UCLA

# CS143
# Map Reduce (Spark)

Professor Junghoo “John” Cho

# Distributed Analytics using Cluster

- Often, our data is non-relational (e.g., flat file) and huge
  - Billions of query logs
  - Billions of web pages
  - ...
- Q: Can we perform analytics on large data quickly using thousands of machines?

# Example 1: Search Log Analysis

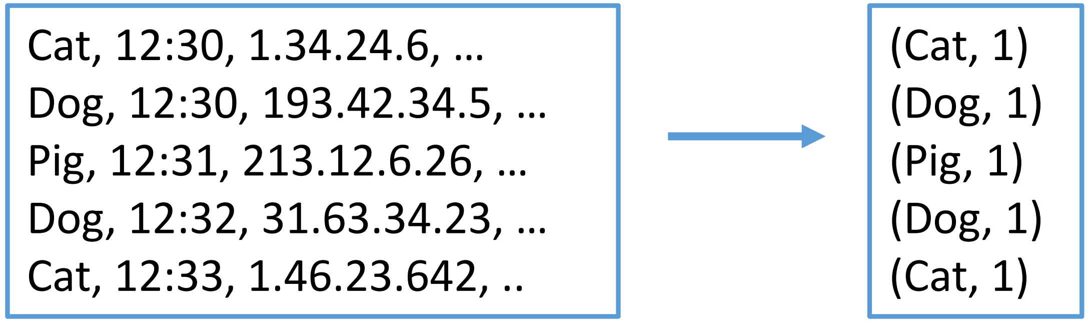
- Log of billions of queries. Count frequency of each query
  - Input query log:

    ```
    cat,time,userid1,ip1,referrer1
    dog,time,userid2,ip2,referrer2
    ...
    ```
  - Output query frequency:

    ```
    cat 200000
    dog 120000
    ...
    ```
- Q: How can we perform this task? How can we parallelize it?

# Example 1: Search Log Analysis (1)

- Step 1: “Transform” each line of query log into (query, 1)

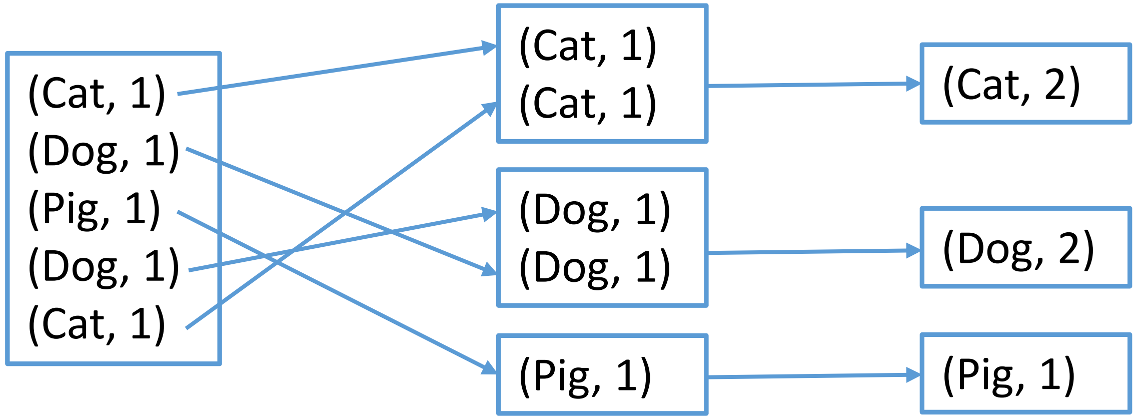


```
Cat, 12:30, 1.34.24.6, …
Dog, 12:30, 193.42.34.5, …
Pig, 12:31, 213.12.6.26, …
Dog, 12:32, 31.63.34.23, …
Cat, 12:33, 1.46.23.642, ..
```

```
(Cat, 1)
(Dog, 1)
(Pig, 1)
(Dog, 1)
(Cat, 1)
```

