Nondeterminism in MapReduce
An Empirical Study on Non-commutative Aggregators in MapReduce Programs

Jiaqi Liu

October 23th 2015
MapReduce Overview

We will focus on reduce function
MapReduce
Things can go wrong

What if...
- mappers complete tasks in random order and
- reducers' output depends on order of input data (non-commutative)

Commutativity of reducer

A reducer is commutative if its output remains the same when its input rows are reordered

A non-commutative reducer can be a source of non-determinism
Commutative reducers

Example
max(1, 2, 3) = max(3, 2, 1) = 3

Commutative operations
- SUM, COUNT, MAX, MIN, AND
- top ten, median
- sort then process
- ...

Reducer that performs operations above are commutative
Many reducers are non-commutative (58% of 507 in paper)

Programmers wrote them unintentionally

5 categories

1. Single Item
2. Index Value Pair
3. Max Row
4. First N
5. Str Concat.
Non-commutative reducers

Single item - example

Example

row = (key, value1, value2)

Process input row in loop - select the first or last row

for each row in input:
  x = row[1] // last row

for each row in input:
  x = row[1]
  break // first row

- Reducer output depends on the value of x
- Input rows come in any order; x will get inconsistent first row and last row
Non-commutative reducers
Single item - how to eliminate

**Functional Dependency (FD)**
Column X functionally determines column Y (X --> Y) iff each X value is associated with **exactly one** Y value.

**FD in MapReduce**
If Key_{reducer} --> some value/column (e.g. row[1]), then we have deterministic results, since all row[1]'s must have the same value.

**Example**
user_ID --> user_gender
### Non-commutative reducers

#### Index-value pair

<table>
<thead>
<tr>
<th>Dictionary with 2 non-key values - $V_{index}$: $V_{value}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input order</strong></td>
</tr>
<tr>
<td>key, 1, a</td>
</tr>
<tr>
<td>key, 2, b</td>
</tr>
<tr>
<td>key, 2, c</td>
</tr>
<tr>
<td>Dictionary = {1: a, 2: b}</td>
</tr>
</tbody>
</table>

| **Input order**                                           |
| key, 1, a                                                 |
| key, 2, c                                                 |
| key, 2, b                                                 |
| Dictionary = {1: a, 2: c}                                 |

### How to ensure determinism

Again, functional dependency

$$(key, V_{index}) \rightarrow V_{value}$$
Non-commutative reducers

Max row

Find max \((V_{\text{max}})\), return other value \((V_{\text{other}})\)

<table>
<thead>
<tr>
<th>key, 1, a</th>
<th>key, 1, a</th>
</tr>
</thead>
<tbody>
<tr>
<td>key, 2, b =&gt; emit (2, c)</td>
<td>key, 2, c =&gt; emit (2, b)</td>
</tr>
<tr>
<td>key, 2, c</td>
<td>key, 2, b</td>
</tr>
</tbody>
</table>

How to ensure determinism (2 methods)

1. Again...functional dependency 
   \[(\text{key}, V_{\text{max}}) \rightarrow V_{\text{other}}\]
2. Emit one with the largest \text{timestamp} \text{(latest event)}
   - In case of timestamp tie, pick any
Non-commutative reducers

First N

Return the first N rows and throw away the rest

Determinism?

- Maybe...if N is large
- Or you can tolerate nondeterminism
Non-commutative reducers

Concatenate strings of a column

Usage

- Object serialization for network communication

Determinism?

Depends on succeeding jobs' logic

- Make a string set? Fine
- Print? No...
Non-commutative reducer can be buggy in MapReduce program

Non-commutativity is not checked, for example in Hadoop

Programmers need to check manually:

1. Recognize non-commutativity pattern
2. Check data properties (FD, etc)
Thank you!