More Big Data Technologies

Naren Ramakrishnan

VirginiaTech

Recap

- Technologies
  - SQL
  - NoSQL
    - Key-value stores, Column oriented DBs, Document DBs, and graph databases
- ML Algorithms
  - Supervised and unsupervised learning
  - Time series, association analysis

Today

- More Big Data Tech
  - Hadoop
  - Hive
  - Spark

Hadoop

- Open source software
  - Reliable, Scalable, Distributed Computing
- Can scale to thousands of machines
- Written in Java
- Supports many key ingredients for data-intensive applications
  - Map Reduce
  - HDFS (Hadoop Distributed File System)

Map Reduce

- Simple abstraction for scalable computing
  - Hides details of parallelization, fault-tolerance, and data balancing
- Can be viewed as a functional programming paradigm for expressing a broad range of big data analysis tasks

Example (warmup)

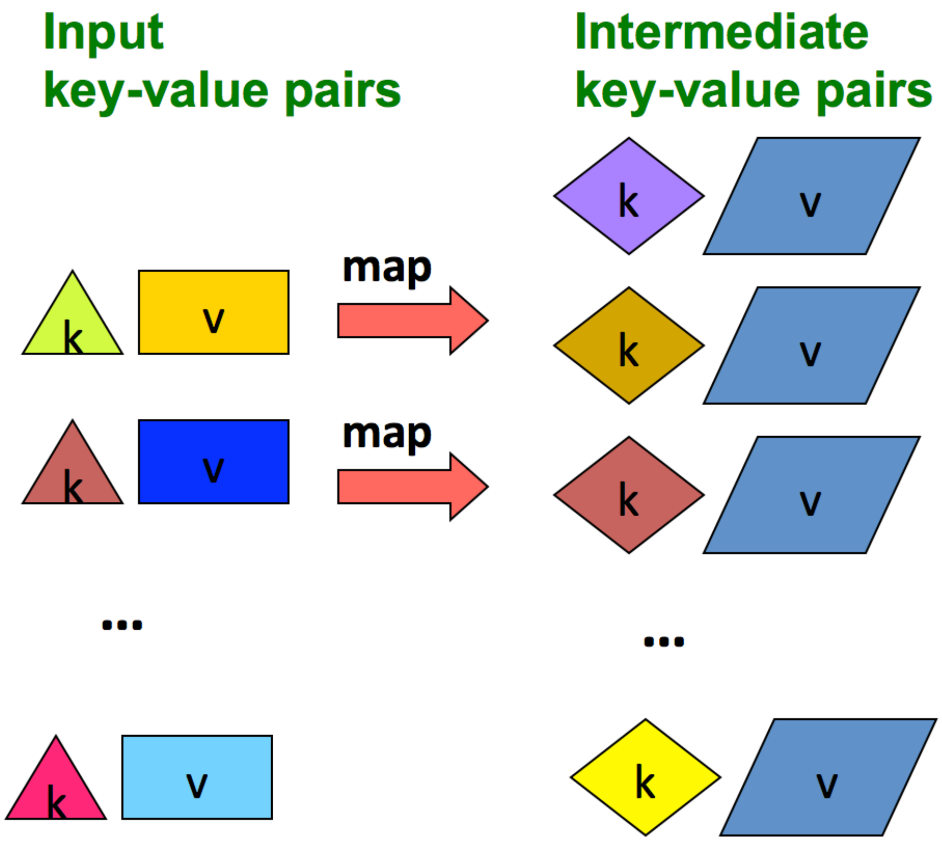
- We have a MASSIVE text document (file)
- Need to count the number of times each distinct word appears in the file
- Example application
  - Count support for political candidates in tweets
  - Analyze web server logs to find popular URLs
- Note:
  - File too large to fit into main memory

