CS639: Data Management for Data Science

Lecture 10: Algorithms in MapReduce (continued)

Theodorus Rekatsinas
Logistics/Announcements

- PA3 out by the end of day
- If you have grading questions or questions for PA1 and PA2 please ask Frank and Huawei
- No class on Monday, we will resume on Wednesday
Today’s Lecture

1. Recap on MapReduce data and programming model

2. More MapReduce Examples
1. Recap on MapReduce data and programming model
Recall: The Map Reduce Abstraction for Distributed Algorithms
Recall: MapReduce’s Data Model

• Files!

• A File is a bag of \((\text{key}, \text{value})\) pairs
  • A bag is a multiset

• A map-reduce program:
  • Input: a bag of \((\text{inputkey}, \text{value})\) pairs
  • Output: a bag of \((\text{outputkey}, \text{value})\) pairs
MapReduce 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)) -> (out_key, list(out_values))
  Combines all intermediate values for a particular key
  Produces a set of merged output values (usually just one)
Example: Word count over a corpus of documents

map(String input_key, String input_value):
   //input_key: document id
   //input_value: document bag of words
   for each word w in input_value:
       EmitIntermediate(w, 1);

reduce(String intermediate_key, Iterator intermediate_values):
   //intermediate_key: word
   //intermediate_values: ????
   result = 0;
   for each v in intermediate_values:
       result += v;
   EmitFinal(intermediate_key, result);
2. More MapReduce Examples
### Relational Join

#### Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

#### Assigned Departments

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>Accounts</td>
</tr>
<tr>
<td>7777777777</td>
<td>Sales</td>
</tr>
<tr>
<td>7777777777</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

#### Employee $\bowtie_{SSN=EmpSSN}$ Assigned Departments

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>9999999999</td>
<td>9999999999</td>
<td>Accounts</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Sales</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Marketing</td>
</tr>
</tbody>
</table>
Relational Join

### Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

### Assigned Departments

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>Accounts</td>
</tr>
<tr>
<td>7777777777</td>
<td>Sales</td>
</tr>
<tr>
<td>7777777777</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

### Join Operation

Join operation with \( \bowtie_{\text{SSN}=\text{EmpSSN}} \) between Employee and Assigned Departments.

### Code Snippet

```python
join_result = []
for e in Employee:
    for d in Assigned Departments:
        if e.SSN == d.EmpSSN:
            r = (<e.Name, e.SSN, d.EmpSSN, d.DepName>)
            join_result.append(r)
rerun join_result
```

Remember the semantics!
### Relational Join

#### Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

#### Assigned Departments

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>Accounts</td>
</tr>
<tr>
<td>7777777777</td>
<td>Sales</td>
</tr>
<tr>
<td>7777777777</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

#### Employee ∟_{SSN=EmpSSN} Assigned Departments

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>9999999999</td>
<td>9999999999</td>
<td>Accounts</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Sales</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Marketing</td>
</tr>
</tbody>
</table>
Relational Join

Remember the semantics!

```python
join_result = []
for e in Employee:
    for d in Assigned Departments:
        if e.SSN == d.EmpSSN:
            r = (e.Name, e.SSN, d.EmpSSN, d.DepName)
            join_result.append(r)
rerun join_result
```

Imagine we have a huge number of records!
Let’s use MapReduce!
We want the map phase to process each tuple.

**Is there a problem?**

The Relational Join is a binary operation!
But MapReduce is a unary operation:
I operate on a single key

*Can we approximate the join using MapReduce?*
Relational Join in MapReduce: Preprocessing before the Map Phase

Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

Assigned Departments

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>Accounts</td>
</tr>
<tr>
<td>7777777777</td>
<td>Sales</td>
</tr>
<tr>
<td>7777777777</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

Key idea: Flatten all tables and combine tuples from different tables in a single dataset

<table>
<thead>
<tr>
<th>Employee</th>
<th>SSN</th>
<th>Assigned Departments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>9999999999</td>
<td>Accounts</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>Sales</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Marketing</td>
</tr>
</tbody>
</table>
Relational Join in MapReduce: Preprocessing before the Map Phase

