MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

OSDI 2004

Presented by Zachary Bischof

Outline

- Motivation
- Summary
- Example
- Implementation
- Discussion

Motivation for MapReduce

- Parallel programming is hard
- Goal of MapReduce:
  - Process a lot of data (terabytes) over a lot of machines (hundreds or thousands)
  - Hide details of parallelization
- Need a new programming model

Summary

- How does MapReduce help?
  - Hides details of parallelization from programmer
    - Provides some transparency
  - Handles fault-tolerance
  - Data distribution is automatic
  - Does load balancing and scheduling
  - Monitors the status of systems and overall progress of the program
- Uses Google File System (GFS)

How it Works

- Restricts the programming model
  - Divide work into key/value pairs
  - Makes it easier to use
- Programmers write *Map* and *Reduce* functions
- MapReduce handles the rest

Map

- User-defined
- Input: key/value pair
  - (input key, input value)
- Output: List of intermediate key/value pairs
  - list(output key, intermediate value)
- Analysis of a worker's dataset produces intermediate values
  - Input and intermediate values may be from a different domain

Reduce

- User-defined
- Input: intermediate key/value pairs
  - (output key, list(intermediate value)
- Output: output keys and values
  - list(output value)
- Merges together all intermediate values for a particular key into a new set of values
  - Output is often one value but does not have to be

