

CSE6339 WEB SEARCH, MINING, AND INTEGRATION

Lecture 16: MapReduce

CSE6339 Web Search, Mining, and Integration, Spring 2009
Department of Computer Science and Engineering, University of Texas at Arlington
Chengkai Li (Slides courtesy of Jeff Dean, Sanjay Ghemawat, and Anand Rajaraman)

Single-node architecture

Machine Learning, Statistics

"Classical" Data Mining

2

Commodity Clusters

- Web data sets can be very large
 - ▣ Tens to hundreds of terabytes
- Cannot mine on a single server (why?)
- Standard architecture emerging:
 - ▣ Cluster of commodity Linux nodes
 - ▣ Gigabit ethernet interconnect
- How to organize computations on this architecture?
 - ▣ Mask issues such as hardware failure

3

Cluster Architecture

2-10 Gbps backbone between racks

1 Gbps between any pair of nodes in a rack

Each rack contains 16-64 nodes

4

Warm up: Word Count

- We have a large file of words, one word to a line
- Count the number of times each distinct word appears in the file
- Sample application: analyze web server logs to find popular URLs

5

Word Count (2)

- Case 1: Entire file fits in memory
- Case 2: File too large for mem, but all <word, count> pairs fit in mem
- Case 3: File on disk, too many distinct words to fit in memory
 - ▣ `sort datafile | uniq -c`

6

Programming model

- Input & Output: each a set of key/value pairs
- Programmer specifies two functions:


```
map (in_key, in_value) -> list(out_key, intermediate_value)
```

 - ▣ Processes input key/value pair
 - ▣ Produces set of intermediate pairs

```
reduce (out_key, list(intermediate_value)) -> list(out_value)
```

 - ▣ Combines all intermediate values for a particular key
 - ▣ Produces a set of merged output values (usually just one)
- Inspired by similar primitives in LISP and other languages

7

MapReduce

- Input: a set of key/value pairs
- User supplies two functions:
 - ▣ $\text{map}(k,v) \rightarrow \text{list}(k1,v1)$
 - ▣ $\text{reduce}(k1, \text{list}(v1)) \rightarrow v2$
- $(k1,v1)$ is an intermediate key/value pair
- Output is the set of $(k1,v2)$ pairs

8

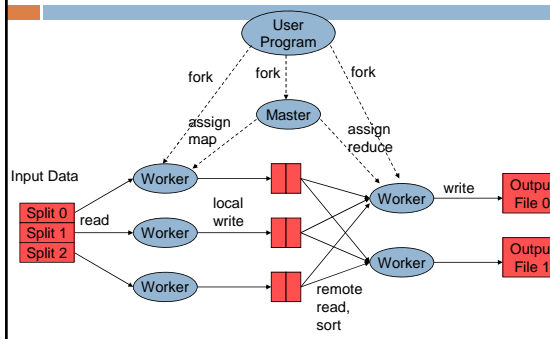
Word Count using MapReduce

```
map(key, value):
// key: document name; value: text of document
for each word w in value:
    emit(w, 1)
```

```
reduce(key, values):
// key: a word; value: an iterator over counts
result = 0
for each count v in values:
    result += v
emit(result)
```

9

Distributed Execution Overview



10

Data flow

- Input, final output are stored on a distributed file system
 - ▣ Scheduler tries to schedule map tasks “close” to physical storage location of input data
- Intermediate results are stored on local FS of map and reduce workers
- Output is often input to another map reduce task

11

Coordination

- Master data structures
 - ▣ Task status: (idle, in-progress, completed)
 - ▣ Idle tasks get scheduled as workers become available
 - ▣ When a map task completes, it sends the master the location and sizes of its R intermediate files, one for each reducer
 - ▣ Master pushes this info to reducers
- Master pings workers periodically to detect failures

12

Failures

- **Map worker failure**
 - Map tasks completed or in-progress at worker are reset to idle
 - Reduce workers are notified when task is rescheduled on another worker
- **Reduce worker failure**
 - Only in-progress tasks are reset to idle
- **Master failure**
 - MapReduce task is aborted and client is notified

13

Combiners

- Often a map task will produce many pairs of the form $(k, v_1), (k, v_2), \dots$ for the same key k
 - E.g., popular words in Word Count
- Can save network time by pre-aggregating at mapper
 - $\text{combine}(k1, \text{list}(v1)) \rightarrow v2$
 - Usually same as reduce function
- Works only if reduce function is commutative and associative

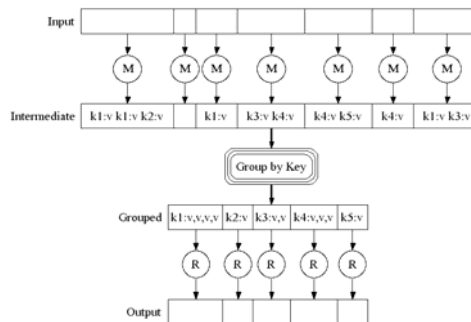
14

Partition Function

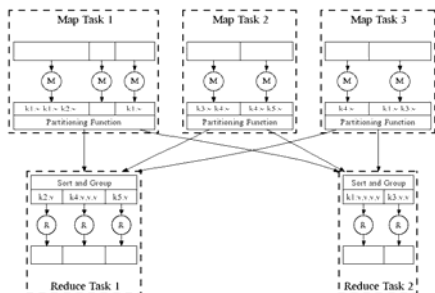
- Inputs to map tasks are created by contiguous splits of input file
- For reduce, we need to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function e.g., $\text{hash}(\text{key}) \bmod R$
- Sometimes useful to override
 - E.g., $\text{hash}(\text{hostname}(\text{URL})) \bmod R$ ensures URLs from a host end up in the same output file

15

Execution

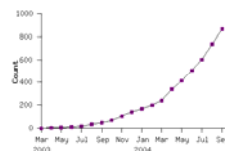


Parallel Execution



Model is Widely Applicable

- MapReduce Programs In Google Source Tree



Example uses:

- | | | |
|---------------------|----------------------|---------------------------------|
| distributed grep | distributed sort | web link-graph reversal |
| term-vector / host | web access log stats | inverted index construction |
| document clustering | machine learning | statistical machine translation |
| ... | ... | ... |

18

Exercise 1: Host size

- Suppose we have a large web corpus
- Let's look at the metadata file
 - ▣ Lines of the form (URL, size, date, ...)
- For each host, find the total number of bytes
 - ▣ i.e., the sum of the page sizes for all URLs from that host

19

Exercise 2: Distributed Grep

- Find all occurrences of the given pattern in a very large set of files

20

Exercise 3: Graph reversal

- Given a directed graph as an adjacency list:
src1: dest11, dest12, ...
src2: dest21, dest22, ...
- Construct the graph in which all the links are reversed

21

Exercise 4: Frequent Pairs

- Given a large set of market baskets, find all frequent pairs
 - ▣ Remember definitions from Association Rules lectures

22