**Employee**

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

**Assigned Departments**

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>Accounts</td>
</tr>
<tr>
<td>7777777777</td>
<td>Sales</td>
</tr>
<tr>
<td>7777777777</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

**Key idea:** Flatten all tables and combine tuples from different tables in a single dataset

<table>
<thead>
<tr>
<th>Employee</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We use the table name to keep track of “which table did the tuple come from”

This is a label that we've attached to every tuple so that we can know where that came from. We'll use it later!
Relational Join in MapReduce: Map Phase

<table>
<thead>
<tr>
<th></th>
<th>Employee</th>
<th>Sue</th>
<th>9999999999</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Employee</td>
<td>Tony</td>
<td>7777777777</td>
</tr>
<tr>
<td>Assigned Departments</td>
<td>9999999999</td>
<td>Accounts</td>
<td></td>
</tr>
<tr>
<td>Assigned Departments</td>
<td>7777777777</td>
<td>Sales</td>
<td></td>
</tr>
<tr>
<td>Assigned Departments</td>
<td>7777777777</td>
<td>Marketing</td>
<td></td>
</tr>
</tbody>
</table>

For each tuple in the flattened input we will generate a key value pair!

key=9999999999, value=(Employee, Sue, 9999999999)
key=7777777777, value=(Employee, Tony, 7777777777)
key=9999999999, value=(Assigned Departments, 9999999999, Accounts)
key=7777777777, value=(Assigned Departments, 7777777777, Sales)
key=7777777777, value=(Assigned Departments, 7777777777, Marketing)
Relational Join in MapReduce: Map Phase

<table>
<thead>
<tr>
<th>Employee</th>
<th>Sue</th>
<th>9999999999</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>Tony</td>
<td>7777777777</td>
</tr>
<tr>
<td>Assigned Departments</td>
<td>9999999999</td>
<td>Accounts</td>
</tr>
<tr>
<td>Assigned Departments</td>
<td>7777777777</td>
<td>Sales</td>
</tr>
<tr>
<td>Assigned Departments</td>
<td>7777777777</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

Why use this value as the key?

For each tuple in the flattened input we will generate a key value pair!

- **key**=9999999999, **value**=(Employee, Sue, 9999999999)
- **key**=7777777777, **value**=(Employee, Tony, 7777777777)
- **key**=9999999999, **value**=(Assigned Departments, 9999999999, Accounts)
- **key**=7777777777, **value**=(Assigned Departments, 7777777777, Sales)
- **key**=7777777777, **value**=(Assigned Departments, 7777777777, Marketing)
Relational Join in MapReduce: Map Phase

<table>
<thead>
<tr>
<th>Employee</th>
<th>Sue</th>
<th>9999999999</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>Tony</td>
<td>7777777777</td>
</tr>
<tr>
<td>Assigned Departments</td>
<td>9999999999</td>
<td>Accounts</td>
</tr>
<tr>
<td>Assigned Departments</td>
<td>7777777777</td>
<td>Sales</td>
</tr>
<tr>
<td>Assigned Departments</td>
<td>7777777777</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

Why use this value as the key?
We are joining on SSN (for Employee) and EmpSSN (for Assigned Depts)

For each tuple in the flattened input we will generate a key value pair!

key=9999999999, value=(Employee, Sue, 9999999999)
key=7777777777, value=(Employee, Tony, 7777777777)
key=9999999999, value=(Assigned Departments, 9999999999, Accounts)
key=7777777777, value=(Assigned Departments, 7777777777, Sales)
key=7777777777, value=(Assigned Departments, 7777777777, Marketing)
Relational Join in MapReduce: Map Phase *(Two Tricks so far)*