How MapReduce works

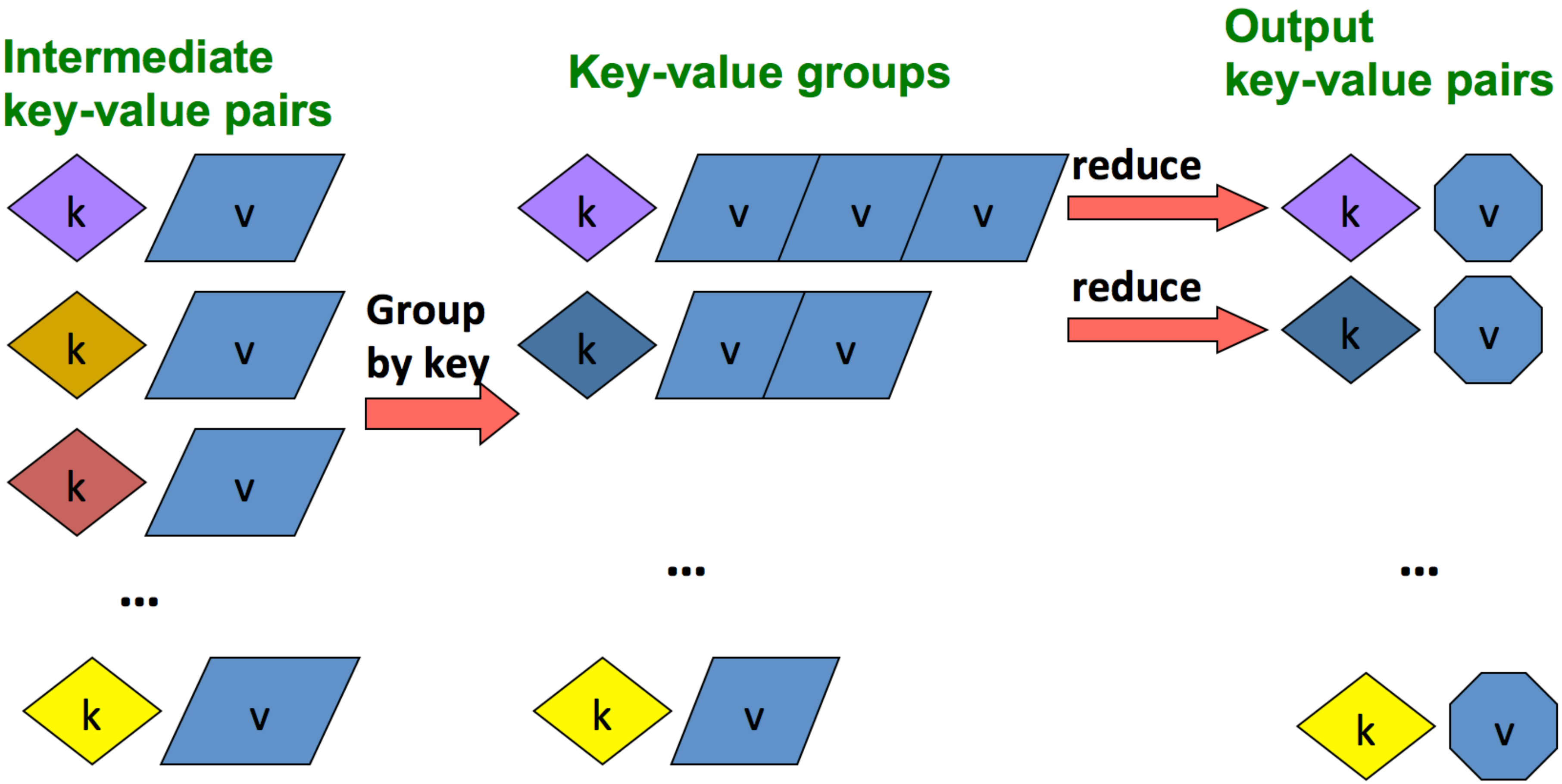
- Sequentially read a lot of data
- Map
  - Extract something you care about
- Group by key
  - Sort and shuffle
- Reduce
  - Aggregate, summarize, filter or transform
- Write the result

Outline stays the same, **Map** and **Reduce** change to fit the problem

MapReduce: Map Step



MapReduce: Reduce Step



In more detail

- **Input:** a set of key-value pairs
- Programmer specifies two methods:
  - **Map(k, v)** → <k', v'>*
    - Takes a key-value pair and outputs a set of key-value pairs
      - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every *(k,v)* pair
  - **Reduce(k', <v'>*)** → <k', v''>*
    - **All values *v'* with same key *k'* are reduced together and processed in *v'* order**
    - There is one Reduce function call per unique key *k'*

MapReduce for word counting

Provided by the programmer

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key

Provided by the programmer

Reduce:
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ....

