CS 744: BIG DATA SYSTEMS

Shivaram Venkataraman
Fall 2018
MOTIVATION

Build Google Web Search!
- Crawl documents, build inverted indexes etc.

Need for
- automatic parallelization
- network, disk optimization
- handling of machine failures
OUTLINE

- Programming Model
- Execution Overview
- Fault Tolerance
- Optimizations
PROGRAMMING MODEL

Data type: Each record is (key, value)

**Map** function:

\[(K_{in}, V_{in}) \rightarrow \text{list}(K_{inter}, V_{inter})\]

**Reduce** function:

\[(K_{inter}, \text{list}(V_{inter})) \rightarrow \text{list}(K_{out}, V_{out})\]
def mapper(line):
    for word in line.split():
        output(word, 1)

def reducer(key, values):
    output(key, sum(values))
WORD COUNT EXECUTION

Input

the quick brown fox
the fox ate the mouse
how now brown cow

Map

the, 1
brown, 1
fox, 1
quick, 1

Map

the, 1
fox, 1

Map

how, 1
now, 1
brown, 1
cow, 1

Shuffle & Sort

the, 1
brown, 1
fox, 1
quick, 1

the, 1
fox, 1

the, 1

ate, 1
mouse, 1

how, 1
now, 1

Reduce

brown, 2
fox, 2
how, 1
now, 1
the, 3

Reduce

ate, 1
cow, 1
mouse, 1
quick, 1

Output
WORD COUNT EXECUTION

Submit a Job → JobTracker

Automatically split work

Map

the quick brown fox

Map

the fox ate the mouse

Map

how now brown cow

Schedule tasks with locality
FAULT RECOVERY

If a task crashes:

– Retry on another node
– If the same task repeatedly fails, end the job
If a task crashes:

- Retry on another node
- If the same task repeatedly fails, end the job

Requires user code to be deterministic
If a node crashes:
   – Relaunch its current tasks on other nodes
What about task inputs? File system replication

Fault Recovery
If a task is going slowly (straggler):
- Launch second copy of task on another node
- Take the output of whichever finishes first
REFINEMENTS

- Combiner functions
- Counters
- Side effects
### MapReduce Usage Statistics Over Time

<table>
<thead>
<tr>
<th></th>
<th>Aug, '04</th>
<th>Mar, '06</th>
<th>Sep, '07</th>
<th>Sep, '09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of jobs</td>
<td>29K</td>
<td>171K</td>
<td>2,217K</td>
<td>3,467K</td>
</tr>
<tr>
<td>Average completion time (secs)</td>
<td>634</td>
<td>874</td>
<td>395</td>
<td>475</td>
</tr>
<tr>
<td>Machine years used</td>
<td>217</td>
<td>2,002</td>
<td>11,081</td>
<td>25,562</td>
</tr>
<tr>
<td>Input data read (TB)</td>
<td>3,288</td>
<td>52,254</td>
<td>403,152</td>
<td>544,130</td>
</tr>
<tr>
<td>Intermediate data (TB)</td>
<td>758</td>
<td>6,743</td>
<td>34,774</td>
<td>90,120</td>
</tr>
<tr>
<td>Output data written (TB)</td>
<td>193</td>
<td>2,970</td>
<td>14,018</td>
<td>57,520</td>
</tr>
<tr>
<td>Average worker machines</td>
<td>157</td>
<td>268</td>
<td>394</td>
<td>488</td>
</tr>
</tbody>
</table>
DISCUSSION: PROGRAMMABILITY

Most real applications require multiple MR steps
- Google indexing pipeline: 21 steps
- Analytics queries (e.g. sessions, top K): 2-5 steps
- Iterative algorithms (e.g. PageRank): 10’s of steps

Multi-step jobs create spaghetti code
- 21 MR steps $\rightarrow$ 21 mapper and reducer classes
DISCUSSION: PERFORMANCE

MR only provides one pass of computation
  – Must write out data to file system in-between

Expensive for apps that need to reuse data
  – Multi-step algorithms (e.g. PageRank)
  – Interactive data mining
QUESTIONS / DISCUSSION ?