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Employee</strong></td>
<td>Sue</td>
<td>9999999999</td>
</tr>
<tr>
<td><strong>Employee</strong></td>
<td>Tony</td>
<td>7777777777</td>
</tr>
<tr>
<td><strong>Assigned</strong></td>
<td>9999999999</td>
<td>Accounts</td>
</tr>
<tr>
<td><strong>Departments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Assigned</strong></td>
<td>7777777777</td>
<td>Sales</td>
</tr>
<tr>
<td><strong>Departments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Assigned</strong></td>
<td>7777777777</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

**Why use this value as the key?**

*We are joining on SSN (for Employee) and EmpSSN (for Assigned Depts)*

- key=9999999999, value=(Employee, Sue, 9999999999)
- key=7777777777, value=(Employee, Tony, 7777777777)
- key=9999999999, value=(Assigned Departments, 9999999999, Accounts)
- key=7777777777, value=(Assigned Departments, 7777777777, Sales)
- key=7777777777, value=(Assigned Departments, 7777777777, Marketing)

**Trick 1:** Flattened and combined tables in a single input file.

**For each tuple in the flattened input we will generate a key value pair!**

**Trick 2:** Produce a key value pair where the key is the join attribute.
Relational Join in MapReduce: Reduce Phase (after the magic Shuffle)

Input to Reducer 1
key=9999999999, value=[(Employee, Sue, 9999999999),
(Assigned Departments, 9999999999, Accounts)]

Input to Reducer 2
key=7777777777, value=[(Employee, Tony, 7777777777),
(Assigned Departments, 7777777777, Sales),
(Assigned Departments, 7777777777, Marketing)]

After the shuffle phase all inputs with the same key will end up in the same reducer!

It does not matter which relation the different tuples came from!
Relational Join in MapReduce: Reduce Phase (after the magic Shuffle)

Input to Reducer 1
key=9999999999, value=[(Employee, Sue, 9999999999),
(Assigned Departments, 9999999999, Accounts)]

Input to Reducer 2
key=7777777777, value=[(Employee, Tony, 7777777777),
(Assigned Departments, 7777777777, Sales),
(Assigned Departments, 7777777777, Marketing)]

We have all the information we need to perform the join for each key in a single machine.

This is how we scale.
Relational Join in MapReduce: Reduce Phase (after the magic Shuffle)

### Input to Reducer 1
- **key**=9999999999, **value**=
  - (Employee, Sue, 9999999999),
  - (Assigned Departments, 9999999999, Accounts)

### Input to Reducer 2
- **key**=7777777777, **value**=
  - (Employee, Tony, 7777777777),
  - (Assigned Departments, 7777777777, Sales),
  - (Assigned Departments, 7777777777, Marketing)

### Desired output of reduce function

### Output of Reduce Function (Reducer 1)
- Sue, 9999999999, 9999999999, Accounts

### Output of Reduce Function (Reducer 2)
- Tony, 7777777777, 7777777777, Sales
- Tony, 7777777777, 7777777777, Marketing
Relational Join in MapReduce: Reduce Phase (after the magic Shuffle)

Input to Reducer 1
key=9999999999, value=[[Employee, Sue, 9999999999),
(Assigned Departments, 9999999999, Accounts)]

Input to Reducer 2
key=7777777777, value=[[Employee, Tony, 7777777777),
(Assigned Departments, 7777777777, Sales),
(Assigned Departments, 7777777777, Marketing)]

Desired output of reduce function
Sue, 9999999999, 9999999999, Accounts

Output of Reduce Function (Reducer 1)
Sue, 9999999999, 9999999999, Accounts

Output of Reduce Function (Reducer 2)
Tony, 7777777777, 7777777777, Sales
Tony, 7777777777, 7777777777, Marketing

This part came from the Employees table
This part came from the Assigned Departments table

What is the reduce function implementation?
Relational Join in MapReduce: Reduce Phase Implementation

**Input to Reducer 1**
key = 9999999999, value = [(Employee, Sue, 9999999999),
(Assigned Departments, 9999999999, Accounts)]