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
...

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
...

In more detail

```
map(key, value):
// key: document name; value: text of the document
  for each word w in value:
   emit(w, 1)


reduce(key, values):
// key: a word; value: an iterator over counts
   result = 0
   for each count v in values:
      result += v
   emit(key, result)
```

MapReduce as SQL

- select count(*) ← **Reducer**

  from DOCUMENT

  group by word

Mapper

Why is this a big deal?

Map-Reduce environment takes care of:

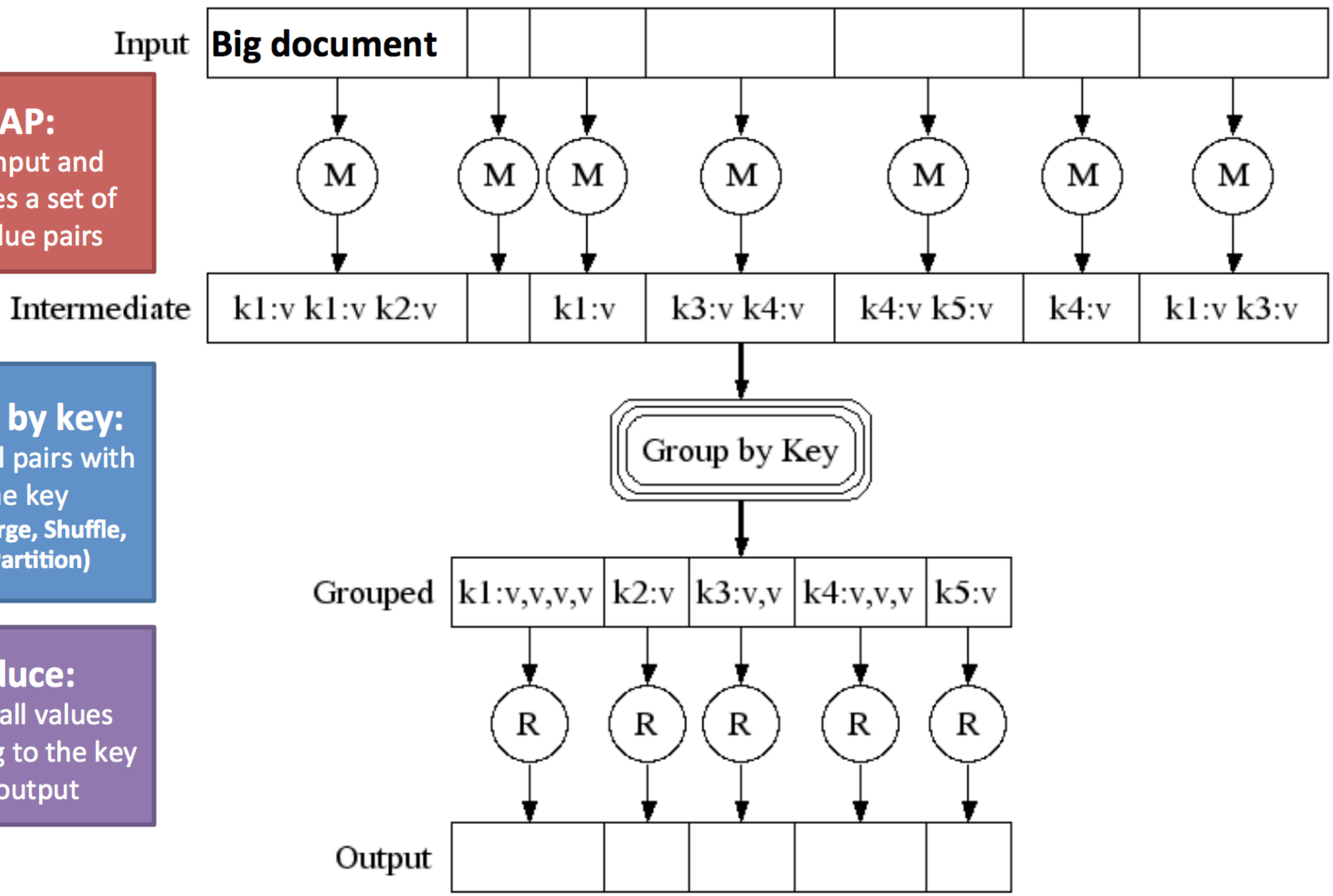
- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the **group by key** step
- Handling machine failures
- Managing required inter-machine communication

