Understanding MapReduce with Hadoop

Tom White

The Problem

- Existing tools are struggling to process today's large datasets
- How long to grep 1TB of log files?
- How long to update a 1TB database?
- Why is this a problem for me?

The Solution

- Parallelism
- Transfer, don't seek
- Alternatives:
  - In memory DBs
  - Streaming DBs

MapReduce

- Sort/merge is the primitive
  - Operates at transfer rate
- Batch-oriented
  - Not for online access
- Ad hoc queries
  - No schema
- Distribution handled by the framework
- Simple model: key/value pairs

History of MapReduce and Hadoop

- Feb 2003 – First MapReduce library written at Google
- Dec 2004 – Google paper published
- July 2005 – Doug Cutting reports that Nutch now uses new MapReduce implementation
- Jan 2006 – Doug Cutting joins Yahoo!
- Feb 2006 – Hadoop code moves out of Nutch into new Lucene subproject
- Apr 2007 – Yahoo! running Hadoop on 1000-node cluster
- Jan 2008 – Hadoop made an Apache Top Level Project
- Feb 2008 – Yahoo! generate production search index with Hadoop

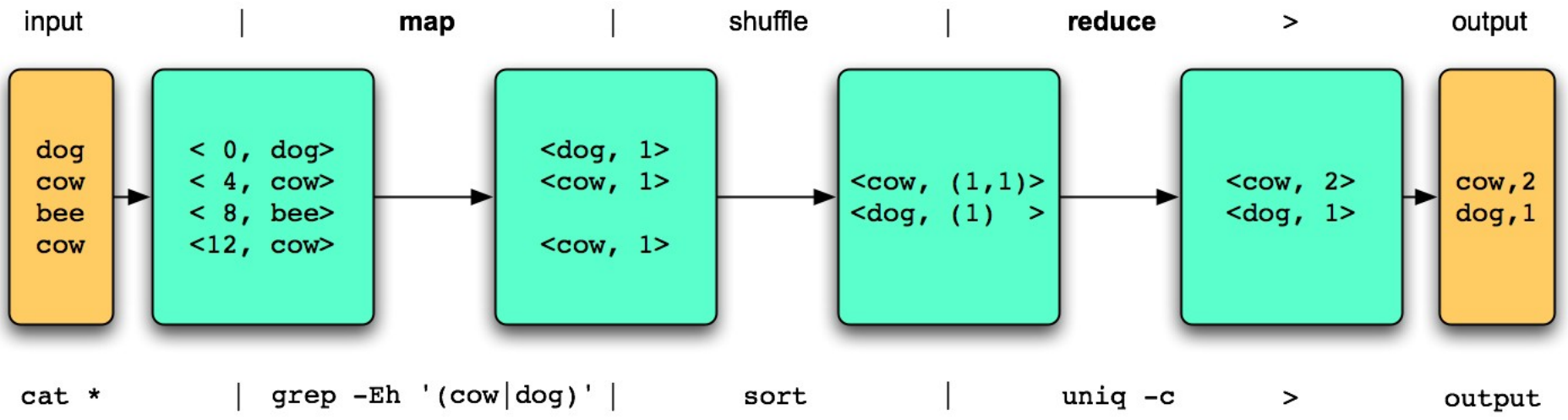
What's in Hadoop?

- Hadoop is more than MapReduce
  - Hadoop Distributed File System
  - MapReduce
  - Pig – high-level language for data analysis
  - HBase – storage for semi-structured data

