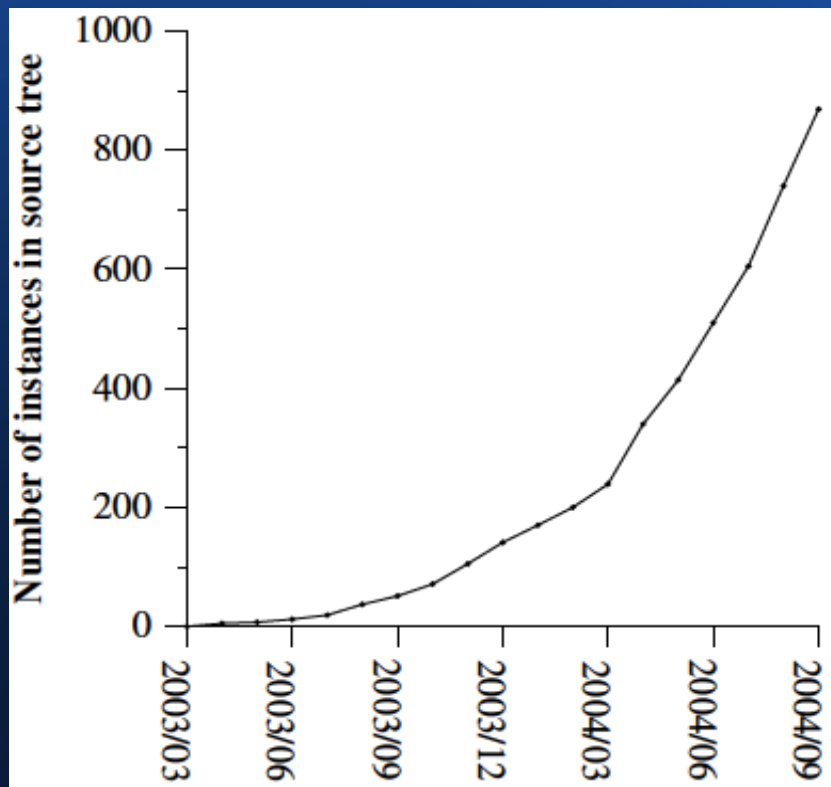


(a) Normal execution

(b) No backup tasks

(c) 200 tasks killed

MapReduce: Statistics



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 <i>map</i> implementations	395
Unique <i>reduce</i> implementations	269
Unique <i>map/reduce</i> combinations	426

MapReduce: Google's web search indexing

- Code is “Simpler, smaller, easier to understand”
- 3800 → 700 lines of code with MapReduce
- Good performance with fault tolerance

MapReduce: Related work

- Parallelization (Bulk Synchronous Programming)
- Locality optimization (Active disks)
- Eager scheduling mechanism (Charlotte System)
- Cluster management system (Condor)
- Sorting facility (NOW-Sort)
- Sending data over distributed queues (River)
- Fault tolerance (BAD-FS, TACC)

MapReduce: Simplified Data Processing on Large Clusters

Questions?

Thank you