**Input to Reducer 2**
key = 7777777777, value = [(Employee, Tony, 7777777777),
(Assigned Departments, 7777777777, Sales),
(Assigned Departments, 7777777777, Marketing)]

**Desired output of reduce function**

**Output of Reduce Function (Reducer 1)**
Sue, 9999999999, 9999999999, Accounts

**Output of Reduce Function (Reducer 2)**
Tony, 7777777777, 7777777777, Sales
Tony, 7777777777, 7777777777, Marketing

**Simple Pseudo-code for Reduce**
reduce(String key, Iterator tuples):
    // intermediate_key: join key
    // intermediate_values: tuples with the same join key
    join_result = [];
    for t1 in tuples:
        for t2 in tuples:
            if t1[0] <> t2[0]:
                output tuple = (t1[1:], t2[1:])
                join_result.append(t)
    rerun (key, join_result)

This is a cross-product operation!
Relational algebra is everywhere!
Notice that we need to keep track of where each tuple came from.
# Social Network Analysis: Count Friends

**Input**

<table>
<thead>
<tr>
<th>Jim</th>
<th>Sue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>Jim</td>
</tr>
<tr>
<td>Lin</td>
<td>Joe</td>
</tr>
<tr>
<td>Joe</td>
<td>Lin</td>
</tr>
<tr>
<td>Jim</td>
<td>Kai</td>
</tr>
<tr>
<td>Kai</td>
<td>Jim</td>
</tr>
<tr>
<td>Jim</td>
<td>Lin</td>
</tr>
<tr>
<td>Lin</td>
<td>Jim</td>
</tr>
</tbody>
</table>

**Symmetric friendship edges**

**Desired Output**

- Jim, 3
- Lin, 2
- Sue, 1
- Kai, 1
- Joe, 1
Social Network Analysis: Count Friends

<table>
<thead>
<tr>
<th>Input</th>
<th>Desired Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jim</td>
<td>(1, 1, 1)</td>
</tr>
<tr>
<td>Lin</td>
<td>(1, 1)</td>
</tr>
<tr>
<td>Sue</td>
<td>1</td>
</tr>
<tr>
<td>Joe</td>
<td>1</td>
</tr>
<tr>
<td>Kai</td>
<td>1</td>
</tr>
<tr>
<td>Jim</td>
<td>1</td>
</tr>
<tr>
<td>Lin</td>
<td>1</td>
</tr>
</tbody>
</table>

**Symmetric friendship edges**

- Emit one for each left-hand value

**Key, value**

- Jim, 1
- Sue, 1
- Lin, 1
- Joe, 1
- Kai, 1
- Jim, 1
- Lin, 1

**Map**

- Jim, Sue
- Sue, Jim
- Lin, Joe
- Joe, Lin
- Jim, Kai
- Kai, Jim
- Jim, Lin
- Lin, Jim

**Shuffle**

- Jim, (1, 1, 1)
- Sue, 1
- Lin, (1,1)
- Joe, 1
- Kai, 1

**Reduce**

- Jim, 3
- Lin, 2
- Sue, 1
- Kai, 1
- Joe, 1
Matrix Multiply in MapReduce

- $C = A \times B$
- $A$ dimensions $m \times n$, $B$ dimensions $n \times l$
- In the map phase:
  - for each element $(i,j)$ of $A$, emit $((i,,k),A[i,j])$ for $k$ in $1..l$
    - Key = $(i,k)$ and value = $A[i,j]$
  - for each element $(i,j)$ of $B$, emit $((i,k),B[i,j])$ for $k$ in $1..m$
    - Key = $(i,k)$ and value = $B[i,j]$
- In the reduce phase, emit
  - key = $(i,k)$
  - Value = $\sum_j (A[i,j] \times B[j,k])$
Matrix Multiply in MapReduce: Illustrated

We have one reducer per output cell
Each reducer computes $\sum_j (A[i,j] \times B[j,k])$

we can attach the same value to multiple keys 1, 1 and 1, 2

send to two locations