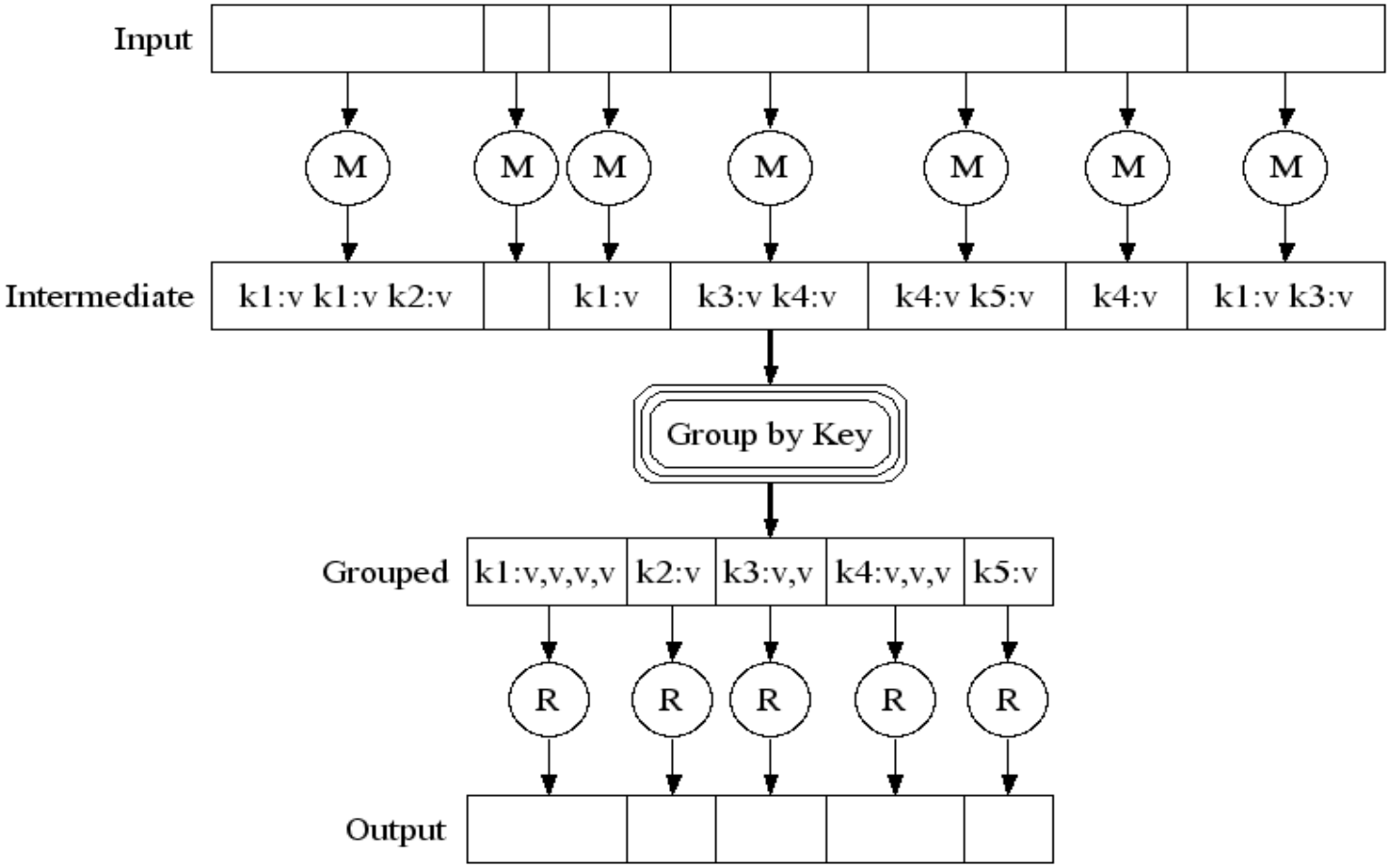
Example: Counting words

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


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

Example (cont'd)

- Document is split up for workers
- Map step:
  - Each word gets an initial value of "1"
  - Each word is a key with a list of values
- Reduce Step:
  - Takes a key (in this case a word), and a list of values (all "1")
  - Adds them up
  - Passes them up the tree


Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output

Other Examples

- Distributed Grep
- Distributed Sort
- Machine Learning
- Reverse Web-Link Graph
- And more…

Implementation

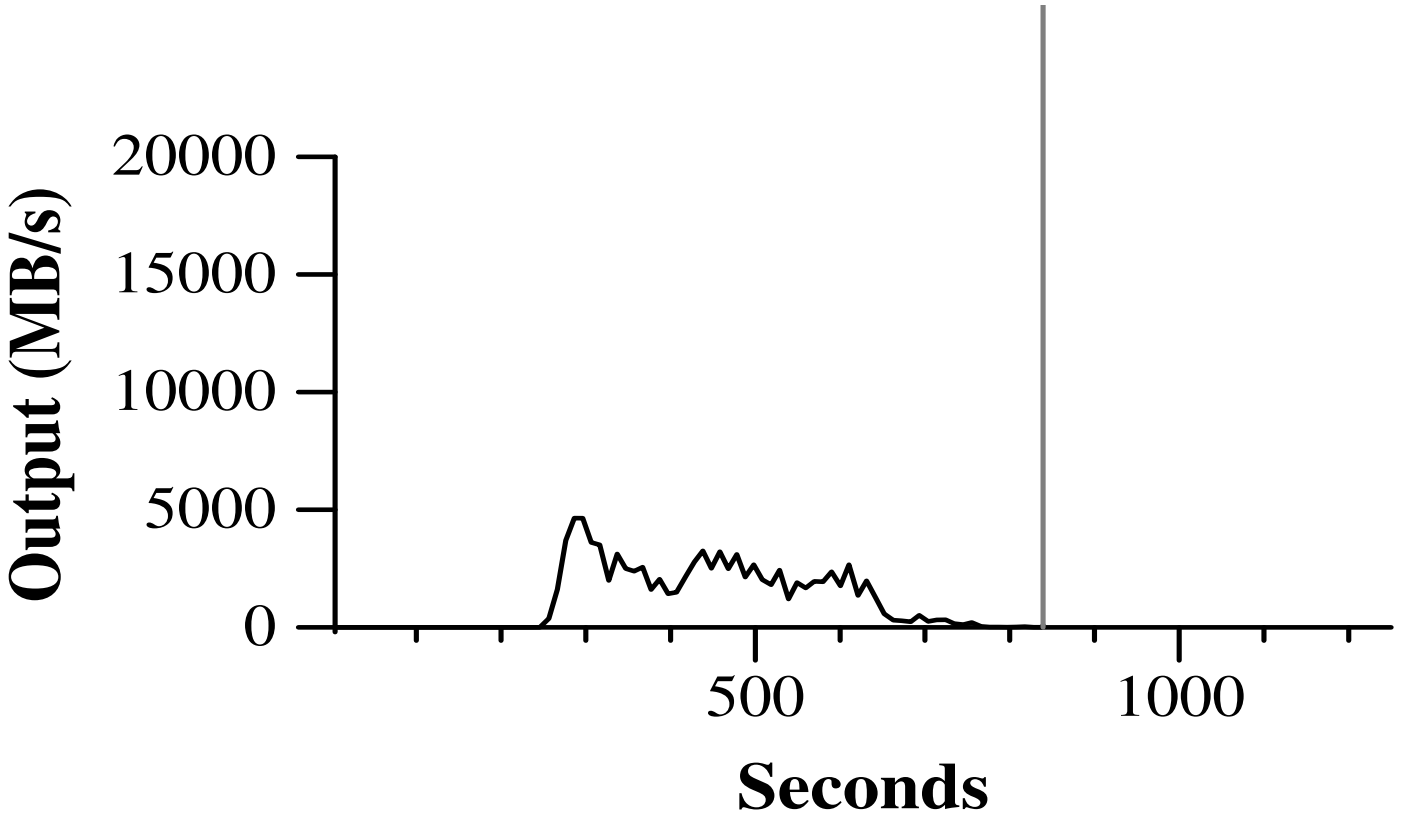
- Uses a distributed file system to manage data
  - GFS (SOSP 2003)
- Bandwidth is a bottleneck
  - Request data location from GFS
  - Assign tasks to the same machine or one on the same switch (localizes activity)
- Combiner Function
  - Do partial merging of intermediate keys
  - Reduce network traffic