# Example 1: Search Log Analysis (2)

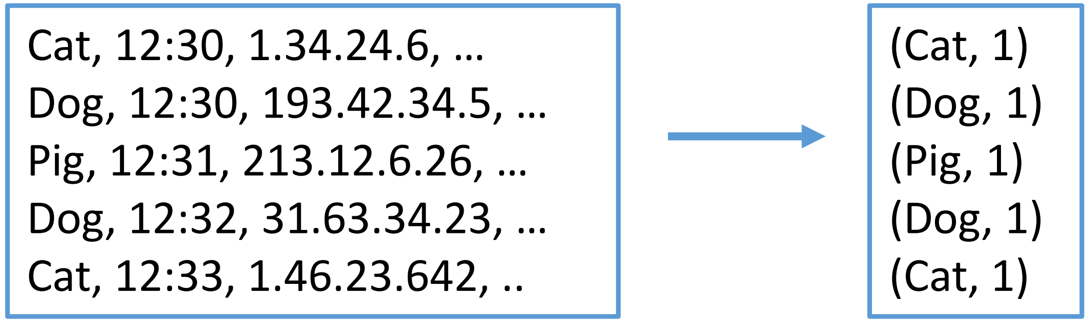
- Step 2: Collect all tuples with the same query and “aggregate” them



- Q: How can we parallelize the two steps?

# Example 1: Search Log Analysis (3)

- Step 1: “Transform” each line of query log into (query, 1)



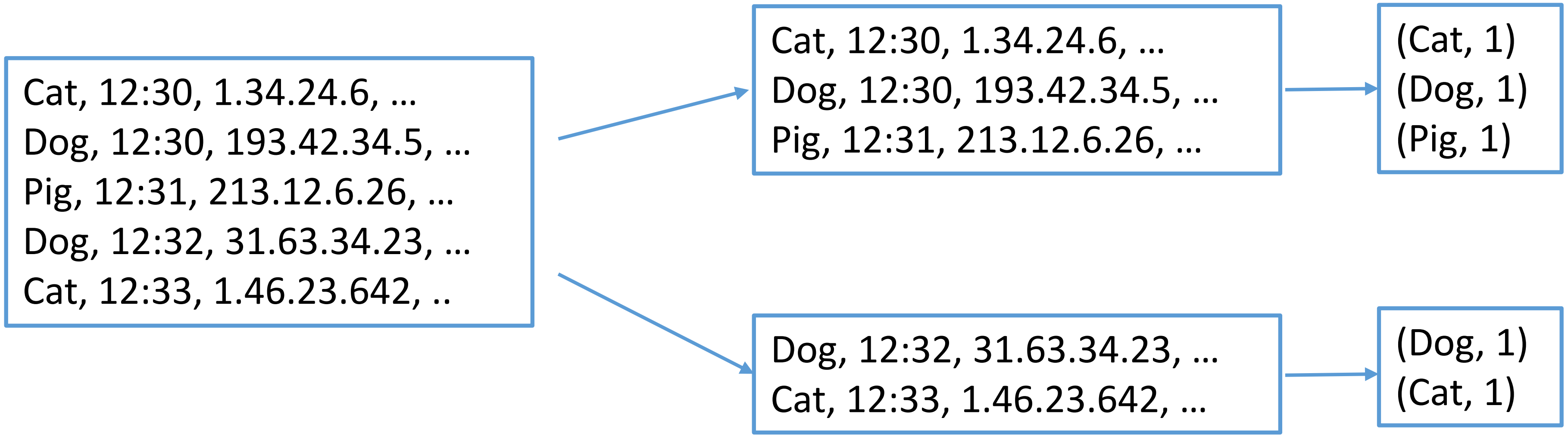
```
Cat, 12:30, 1.34.24.6, …
Dog, 12:30, 193.42.34.5, …
Pig, 12:31, 213.12.6.26, …
Dog, 12:32, 31.63.34.23, …
Cat, 12:33, 1.46.23.642, ..
```

```
(Cat, 1)
(Dog, 1)
(Pig, 1)
(Dog, 1)
(Cat, 1)
```

- Q: Can the transformation of each line done independently of each other?

# Example 1: Search Log Analysis (4)

- Step 1: For parallel processing
  - Split input data into multiple independent chunks
  - Move each chunk to separate machine
  - Perform “transformation” on multiple machines in parallel



Cat, 12:30, 1.34.24.6, …
Dog, 12:30, 193.42.34.5, …
Pig, 12:31, 213.12.6.26, …
Dog, 12:32, 31.63.34.23, …
Cat, 12:33, 1.46.23.642, ..

