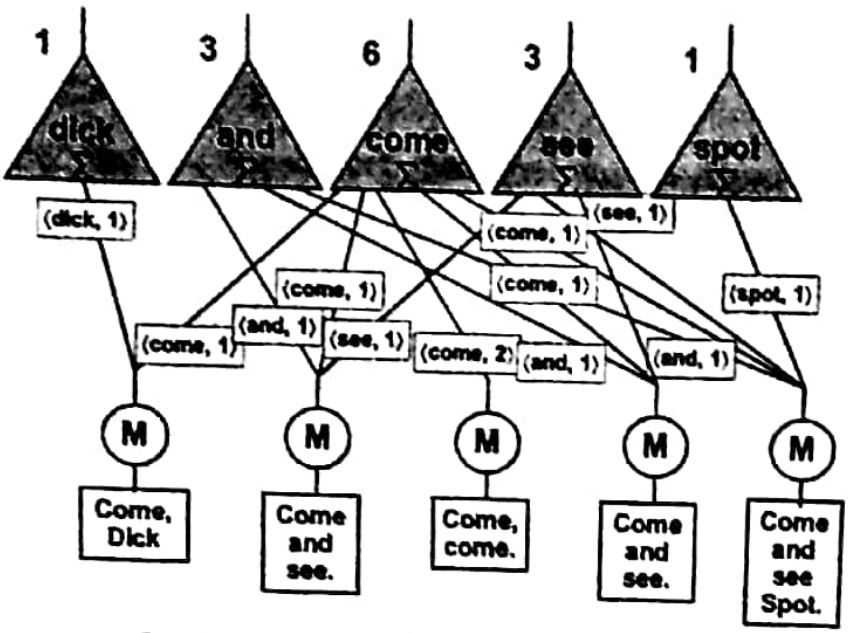
Map/Reduce Example



Word-Count Pairs

Extract

- Create an word index of set of documents
- Map: generate ⟨word, count⟩ pairs for all words in document
- Reduce: sum word counts across documents

Hadoop API

Requirements

- Programmer must supply Mapper & Reducer classes

Mapper

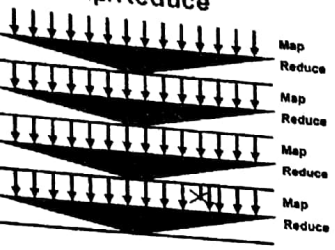
- Steps through file one line at a time
- Code generates sequence of <key, value>
  - Call output.collect(key, value)
- Default types for keys & values are strings
  - Lots of low-level machinery to convert to & from other data types
  - But can use anything "writable"

Reducer

- Given key + iterator that generates sequence of values
- Generate one or more <key, value> pairs
  - Call output.collect(key, value)

...ault Tolerance

Map/Reduce



Data Integrity

- Store multiple copies of each file
- Including intermediate results of each Map / Reduce
  - Continuous checkpointing

Recovering from Failure

- Simply recompute lost result
  - Localized effect
- Dynamic scheduler keeps all processors busy

Example: Sparse Matrices with Map/Reduce

$$A = \begin{bmatrix} 10 & & 20 \\ & 30 & 40 \\ 50 & 60 & 70 \end{bmatrix} \quad X \quad B = \begin{bmatrix} -1 & \\ -2 & -3 \\ & -4 \end{bmatrix} \quad = \quad C = \begin{bmatrix} -10 & -80 \\ -60 & -250 \\ -170 & -460 \end{bmatrix}$$

- Task: Compute product C = A·B
- Assume most matrix entries are 0

Motivation

- Core problem in scientific computing
- Challenging for parallel execution
- Demonstrate expressiveness of Map/Reduce

Computing Sparse Matrix Product

A

$$\begin{bmatrix} 10 & & 20 \\ & 30 & 40 \\ 50 & 60 & 70 \end{bmatrix}$$

Row Col

1 10 → 1 (A)
1 20 → 3 (A)
2 30 → 2 (A)
2 40 → 3 (A)
3 50 → 1 (A)
3 60 → 2 (A)
3 70 → 3 (A)

B

$$\begin{bmatrix} -1 & \\ -2 & -3 \\ & -4 \end{bmatrix}$$

Row Col

1 -1 → 1 (B)
2 -2 → 1 (B)
2 -3 → 2 (B)
3 -4 → 2 (B)

- Represent matrix as list of nonzero entries
  ⟨row, col, value, matrixID⟩
- Strategy
  - Phase 1: Compute all products $a_{i,k} \cdot b_{k,j}$
  - Phase 2: Sum products for each entry i,j
  - Each phase involves a Map/Reduce

Phase 1 Map of Matrix Multiply

Row Col

1 10 → 1 (A)
1 20 → 3 (A)
2 30 → 2 (A)
2 40 → 3 (A)
3 50 → 1 (A)
3 60 → 2 (A)
3 70 → 3 (A)

Key = col

Key = row

1 -1 → 1 (B)
2 -2 → 1 (B)
2 -3 → 2 (B)
3 -4 → 2 (B)

Key = 1
1 10 → 1 (A)
3 50 → 1 (A)
1 -1 → 1 (B)

Key = 2
2 30 → 2 (A)
3 60 → 2 (A)
2 -2 → 1 (B)
2 -3 → 2 (B)

Key = 3
1 20 → 3 (A)
2 40 → 3 (A)
3 70 → 3 (A)
3 -4 → 2 (B)

- Group values $a_{i,k}$ and $b_{k,j}$ according to key k

Phase 1 "Reduce" of Matrix Multiply

Key = 1
1 10 → 1 (A)
3 50 → 1 (A)
X
1 -1 → 1 (B)