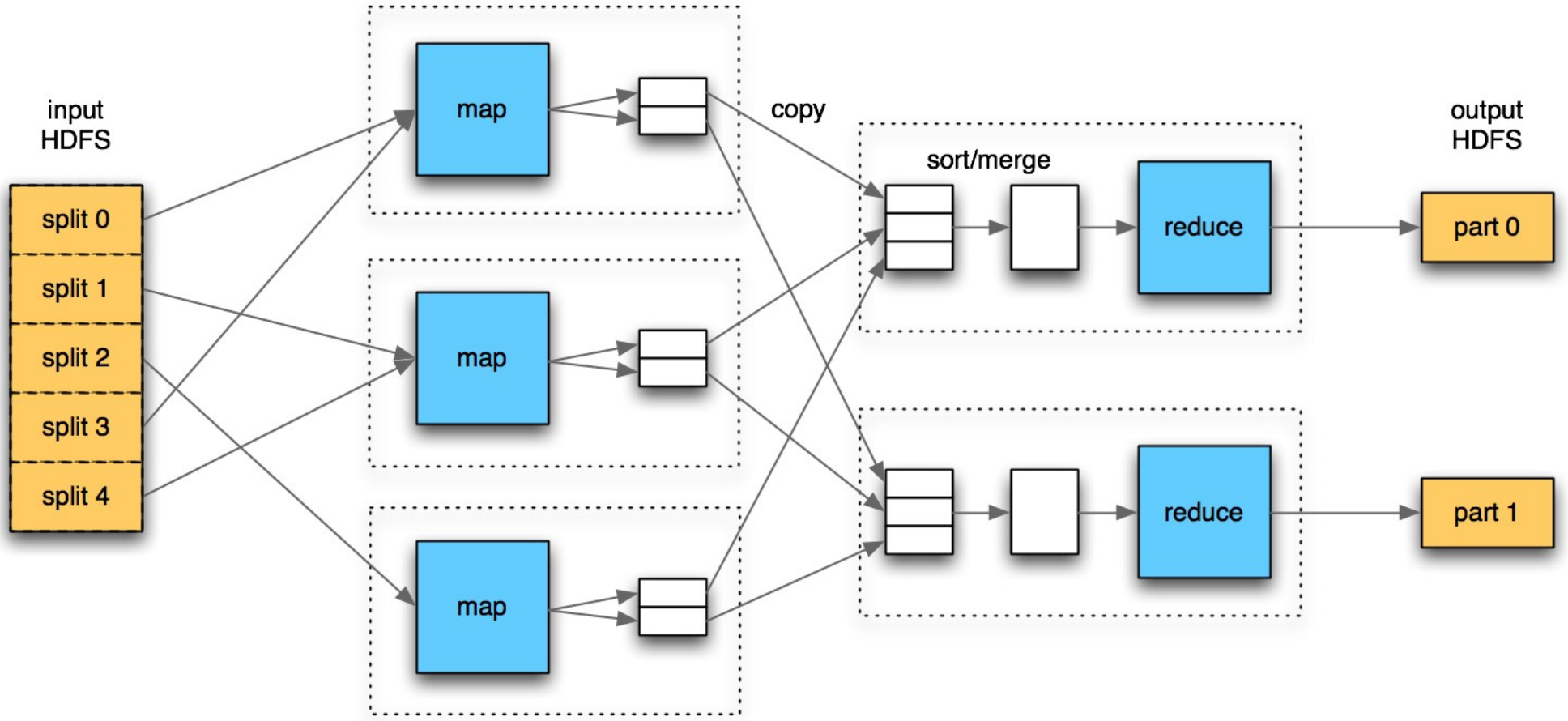
My First MapReduce Program

- General form:
  - Map: (K1, V1) → list(K2, V2)
  - Reduce: (K2, list(V2)) → list(K3, V3)
- grep
  - Map: (offset, line) → [(match, 1)]
  - Reduce: (match, [1, 1, ...]) → [(match, n)]

Logical Flow



Physical Flow



Architecture

- Single Job Tracker
  - accepts job submission
  - divides job into map and reduce tasks
  - parcels out tasks to trackers
  - reschedules failed tasks
- Many Task Trackers
  - run tasks in child VMs
  - inform Job Tracker of progress

Code

```
public void map(LongWritable key, Text val,
    OutputCollector<Text, IntWritable> output,
    Reporter reporter) throws IOException {

  if (pattern.matcher(val.toString()).matches()) {
    output.collect(val, new IntWritable(1));
  }
}

public void reduce(Text key, Iterator<IntWritable> vals,
    OutputCollector<Text, IntWritable> output,
    Reporter reporter) throws IOException {

  int sum = 0;
  while (vals.hasNext()) {
    sum += vals.next().get();
  }
  output.collect(key, new IntWritable(sum));
}
```

Input and Output

- InputFormat produces splits and records
- OutputFormat accepts records
- Example formats
  - TextInputFormat/OutputFormat
  - KeyValueTextInputFormat
  - SequenceFileInputFormat/OutputFormat
- Types are Hadoop Writables or other serialization format

Other Features

- Compression
- Counters
- Partitioner
- DistributedCache
- Aggregation Library
- Data Join Library

More examples

- Sort
  - Map: (k, v) → [(k, v)]
  - Reduce: (k, [v1, v2, ...]) → [(k, v1), (k, v2), ...]
- Word Count
  - Map: (offset, line) → [(word1, 1), (word2, 1), ...]
  - Reduce: (word, [1, 1, ...]) → [(word, n)]

Your Turn

- Choose a partner to work with on one of the problems on the handout.
- Express your solution as a MapReduce program on paper.
- Demonstrate how your program works with a small set of input data.

Problems

1. Find the hits by 5 minute timeslot for a website given its access logs.
2. Find the pages with over 1 million hits in day for a website given its access logs.
3. Find the pages that link to each page in a collection of webpages.
4. Calculate the proportion of lines that match a given regular expression for a collection of documents.
5. Sort tabular data by a primary and secondary column.
6. Find the most popular pages for a website given its access logs.