Cat, 12:30, 1.34.24.6, …
Dog, 12:30, 193.42.34.5, …
Pig, 12:31, 213.12.6.26, …

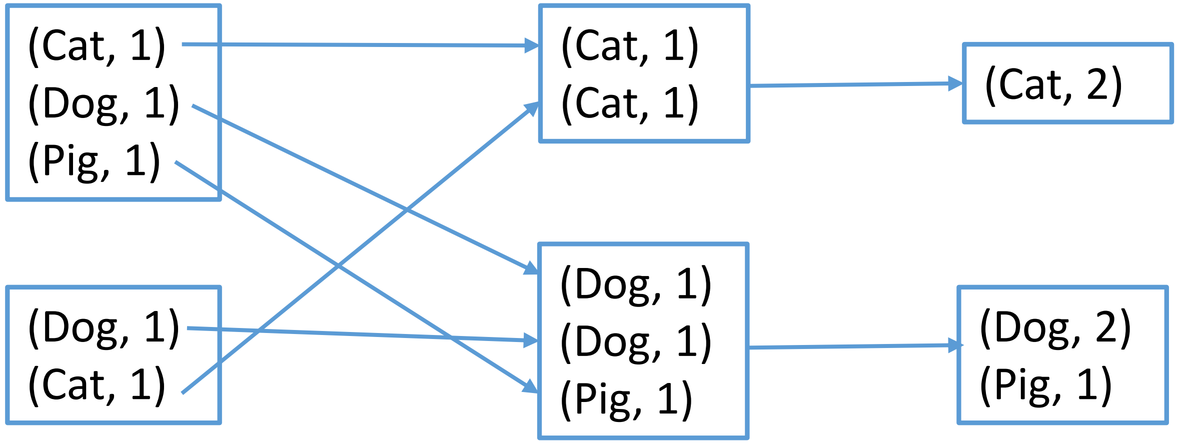
(Cat, 1)
(Dog, 1)
(Pig, 1)

Dog, 12:32, 31.63.34.23, …
Cat, 12:33, 1.46.23.642, …

(Dog, 1)
(Cat, 1)

# Example 1: Search Log Analysis (5)

- Q: How do we parallelize the second “aggregation” step?
- Step 2: For parallel processing
  - Move the tuples with the same query to the same machine
  - Perform aggregation on multiple machine in parallel



# Example 2: Web Indexing

- 1 billion pages. Build “inverted index”
  - Input documents:

    1: cat chases dog

    2: dog loves cat

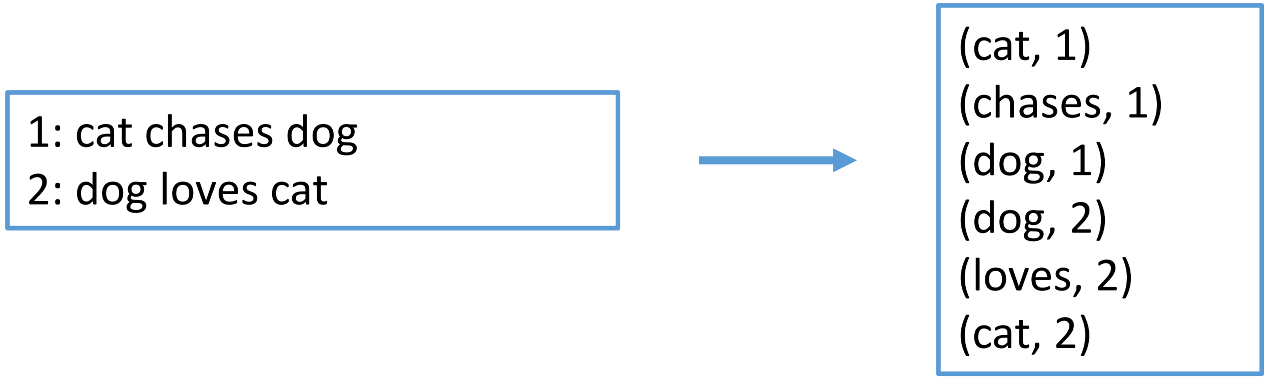
    …
  - Output index:

    cat 1,2,5,10,20

    dog 1,2,3,8,9
- Q: How can we do this?