1 -10 → 1 (C)
3 -50 → 1 (C)

Key = 2
2 30 → 2 (A)
3 60 → 2 (A)
X
2 -2 → 1 (B)
2 -3 → 2 (B)

2 -60 → 1 (C)
2 -90 → 2 (C)
3 -120 → 1 (C)
3 -180 → 2 (C)

Key = 3
1 20 → 3 (A)
2 40 → 3 (A)
3 70 → 3 (A)
X
3 -4 → 2 (B)

1 -80 → 2 (C)
2 -160 → 2 (C)
3 -280 → 2 (C)

- Generate all products $a_{i,k} \cdot b_{k,j}$

Phase 2 Map of Matrix Multiply

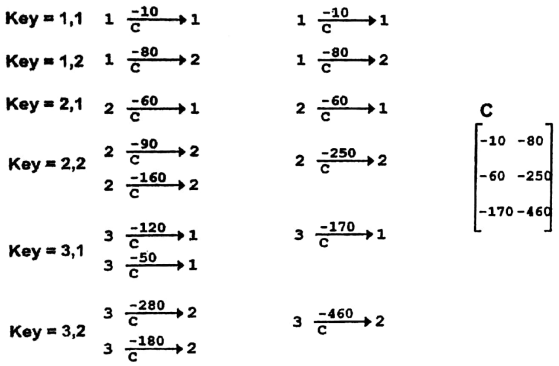
1 -10 → 1 (C)
3 -50 → 1 (A)
2 -60 → 1 (C)
2 -90 → 2 (C)
3 -120 → 1 (C)
3 -180 → 2 (C)
1 -80 → 2 (C)
2 -160 → 2 (C)
3 -280 → 2 (C)

Key = row,col

Key = 1,1 1 -10 → 1 (C)
Key = 1,2 1 -80 → 2 (C)
Key = 2,1 2 -60 → 1 (C)
Key = 2,2 2 -90 → 2 (C)
2 -160 → 2 (C)
Key = 3,1 3 -120 → 1 (C)
3 -50 → 1 (C)
Key = 3,2 3 -280 → 2 (C)
3 -180 → 2 (C)

- Group products $a_{i,k} \cdot b_{k,j}$ with matching values of i and j

Phase 2 Reduce of Matrix Multiply



- Sum products to get final entries

Matrix Multiply Phase 1 Mapper

```
public class P1Mapper extends MapReduceBase implements Mapper {

    public void map(WritableComparable key, Writable values,
                    OutputCollector output, Reporter reporter) throws
IOException {
        try {
            GraphEdge e = new GraphEdge(values.toString());
            IntWritable k;
            if (e.tag.equals("A"))
                k = new IntWritable(e.toNode);
            else
                k = new IntWritable(e.fromNode);
            output.collect(k, new Text(e.toString()));
        } catch (BadGraphException e) {}
    }
}
```

Matrix Multiply Phase 1 Reducer

```
lic class P1Reducer extends MapReduceBase implements Reducer {

    public void reduce(WritableComparable key, Iterator values,
                       OutputCollector output, Reporter reporter)
                       throws IOException
{
    Text outv = new Text(""); // Don't really need output values
    /* First split edges into A and B categories */
    LinkedList<GraphEdge> alist = new LinkedList<GraphEdge>();
        LinkedList<GraphEdge> blist = new LinkedList<GraphEdge>();
        while(values.hasNext()) {
            try {
                GraphEdge e =
                    new GraphEdge(values.next().toString());
                if (e.tag.equals("A")) {
                    alist.add(e);
                } else {
                    blist.add(e);
                }
            } catch (BadGraphException e) {}
        }
    // Continued
```

atrix Multiply Phase 2 Mapper

```
ic class P2Mapper extends MapReduceBase implements Mapper {

public void map(WritableComparable key, Writable values,
                OutputCollector output, Reporter reporter)
                    throws IOException {
    String es = values.toString();
    try {
        GraphEdge e = new GraphEdge(es);
        // Key based on head & tail nodes
        String ks = e.fromNode + " " + e.toNode;
        output.collect(new Text(ks), new Text(e.toString()));
    } catch (BadGraphException e) {}

}
```

MapReduce Implementation

Built on Top of Parallel File System

- Google: GFS, Hadoop: HDFS
- Provides global naming
- Reliability via replication (typically 3 copies)

Breaks work into tasks

- Master schedules tasks on workers dynamically
- Typically #tasks >> #processors

Net Effect

- Input: Set of files in reliable file system
- Output: Set of files in reliable file system
- Can write program as series of MapReduce steps

Reducing

Shuffle

- Each reducer fetches its share of key, value pairs from each mapper using RPC
- Sort data according to keys
  - Use disk-based ("external") sort if too much data for memory

Reduce Operation

- Step through key-value pairs in sorted order
- For each unique key, call reduce function for all values
- Append result to output file

Result

- R output files
- Typically supply to next round of MapReduce

Example Parameters

Sort Benchmark

- 10^{10} 100-byte records
- Partition into M = 15,000 64MB pieces
  - Key = value
  - Partition according to most significant bytes
- Sort locally with R = 4,000 reducers

Machine

- 1800 2Ghz Xeons
- Each with 2 160GB IDE disks
- Gigabit ethernet
- 891 seconds total

Interesting Features

Fault Tolerance

- **Assume reliable file system**
- **Detect failed worker**
  - Heartbeat mechanism
- **Rescheduled failed task**

Stragglers

- **Tasks that take long time to execute**
- **Might be bug, flaky hardware, or poor partitioning**
- **When done with most tasks, reschedule any remaining executing tasks**
  - Keep track of redundant executions
  - Significantly reduces overall run time