Diagrammatic notation

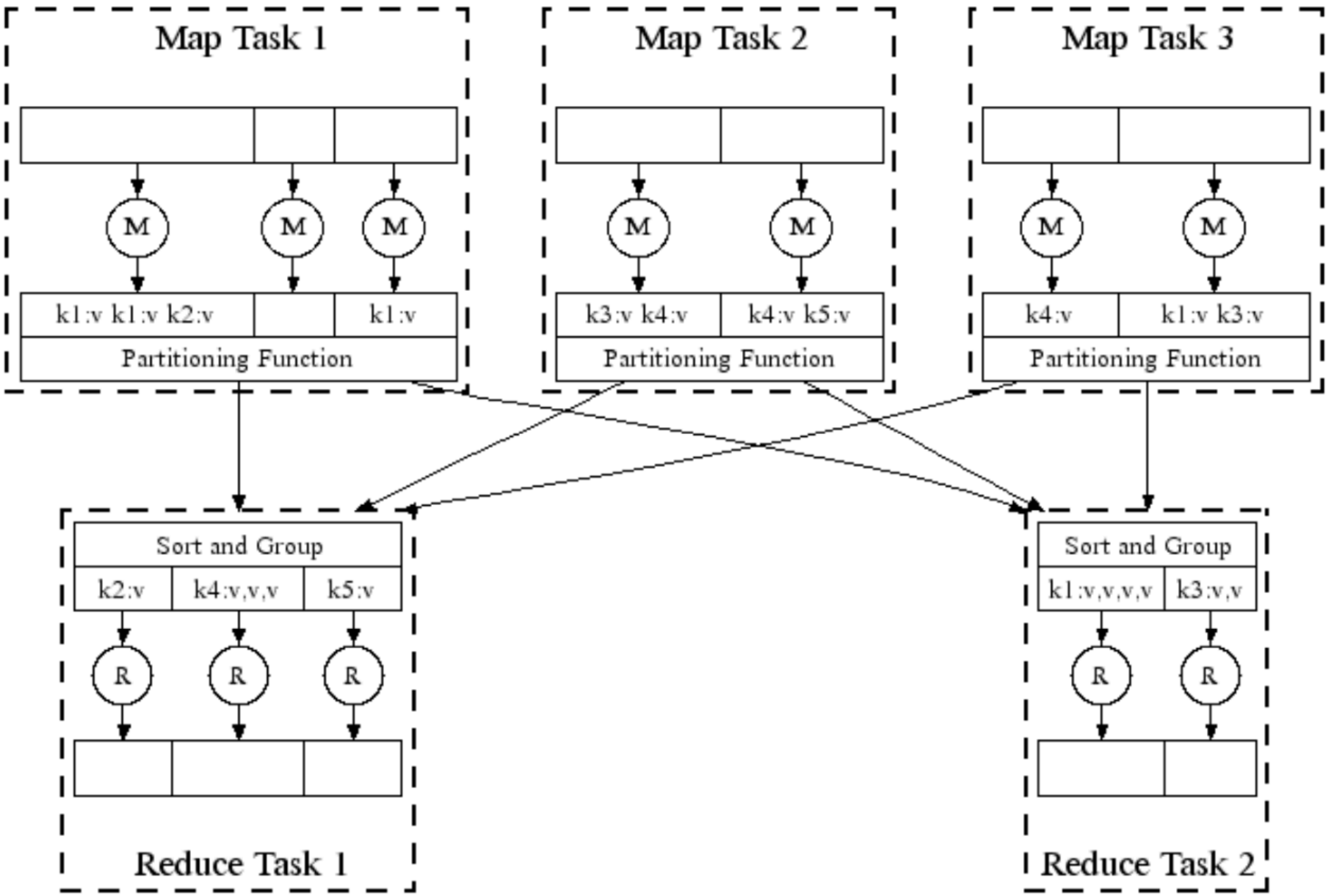
MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output