Implementation

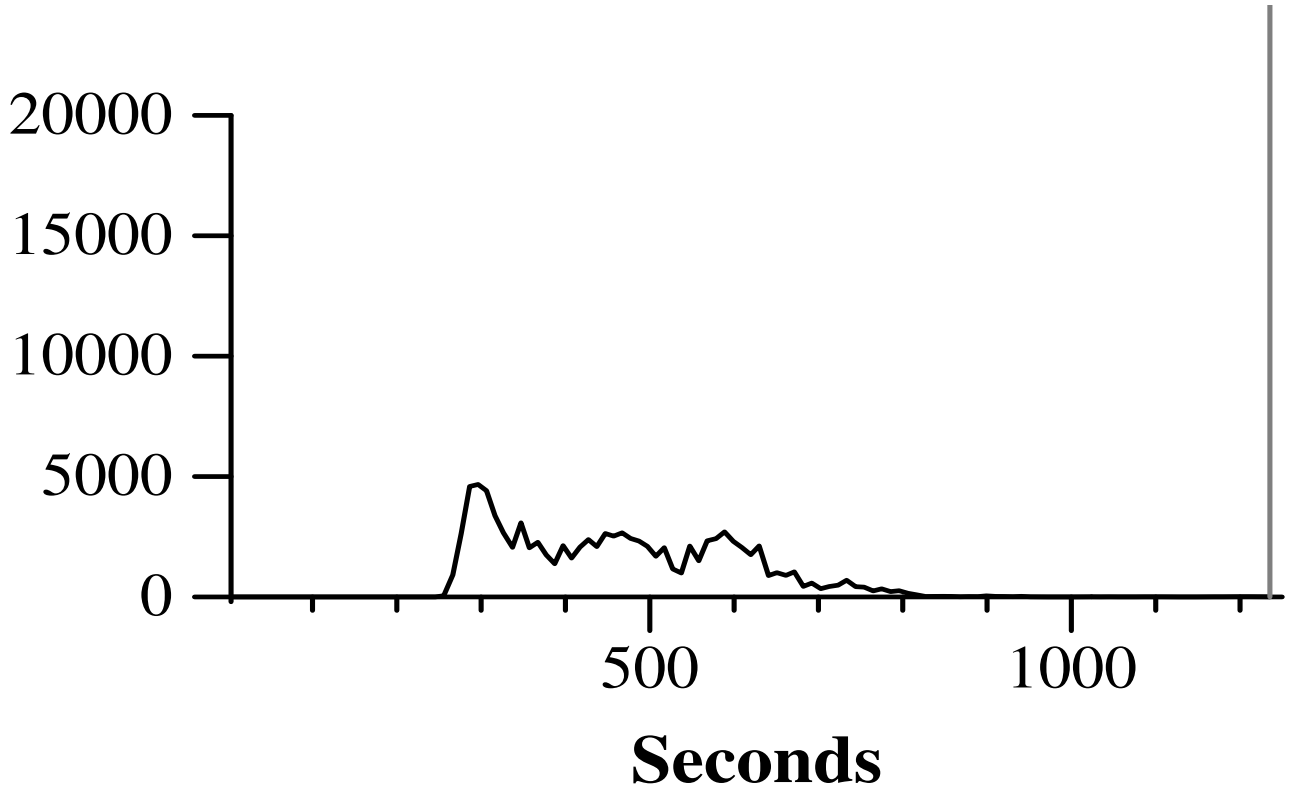
- Worker failure
  - Detect with heartbeat
  - Use backup tasks to reduce “stragglers”
- Some failures caused by inputs
  - Debug and fix?
    - Local Execution
  - Send message to master from signal handler on seg_fault
  - Master skips a record after seeing two failures

