Lecture 18 Addendum: Pig to MapReduce Translation

Samuel Madden

April 25, 2012

This document briefly summarizes how Pig programs are translated into MapReduce jobs. The basic idea is that every operation up to a GROUP/COGROUP (and JOIN, when JOIN is implemented via COGROUP) can be pushed into a map job which is executed in parallel on one data item at a time. Each GROUP/COGROUP/JOIN requires the use of a reduce operation to create grouped tuples with a new nesting by collecting all of the tuples with the same key at a particular node. Hence, a Pig program with \( N \) GROUP operations will generate \( N \) MapReduce jobs.

Consider the following Pig program:

```pig
m = LOAD('movies.txt') USING PigStorage('	') AS (title:chararray, director:chararray, actors {name:chararray})
m2 = FILTER m BY director = 'Bob'
m4 = FOREACH m2 GENERATE title, director, FLATTEN(actors)
m6 = GROUP m4 ON actors::name
```

This would be compiled into a single MapReduce job, where the map phase computes \( m, m2, \) and \( m4 \), and the reduce phase computes \( m6 \).

Psuedocode for the Map phase is:

```java
function map(input): // assume input is a file
    // assume input is a file
    records = load(input, ...)
    for r in records:
        if (r.director = 'Bob'):
            emit r.actors.name, {r.title, r.director, flatten(r.actors)}
            //key is actor name
```

Psuedocode for the Reduce phase is:

```java
function reduce(key,input): // assume reduce is called once per key
    emit (key, input) //emit the grouped record
    // here input is a bag of tuples
    // key is actor name
```

As the paper notes, any Pig statements that appear between the \( i \)th and \( i + 1 \)st GROUP statement can be pushed into either the end of \( i \)th reduce phase, or into the \( i + 1 \)st map phase. The paper says that the implementation (as of 2007) chooses the former, because it avoids writing out intermediate results to HDFS for as long as possible (since results have to be written out when going from the \( i \)th reduce to the \( i + 1 \)st map). The presumption here seems to be that the data will get progressively smaller as computation proceeds.

1 Handling COGROUPS/Joins

The above example doesn’t show how operations that combine multiple data sets (e.g., COGROUP/JOIN) are compiled into Map Reduce jobs.

The paper doesn’t describe how this works clearly, but reading between the lines, it seems that it works almost identically to the above. A single map is used for processing all data sets up to the first (CO)GROUP, and the first reduce does the combination of data sets.
Consider the following Pig program (which also uses the definition m from above):

\[ r = \text{LOAD} (\text{\texttt{\textquotesingle}\texttt{ratings.txt}\textquotesingle}) \text{ AS } (\text{movie:chararray, rating:int}) \]
\[ \text{COGROUP} \ r \ \text{ON} \ \text{movie,} \ m \ \text{ON} \ \text{name} \]

As above, the map phase iterates over splits of both the ratings.txt and movies.txt files; the pseudocode would look as follows:

function map(input):
    records = load(input, ...)
    if input from r:
        for rec in records:
            emit rec.movie, \{RATINGS, rec.movie, rec.rating\} //key is movie title
    if input FROM m:
        for rec in records:
            emit rec.name, \{MOVIES, rec.title, rec.director, rec.actors\} //key is movie title

The reduce phase then receives ratings and movies tuples about the same movie, and combines them together to produce the co-grouped result, e.g.:

function reduce(key,input):
    movies = bag of tuples in input where field 0==MOVIES
    ratings = bag of tuples in input where field 1==RATINGS
    emit (key,movies,ratings)

2 References


The second paper describes the compilation process a bit more clearly than the first paper.