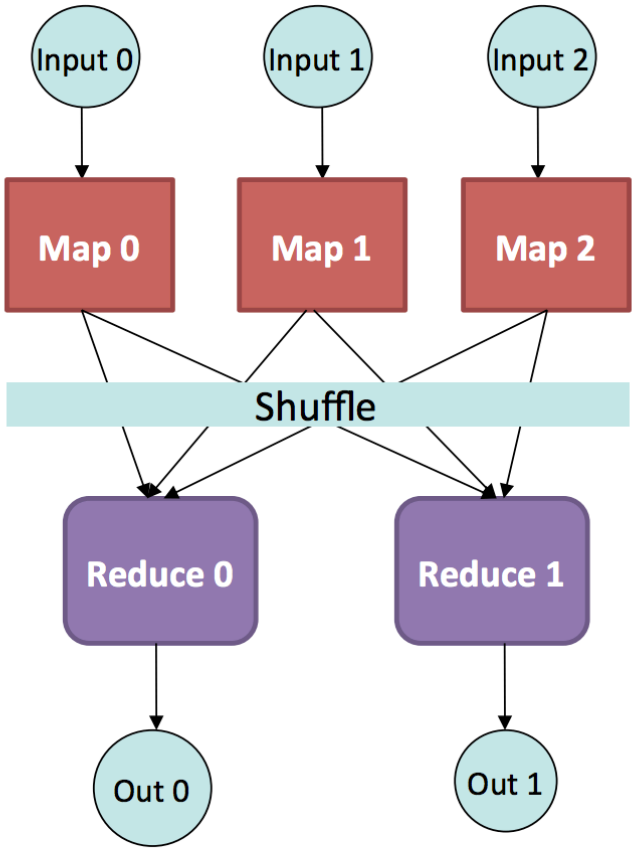


MapReduce in parallel



MapReduce in parallel

- Programmer specifies:
  - Map and Reduce and input files
- **Workflow:**
  - Read inputs as a set of key-value-pairs
  - **Map** transforms input kv-pairs into a new set of k'v'-pairs
  - Sorts & Shuffles the k'v'-pairs to output nodes
  - All k'v'-pairs with a given k' are sent to the same **reduce**
  - **Reduce** processes all k'v'-pairs grouped by key into new k''v''-pairs
  - Write the resulting pairs to files
- All phases are distributed with many tasks doing the work



Data Flow

- Input and final output are stored on a distributed file system (FS):
  - Scheduler tries to schedule map tasks "close" to physical storage location of input data

- Intermediate results are stored on local FS of Map and Reduce workers

- Output is often input to another MapReduce task

Coordination: Master

- Master node takes care of coordination:
  - **Task status:** (idle, in-progress, completed)
  - **Idle tasks** get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its *R* intermediate files, one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to detect failures

Dealing with Failures

- **Map worker failure**
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker
- **Reduce worker failure**
  - Only in-progress tasks are reset to idle
  - Reduce task is restarted
- **Master failure**
  - MapReduce task is aborted and client is notified

Other problems suitable for MR

- Counting, size determination, many statistical estimation routines
- Graph analytics
  - e.g, finding degrees, degree distribution
- Many machine learning algorithms can be cast in terms of MR
  - Often need to “dumb” your algorithms down!