# Example 2: Web Indexing (1)

- Step 1: “Transform” every document into (word, doc_id) tuples

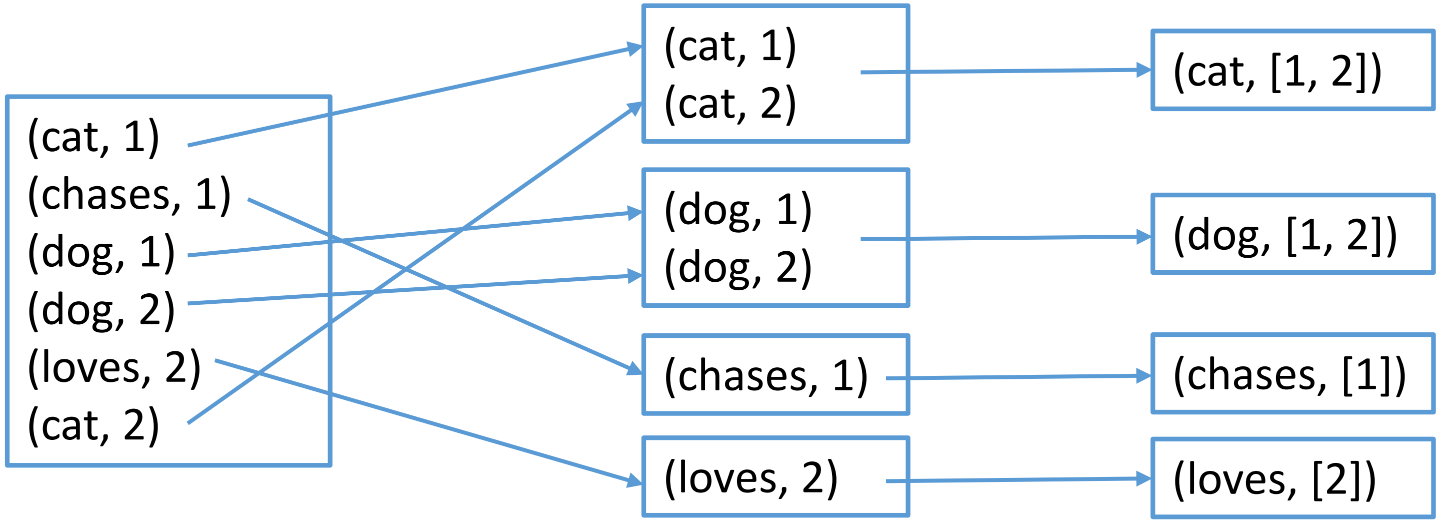


1: cat chases dog
2: dog loves cat

(cat, 1)
(chases, 1)
(dog, 1)
(dog, 2)
(loves, 2)
(cat, 2)

# Example 2: Web Indexing (2)

- Step 2: Collect all tuples with the same word and “aggregate” (or concatenate) the doc_id’s



(cat, 1)
(chases, 1)
(dog, 1)
(dog, 2)
(loves, 2)
(cat, 2)

(cat, 1)
(cat, 2)

(dog, 1)
(dog, 2)

(chases, 1)

(loves, 2)

(cat, [1, 2])

(dog, [1, 2])

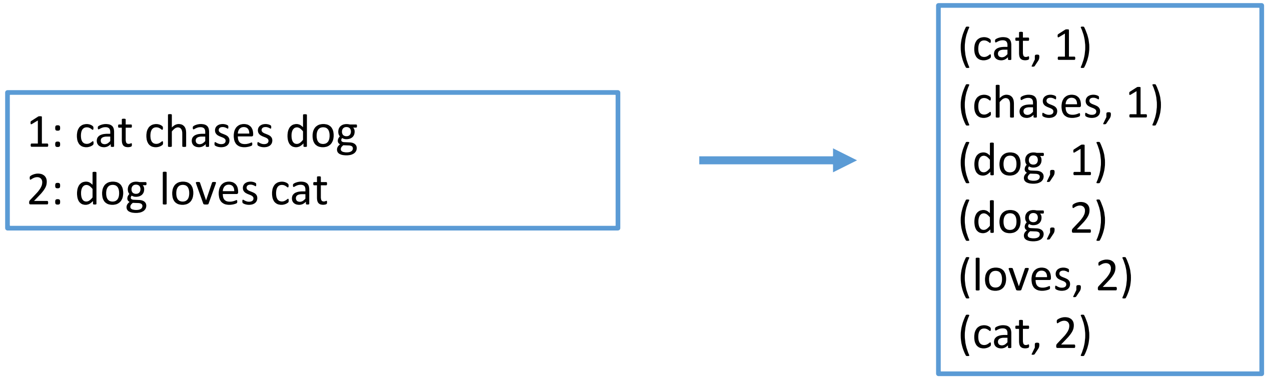
(chases, [1])

(loves, [2])

- Q: How can we parallelize the two steps on multiple machines?

