15-440

MapReduce Programming
Oct 25, 2011

Topics

- **Large-scale computing**
  - Traditional high-performance computing (HPC)
  - Cluster computing
- **MapReduce**
  - Definition
  - Examples
- **Implementation**
- **Properties**

Example: Sparse Matrices with Map/Reduce

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

- 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

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

$1 \xrightarrow[A]{10} 1$

$1 \xrightarrow[A]{20} 3$

$2 \xrightarrow[A]{30} 2$

$2 \xrightarrow[A]{40} 3$

$3 \xrightarrow[A]{50} 1$

$3 \xrightarrow[A]{60} 2$

$3 \xrightarrow[A]{70} 3$

B

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

$1 \xrightarrow[B]{-1} 1$

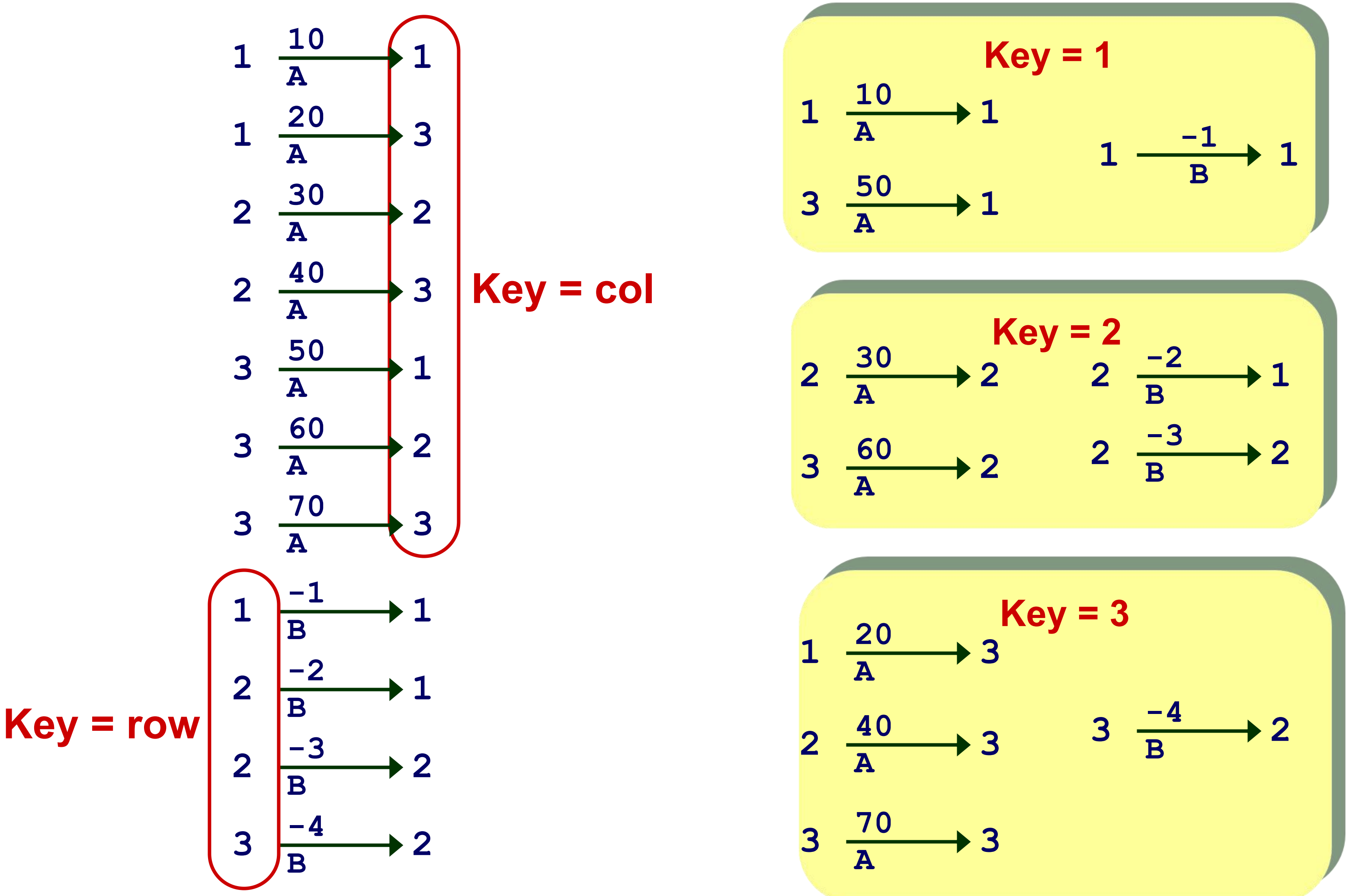
$2 \xrightarrow[B]{-2} 1$

$2 \xrightarrow[B]{-3} 2$

$3 \xrightarrow[B]{-4} 2$

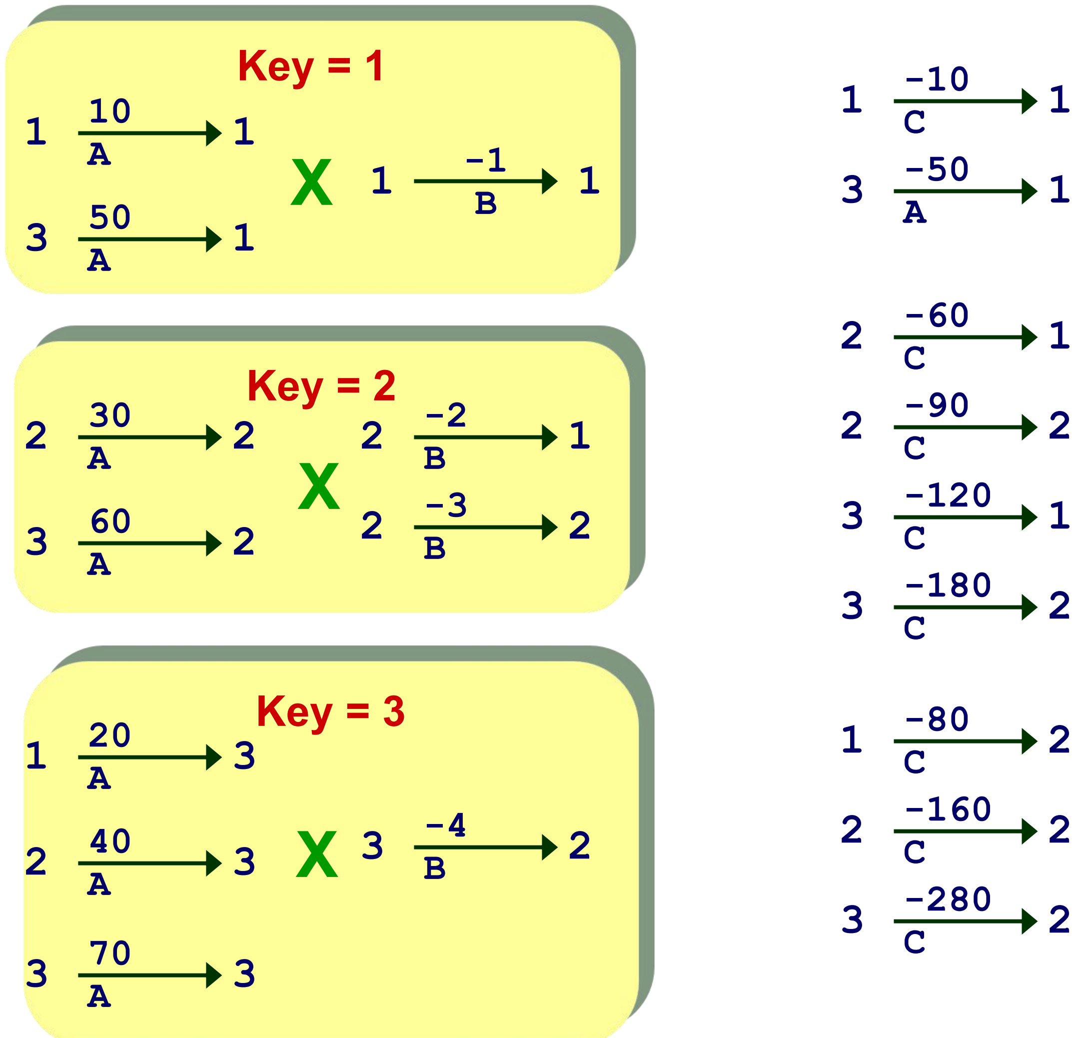
- Represent matrix as list of nonzero entries
  - ⟨row, col, value, matrixID⟩
- Strategy
  - Phase 1: Compute all products $a_{i,k} \cdot b_{k,j}$
  - Phase 2: Sum products for each entry i,j
  - Each phase involves a Map/Reduce