Back to Hadoop

- Hadoop provides
  - Map reduce (MR)
  - Distributed, fault-tolerant file system (HDFS)
    - E.g., Store files multiple times for reliability and to avoid copying costs over the network

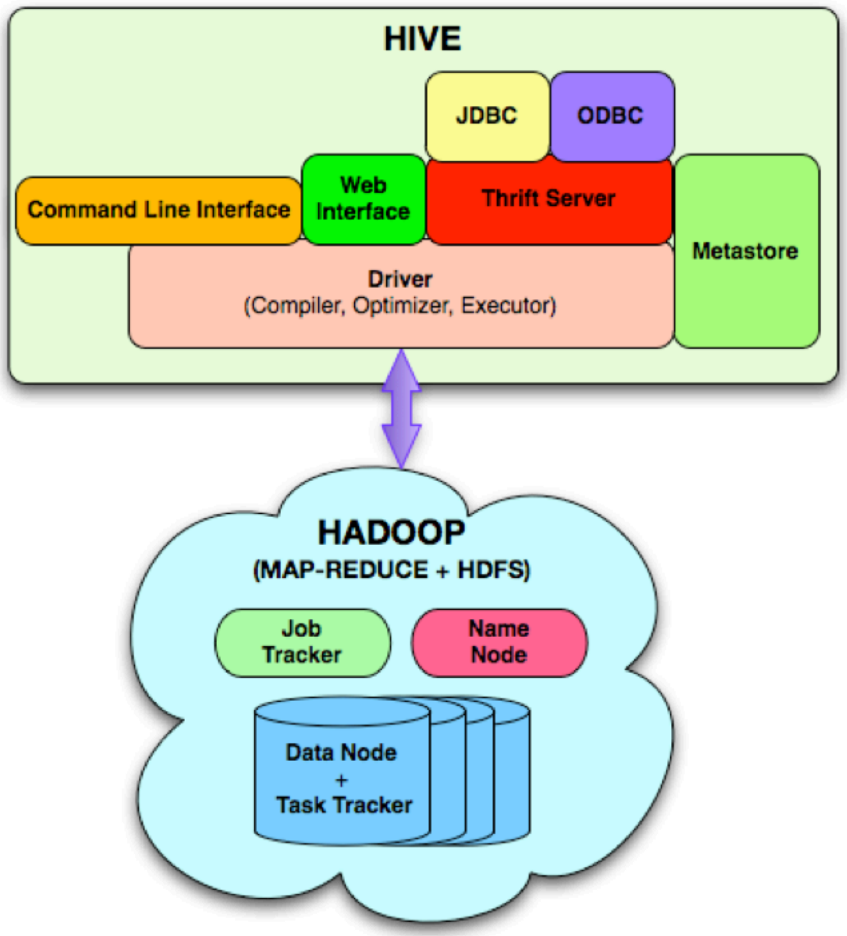
Criticisms of Hadoop/MR

- Still a bit low-level
- Analytic pipelines can get quite complex
- Researchers started developing higher-level substrates/layers over MR
  - Yahoo: Pig
  - Facebook: Hive

Hive

- Supports queries expressed in SQL-like language called HiveQL which are compiled into MR jobs that are executed on Hadoop.
  - Also allows MR scripts!
- Helps structure data into classical concepts like tables, rows, columns, and partitions

