NYU | Center for Data Science

Week 03.2: Criticisms of Map-Reduce

DS-GA 1004: Big Data

5 criticisms

(DeWitt & Stonebraker, 2008)

1. Too low-level
2. Poor implementation
3. Not novel
4. Missing important features
5. No DBMS compatibility

Criticism 1: Too low-level

> - *Schemas are good.*
> - *Separation of the schema from the application is good.*
> - *High-level access languages are good.*
>
> *MapReduce has learned none of these lessons and represents a throw back to the 1960s, before modern DBMSs were invented.*

- Is this fair?
- Is this valid?
- Layers on top of MapReduce can address this, e.g. Apache Pig

Criticism 2: Poor implementation

- MapReduce **does not index** data like an RDBMS
    - Much more like a file system, with keys as filenames

- Indexing should be better… right?
    - For example, analyzing only a given subset of data

- Is this a major failing?

Criticism 3: not novel

- Plenty of previous systems used partitioning and aggregation
- Was MapReduce novel?
- Did Dean & Ghemawat claim that it was?
- Does it matter?

Criticism 4: missing features

- Indexing, transactions, schema, integrity constraints, views, …

  These are all missing from MapReduce!

- Why are they missing? Is this a drawback?
- Is this a fair comparison?

Criticism 5: lack of DBMS compatibility

- Lots of infrastructure has been built on top of standard DBMS for, e.g.,
    - Visualization
    - Data migration
    - Database design

- DW&S was over 10 years ago, things have changed a bit since then
    - Apache Hive, Phoenix + HBase can fill in some of these gaps

Why was map-reduce successful?

- DW&S raise some valid points, so why was MapReduce so successful?

Why was map-reduce successful?

- DW&S raise some valid points, so why was MapReduce so successful?
- Some possible reasons:
    - Simplicity: “**map**” and “**reduce**” are powerful abstractions, and often easy to write
    - Many jobs are single-shot: not worth building elaborate DB infrastructure

Some very real problems with MR

- Latency and scheduling
- Intermediate storage
- Not everything fits nicely in map-reduce
  - Iterative algorithms (e.g., gradient descent) are especially painful
  - Interactive processes (visualization, exploration) are too

What's the role of map-reduce today?

- (**Warning**: opinions!)
- MR is great for large **Ω(*N*)** batch jobs that run infrequently, e.g.:
  - Data transformation / feature extraction
  - Index / data-structure construction
- It's not so great for iterative or interactive jobs:
  - Machine learning (training)
  - Search and retrieval
  - Data exploration

So why do we study map-reduce?

- It’s historically important!
- It’s a useful way of thinking about breaking down problems
- The Hadoop ecosystem is much bigger than map-reduce
- Odds are non-zero that you may inherit some legacy code

Next week

Hadoop distributed file system

- Map-Reduce is only half the story
- MR lets us distribute computation, but how do we distribute data?
- HDFS lets us share data for MR, but also has a life outside MR