Data-Intensive Distributed Computing
CS 431/631 (Winter 2021)

Part 3: From MapReduce to Spark (1/3)

Ali Abedi

These slides are available at https://www.student.cs.uwaterloo.ca/~cs451/

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States
See http://creativecommons.org/licenses/by-nc-sa/3.0/us/ for details
The datacenter *is* the computer!
What’s the instruction set?
We need a solution for both storage and computing.
So you like programming in assembly?

So when we program in MapReduce is it like programming in assembly?! How can we do better?
What’s the solution?

Design a higher-level language
Write a compiler
Hadoop is great, but it’s really waaaaay too low level!

What we really need is SQL!
Answer:

What we really need is a scripting language!
Answer:

Yahoo and Facebook designed their own solutions on top of Hadoop to make it more flexible for their engineers.
SQL

Aside: Why not just use a database?

Pig Scripts

Both open-source projects today!
Pig and Hive programs are converted to MapReduce jobs at the end of the day.
### Pig: Example

**Task:** Find the top 10 most visited pages in each category

<table>
<thead>
<tr>
<th>Visits</th>
<th>URL Info</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User</strong></td>
<td><strong>Url</strong></td>
</tr>
<tr>
<td>Amy</td>
<td>cnn.com</td>
</tr>
<tr>
<td>Amy</td>
<td>bbc.com</td>
</tr>
<tr>
<td>Amy</td>
<td>flickr.com</td>
</tr>
<tr>
<td>Fred</td>
<td>cnn.com</td>
</tr>
</tbody>
</table>

Pig Slides adapted from Olston et al. (SIGMOD 2008)
Pig: Example Script

visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits generate url, count(visits);
urlInfo = load '/data/urlInfo' as (url, category, pRank);
visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by url, category;
topUrls = foreach gCategories generate top(visitCounts, 10);

store topUrls into '/data/topUrls';
Pig Query Plan

load visits

foreach url
    generate count

group by url

load urlinfo

join on url

foreach category
    generate top(uris, 10)

group by category
Pig: MapReduce Execution

1. **Map** 1: `load visits
   group by url
   foreach url generate count`

2. **Reduce** 1: `join on url
   group by category
   foreach category generate top(urls, 10)`

3. **Map** 2: `load urlinfo`

4. **Reduce** 2: `join on url
   group by category`

5. **Map** 3: `foreach category generate top(urls, 10)`

6. **Reduce** 3: 

Pig Slides adapted from Olston et al. (SIGMOD 2008)
visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits generate url, count(visits);
urlInfo = load '/data/urlInfo' as (url, category, pRank);
visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts, 10);

store topUrls into '/data/topUrls';
But isn’t Pig slower?
Sure, but c can be slower than assembly too...
The datacenter *is* the computer!

What’s the instruction set?
Okay, let’s fix this!

Having to formulate the problem in terms of map and reduce only is restrictive.
There is a lot of disk i/o involved which significantly reduces running MapReduce jobs like this.
It’s okay not to have reduce but the output of map cannot go to another map.
Similarly we cannot directly move the output of reduce to another reduce.
The datacenter is the computer!
Let’s enrich the instruction set!

Can we add more operations to make the instruction set more flexible?
Spark
Answer to “What’s beyond MapReduce?”

Brief history:
Developed at UC Berkeley AMPLab in 2009
Open-sourced in 2010
Became top-level Apache project in February 2014
Spark vs. Hadoop

Spark is more popular than Hadoop today.
This is the only mechanism we had in MapReduce.
Map-like Operations

But Spark provides many more operations (enriched instruction set).
Reduce-like Operations

- groupByKey
  - Input: RDD[(K, V)]
  - Output: RDD[(K, Iterable[V])]`

- reduceByKey
  - f: (V, V) ⇒ V
  - Input: RDD[(K, V)]
  - Output: RDD[(K, V)]`

- aggregateByKey
  - seqOp: (U, V) ⇒ U
  - combOp: (U, U) ⇒ U
  - Input: RDD[(K, V)]
  - Output: RDD[(K, U)]`
And many other operations!