Hive architecture



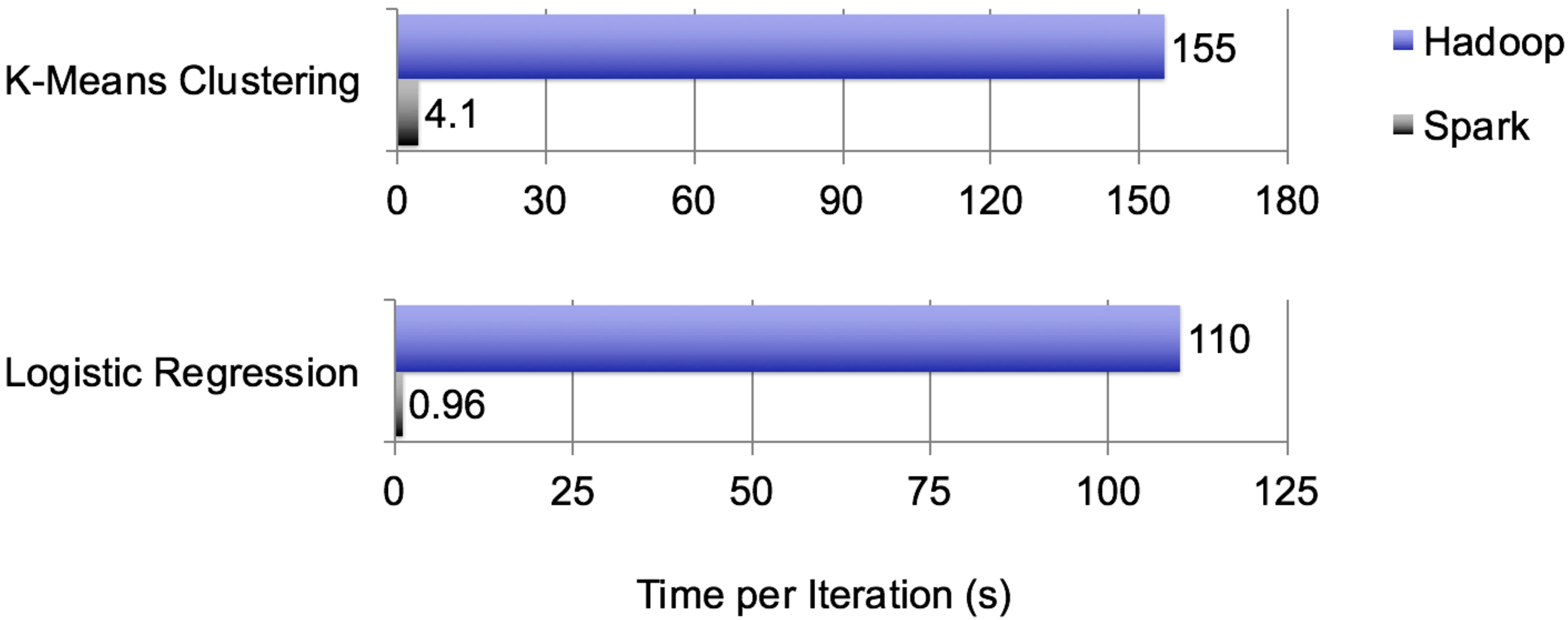
Wordcount in Hive

```
FROM (
MAP doctext USING 'python wc_mapper.py' AS
   (word, cnt)
FROM docs
CLUSTER BY word
) a
REDUCE word, cnt USING 'pythonwc_reduce.py';
```

Apache Spark

- From AMP Lab at Berkeley
- Spark vs Hadoop
  - Like Hadoop, works on distributed data collections
  - Unlike Hadoop, does not provide its own file system
    - But can use HDFS!
- Primarily suited for in-memory analytics;
  - can be 10X faster than MR for batch processing
  - can be 100X faster than MR for in-memory analytics

Example performance



Resilient Distributed Datasets (RDDs)

- Even though data is not written to disk at each step, fault recovery/resiliency is built in because data is distributed across the cluster
  - Immutable collections of objects spread across a cluster
  - Built through parallel transformations (map, filter, etc)
  - Automatically rebuilt on failure
  - Controllable persistence (e.g. caching in RAM)

Using Spark

- APIs in Java, Scala and Python
- Transformations (e.g. map, filter, groupBy, join)
  - Lazy operations to build RDDs from other RDDs
- Actions (e.g. count, collect, save)
  - Return a result or write it to storage
- Spark can read/write to any storage system / format that has a plugin for Hadoop!
  - Examples: HDFS, S3, HBase, Cassandra, Avro, SequenceFile
  - Reuses Hadoop's InputFormat and OutputFormat APIs

What we have learnt thus far

- Map Reduce
- Hadoop
- Hive
- Spark