Phase 1 Map of Matrix Multiply



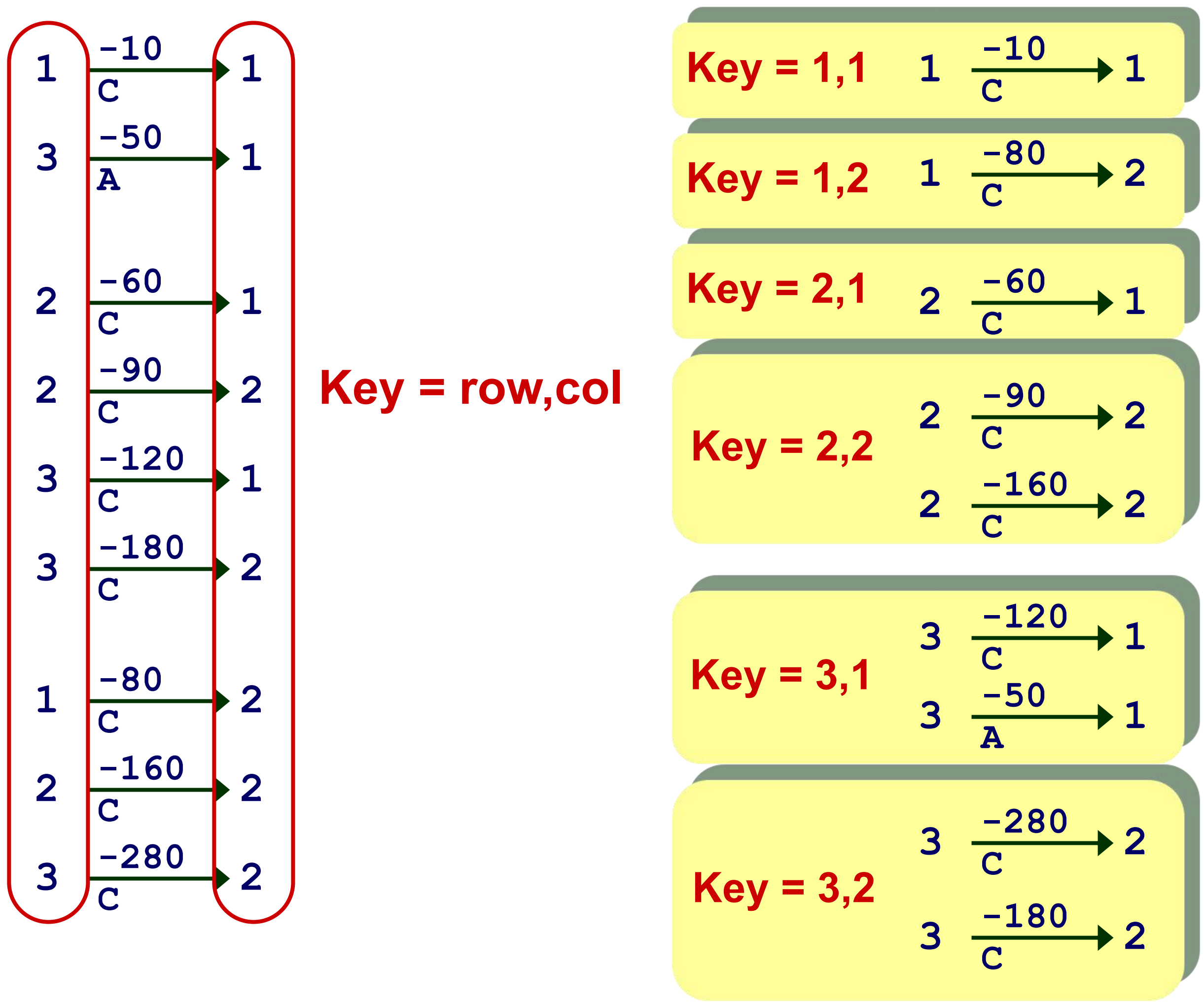
- Group values $a_{i,k}$ and $b_{k,j}$ according to key k

