The MapReduce framework

Dr. Garrett Dancik

Map Reduce background

- Based on a Google paper:
  - https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf
- MapReduce provides the original processing engine for Hadoop, that allows for parallel processing of data in a cluster
- Hadoop is built on Java, but you do not need to write Java programs to use Map Reduce, though Java programs will be faster.

Map Reduce Overview and Word Count Example

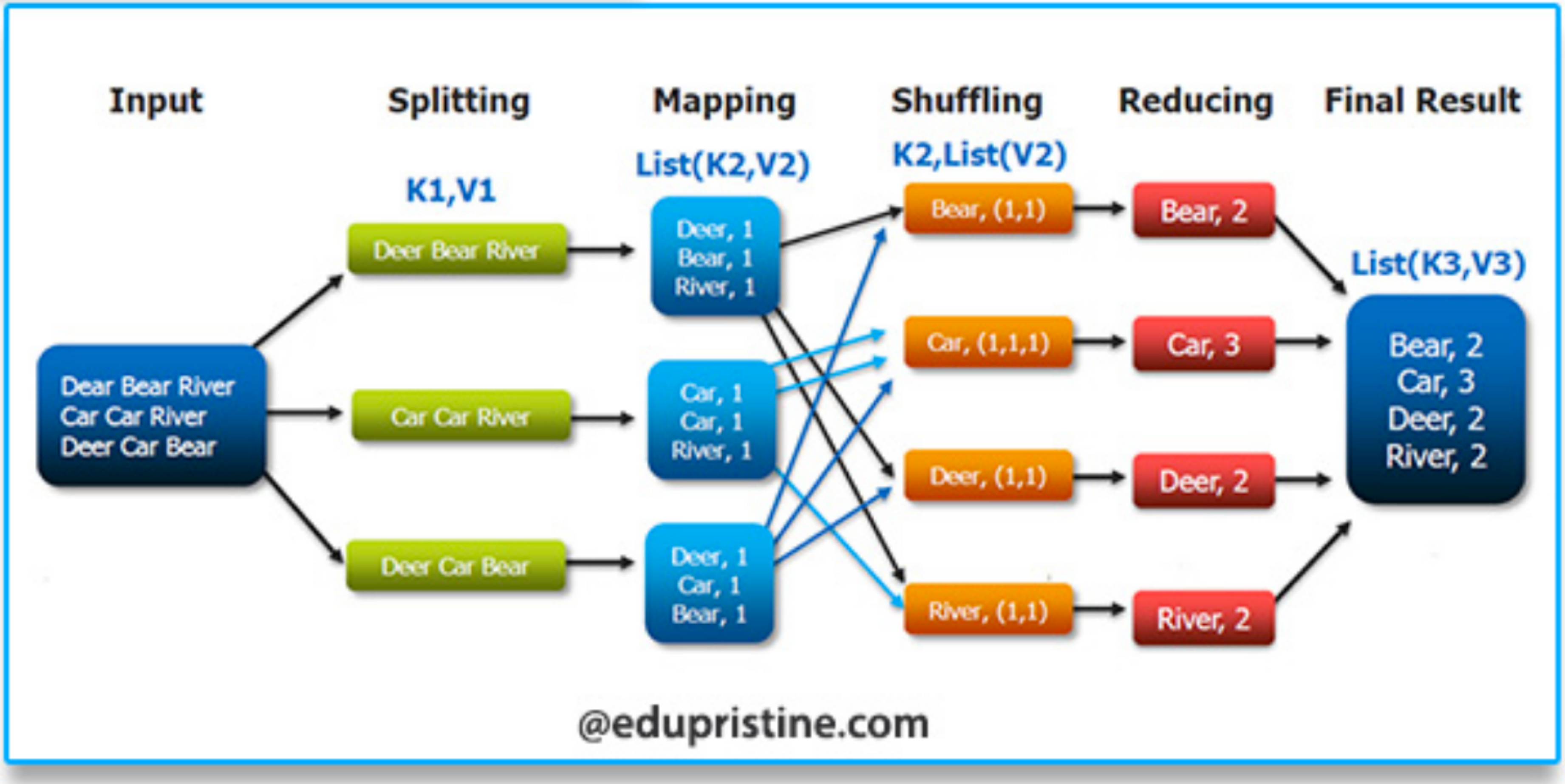
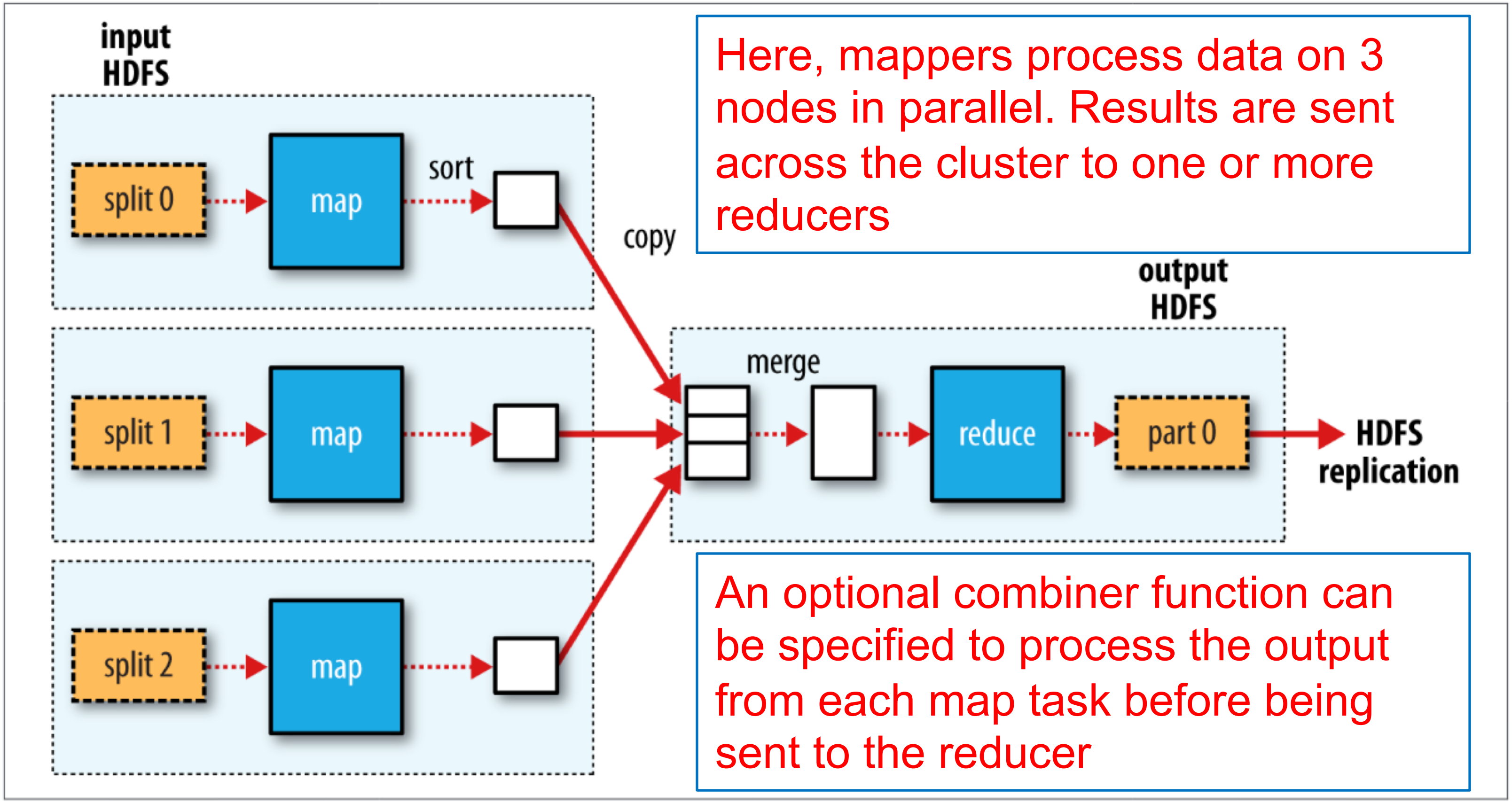