# Example 2: Web Indexing (3)

- Step 1: “Transform” every document into (word, doc_id) tuples



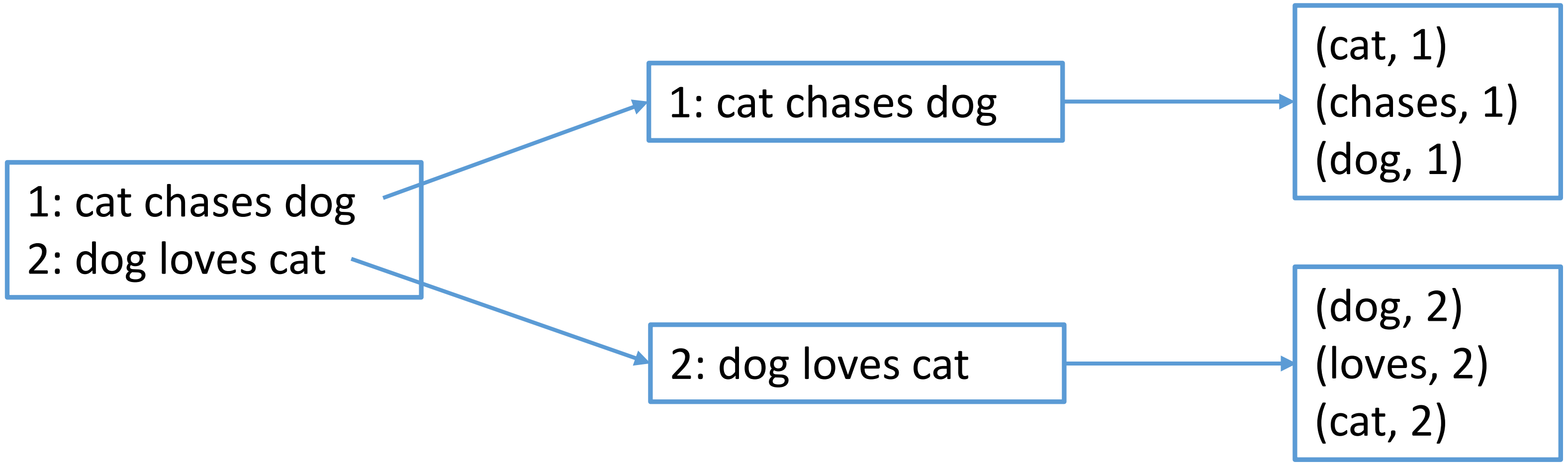
1: cat chases dog
2: dog loves cat

(cat, 1)
(chases, 1)
(dog, 1)
(dog, 2)
(loves, 2)
(cat, 2)

- Q: Can the transformation of each document be done independently of each other?

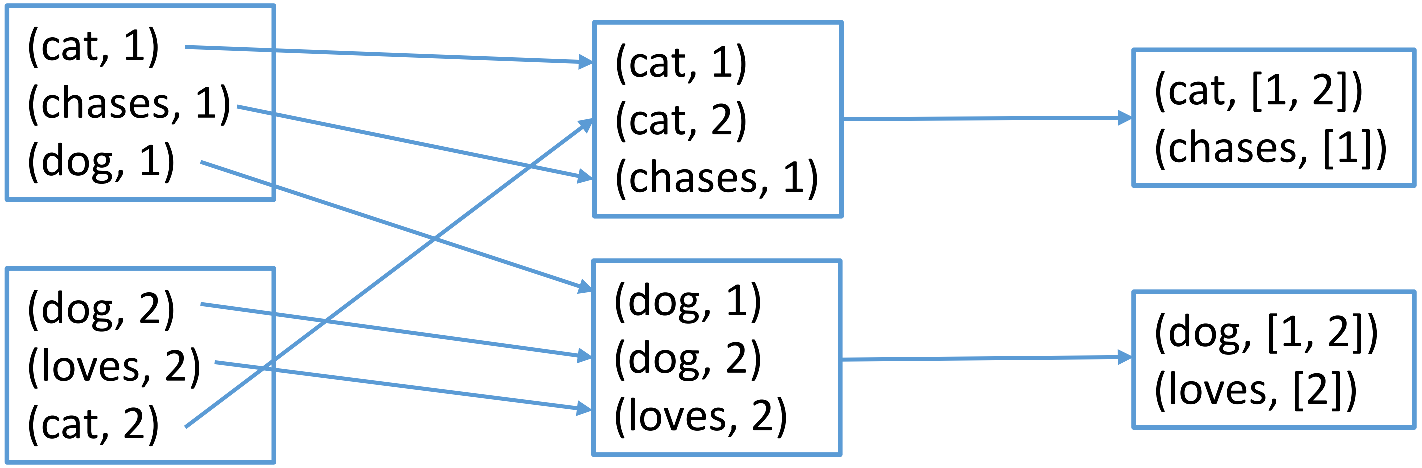
# Example 2: Web Indexing (4)