Sort

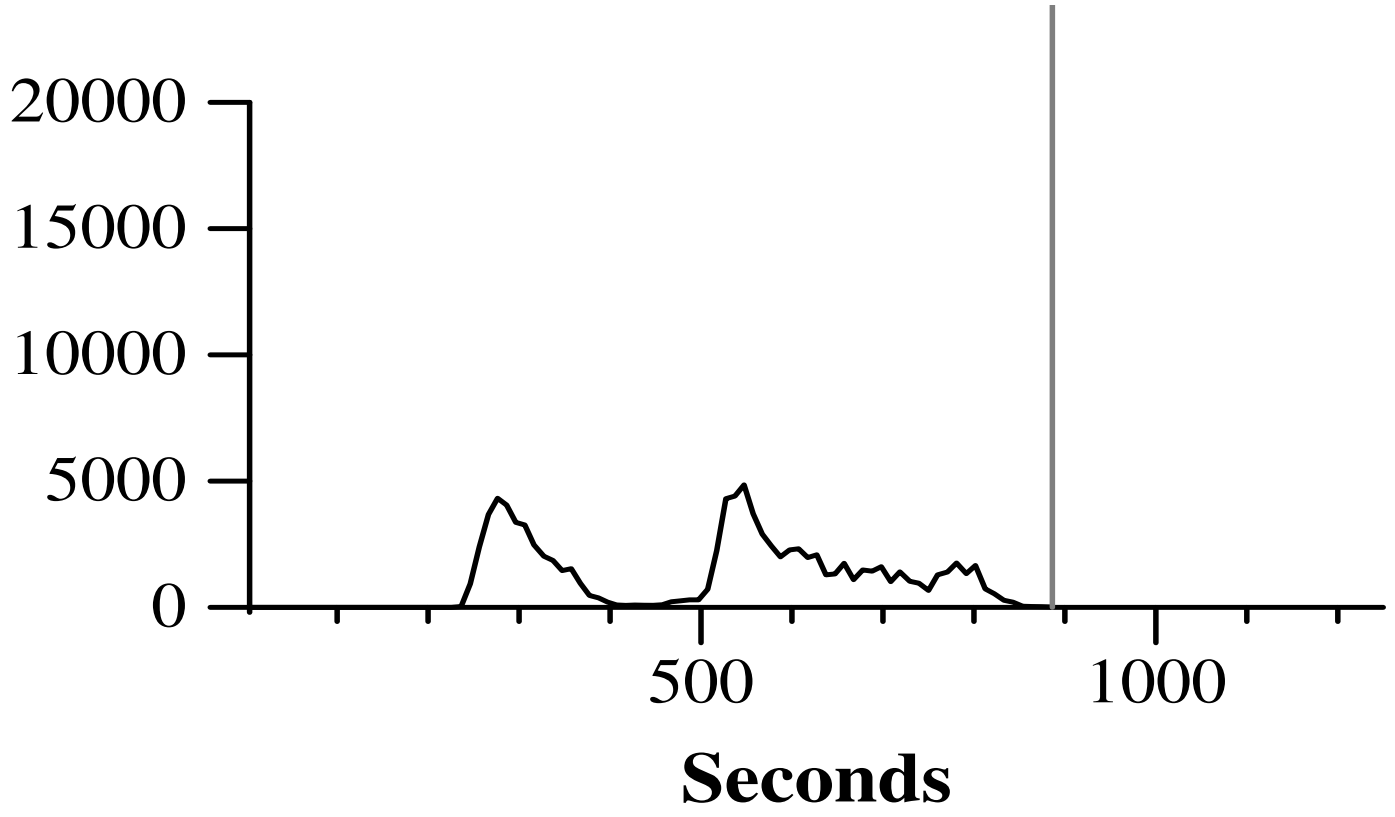
Normal execution	891 seconds
Without backup tasks	1283 seconds
200 tasks killed	933 seconds



(a) Normal execution



(b) No backup tasks



(c) 200 tasks killed

Comments/Questions?

Discussion

- What does MapReduce provide that is novel?
  - Some benefits of MapReduce are not new
- Master failure (use checkpoints)

  "our current implementation aborts the MapReduce computation if the master fails."
  - Is there a better way to handle master failure?
  - When would checkpoints be useful?
- What are some other types of problems that we could solve using MapReduce?
- What are some limitations of MapReduce?
  - MapReduce vs. DBMS