Image source: https://www.edupristine.com/blog/hadoop-mapreduce-framework

Steps of a MapReduce Job

1. Hadoop divides the data into *input splits*, and creates one map task for each split.
2. Each mapper reads each record (each line) of its input split, and outputs a key-value pair
3. Output from the mappers are transferred to reducers as inputs, such that the input to each reducer is sorted by key.
4. The reducer processes the data for each key, and outputs the result. The reducer is optional.

MapReduce data flow with a single reduce task



Here, mappers process data on 3 nodes in parallel. Results are sent across the cluster to one or more reducers

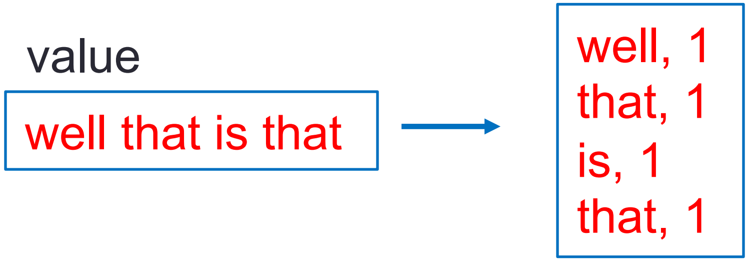
An optional combiner function can be specified to process the output from each map task before being sent to the reducer

Figure 2-3. MapReduce data flow with a single reduce task

Image from White, T. (2015). *Hadoop: The definitive guide*. Sebastopol, CA: OReilly.