- Step 1: For parallel processing
  - Split input data into multiple independent chunks
  - Move each chunk to separate machine
  - Perform “transformation” on multiple machines in parallel



# Example 2: Web Indexing (5)

- Q: How can we parallelize second “concatenation step”?
- Step 2: For parallel processing
  - Move the tuples with the same word to the same machine
  - Perform aggregation on multiple machine in parallel



(cat, 1)
(chases, 1)
(dog, 1)

(dog, 2)
(loves, 2)
(cat, 2)

(cat, 1)
(cat, 2)
(chases, 1)

(dog, 1)
(dog, 2)
(loves, 2)

(cat, [1, 2])
(chases, [1])

(dog, [1, 2])
(loves, [2])

# Generalization (1)

- Input data consists of multiple independent units
  - Each line of query log
  - Each web page
- Partition input data into multiple “chunks” and distribute them to multiple machines
- Transformation/map input into (key, value) tuples
  - Query log: query_log_line → (query, 1)
  - Indexing: web_page → ($\text{word}_1$, page_id), ($\text{word}_2$, page_id), …
- Reshuffle tuples of the same key to the same machine
- Aggregate/reduce the tuples of same keys
  - Query log: (query, 1), (query, 1), … → (query, count)
  - Indexing: (word, 1), (word, 3), … → (word, [1, 3, …])
- Collect and output the aggregation results

# Generalization (2)

- The two examples are almost the same except
  - “The mapping function”
    - Query log: query_log_line → (query, 1)
    - Indexing: web_page → (word$_1$, page_id), (word$_2$, page_id), …
  - “The reduction function”
    - Query log: (query, 1), (query, 1), … → (query, count)
    - Indexing: (word, 1), (word, 3), … → (word, [1, 3, …])

# MapReduce Model

- Programmer provides
  1. Map function: “unit data” → $(k', v'), (k'', v''), \dots$
  2. Reduce function: $(k, v_1), (k, v_2), \dots \rightarrow (k, \mathrm{aggr}(v_1, v_2, \dots))$
- MapReduce handles the rest
  - Automatic data partition, distribution, and collection
  - Failure and speed-disparity handling
- Many systems exist supporting MapReduce model

# Hadoop

- First open-source implementation of MapReduce and GFS (Google File System)
  - Implemented in Java
- User implements map and reduce functions as:
  - Mapper.map(key, value, output, reporter)
  - Reducer.reduce(key, value, output, reporter)

# Spark

- Open-source cluster computing infrastructure
- Supports MapReduce and SQL
  - Supports data flow more general than simple MapReduce
- Input data is converted into RDD (resilient distributed dataset)
  - A collection of independent tuples
  - The tuples are automatically distributed and shuffled by Spark
- Supports multiple programming languages
  - Scala, Java, Python, …
  - Scala and Java are much more performant than others

# Spark Example: Count words

```
lines = sc.textFile("input.txt")
words = lines.flatMap(lambda line: line.split(" "))
word1s = words.map(lambda word: (word, 1))
wordCounts = word1s.reduceByKey(lambda a,b: a+b)
wordCounts.saveAsTextFile("output")
```

# Key Spark Functions

- Transformation: Convert RDD tuple into RDD tuple(s)
    - map(): convert one input tuple into one output tuple
    - flatMap(): convert one input into multiple output tuples
    - reduceByKey(): specify how two input “values” should be aggregated
    - filter(): filter out tuples based on condition
- Action: Perform “actions” on RDD
    - saveAsTextFile(): save RDD in a directory as text file(s)
    - collect(): create Python tuples from Spark RDD
    - textFile(): create RDD from text (each line becomes an RDD tuple)

# What We Learned

- Large-scale data analytics on distributed cluster
- MapReduce model
- Spark