Phase 1 “Reduce” of Matrix Multiply



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

Phase 2 Map of Matrix Multiply



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

Phase 2 Reduce of Matrix Multiply

Key = 1,1	$1 \xrightarrow[C]{-10} 1$	$1 \xrightarrow[C]{-10} 1$
Key = 1,2	$1 \xrightarrow[C]{-80} 2$	$1 \xrightarrow[C]{-80} 2$
Key = 2,1	$2 \xrightarrow[C]{-60} 1$	$2 \xrightarrow[C]{-60} 1$
Key = 2,2	$2 \xrightarrow[C]{-90} 2$ $2 \xrightarrow[C]{-160} 2$	$2 \xrightarrow[C]{-250} 2$
Key = 3,1	$3 \xrightarrow[C]{-120} 1$ $3 \xrightarrow[A]{-50} 1$	$3 \xrightarrow[C]{-170} 1$
Key = 3,2	$3 \xrightarrow[C]{-280} 2$ $3 \xrightarrow[C]{-180} 2$	$3 \xrightarrow[C]{-460} 2$

C

$$\begin{bmatrix} -10 & -80 \\ -60 & -250 \\ -170 & -460 \end{bmatrix}$$

- 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

```
public 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
```

MM Phase 1 Reducer (cont.)

```
            // Continuation

            Iterator<GraphEdge> aset = alist.iterator();
            // For each incoming edge
            while(aset.hasNext()) {
                GraphEdge aedge = aset.next();
                // For each outgoing edge
                Iterator<GraphEdge> bset = blist.iterator();
                while (bset.hasNext()) {
                    GraphEdge bedge = bset.next();
                    GraphEdge newe = aedge.contractProd(bedge);
                    // Null would indicate invalid contraction
                    if (newe != null) {
                        Text outk = new Text(newe.toString());
                        output.collect(outk, outv);
                    }
                }
            }
    }
}
```

Matrix Multiply Phase 2 Mapper

```
public 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) {}

    }
}
```

Matrix Multiply Phase 2 Reducer

```
public class P2Reducer extends MapReduceBase implements Reducer {

        public void reduce(WritableComparable key, Iterator values,
                           OutputCollector output, Reporter reporter)
                               throws IOException
    {
        GraphEdge efinal = null;
        while (efinal == null && values.hasNext()) {
            try {
                efinal = new GraphEdge(values.next().toString());
            } catch (BadGraphException e) {}
        }
        if (efinal != null) {
            while(values.hasNext()) {
                try {
                    GraphEdge eother =
                        new GraphEdge(values.next().toString());
                    efinal.weight += eother.weight;
                } catch (BadGraphException e) {}
            }
            if (efinal.weight != 0)
                output.collect(new Text(efinal.toString()),
                        new Text(""));
        }
    }
}
```

Lessons from Sparse Matrix Example

Associative Matching is Powerful Communication Primitive

- Intermediate step in Map/Reduce

Similar Strategy Applies to Other Problems

- Shortest path in graph
- Database join

Many Performance Considerations

- Kiefer, Volk, Lehner, TU Dresden
- Should do systematic comparison to other sparse matrix implementations

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

Mapping

Parameters

- M: Number of mappers
  - Each gets ~1/M of the input data
- R: Number of reducers
  - Each reducer i gets keys k such that hash(k) = i

Tasks

- Split input files into M pieces, 16—64 MB each
- Scheduler dynamically assigns worker for each “split”

Task operation

- Parse “split”
- Generate key, value pairs & write R different local disk files
  - Based on hash of keys
- Notify master of worker of output file locations

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