Pseudocode for word count mapper and reducer

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");
```

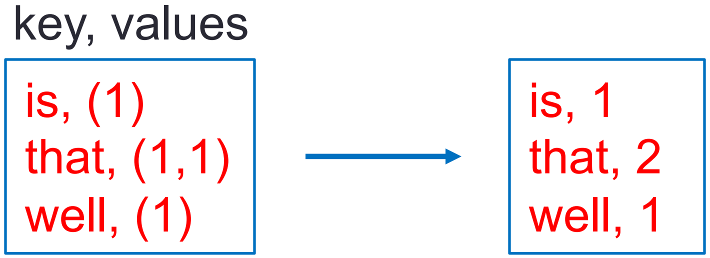


value

well that is that

well, 1
that, 1
is, 1
that, 1

```
reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```



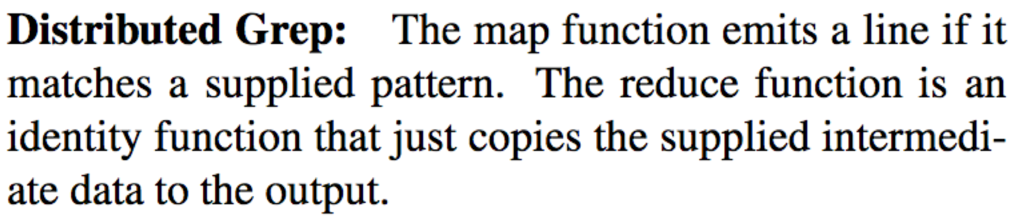
key, values

is, (1)
that, (1,1)
well, (1)

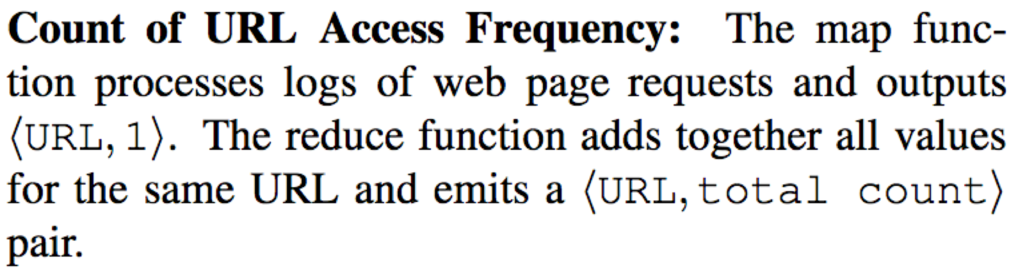
is, 1
that, 2
well, 1

Dean, J., Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters, (2004). OSDI'04: Sixth Symposium on Operating System Design and Implementation, pgs 137-150.

Some Additional MapReduce Applications



Distributed Grep: The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.



Count of URL Access Frequency: The map function processes logs of web page requests and outputs ⟨`URL`, `1`⟩. The reduce function adds together all values for the same URL and emits a ⟨`URL`, `total count`⟩ pair.

Dean, J., Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters, (2004). OSDI'04: Sixth Symposium on Operating System Design and Implementation, pgs 137-150.

MapReduce Streaming

- MapReduce jobs were originally written in Java
  - MapReduce in Java is the most efficient
  - For word count example, see: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
- We will use Hadoop MapReduce Streaming
  - Uses Unix/Linux input and output streams as interface with Hadoop
  - Map input data is passed over standard input to the map function
  - Map output is a tab-separated key,value pair…
  - …which is passed to the reducer over standard input
  - The reduce function reads lines from standard input, sorted by key
  - Any language can be used (we will use Python)

Running a Map Reduce Streaming Job

```
hadoop jar /usr/jars/hadoop-streaming-2.6.0-cdh5.7.0.jar \
-mapper "python3.4 $PWD/mapper.py" \
-reducer "python3.4 $PWD/reducer.py" \
-input "inputFiles" \
-output "outputDirectory"
```

More background and additional options:
https://hadoop.apache.org/docs/r1.2.1/streaming.html

- You may monitor jobs from localhost:8088 (for some links, you will need to replace *quickstart.cloudera* with *localhost*). Some links also require mapping of additional ports.

- The *mapred* terminal command may also be useful:
  - mapred job -list (list jobs)
  - mapred job –kill jobID (kill job with jobID)