MapReduce for Key-Value Data

Computing on Data

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Describe how MapReduce computes on key-value data.
- Design map and reduce functions for a computation task.
- Specify how relational operations can be performed by MapReduce.

MapReduce for Kev-Value Data

- A programming model for processing key-value data.
- For a **simplified** data processing framework over **large** clusters.
- Created at Google in 2004.
- Many implementations
  - E.g., Apache Hadoop

Apache Hadoop

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many ter-

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-

Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004: 137-150

Map and Reduce in Functional Programming

- MapReduce: Inspired by map and reduce in functional programming.
- Map:
  - Input: Function M, list $L = [e_1, \dots, e_n]$
  - Output: map $M\ L = [M(e_1), \dots, M(e_n)]$
    - Apply given function to every element of the list.
  - E.g., map square [1, 2, 3, 4, 5] = [1, 4, 9, 16, 25]

- Reduce:
  - Input: Function R, list $L = [e_1, \dots, e_n]$
  - Output: reduce $R\ L = R(e_1, \dots, e_n)$.
    - Apply given function to aggregate all elements of the list.
  - E.g., reduce sum [1, 4, 9, 16, 25] = 55

Computing on K-V Data with MapReduce

- Database DB = $[e_1, \ldots, e_n]$, where $e_i = (k_i, v_i)$, a key-value pair.
- Programmer gives map function M and reduce function R.
- Execute M and R in MapReduce system with steps

  Map → Group → Reduce.

- **Map** on each $e_i = (k_i, v_i)$ with map function M.
  - $M(k_i, v_i) \rightarrow [(k_{i1}, v_{i1}), \ldots, (k_{im}, v_{im})]$
- **Group**
  - Organize (k_{ij}, v_{ij}) pairs by key k_{ij} -- each group is $(k_{ij},\ [values\ with\ key\ k_{ij}])$.
- **Reduce** on each group $(k_{ij},\ [values])$ by reduce function R.
  - Output: $[k_{ij}, R(values\ values\ with\ key\ k_{ij})]$

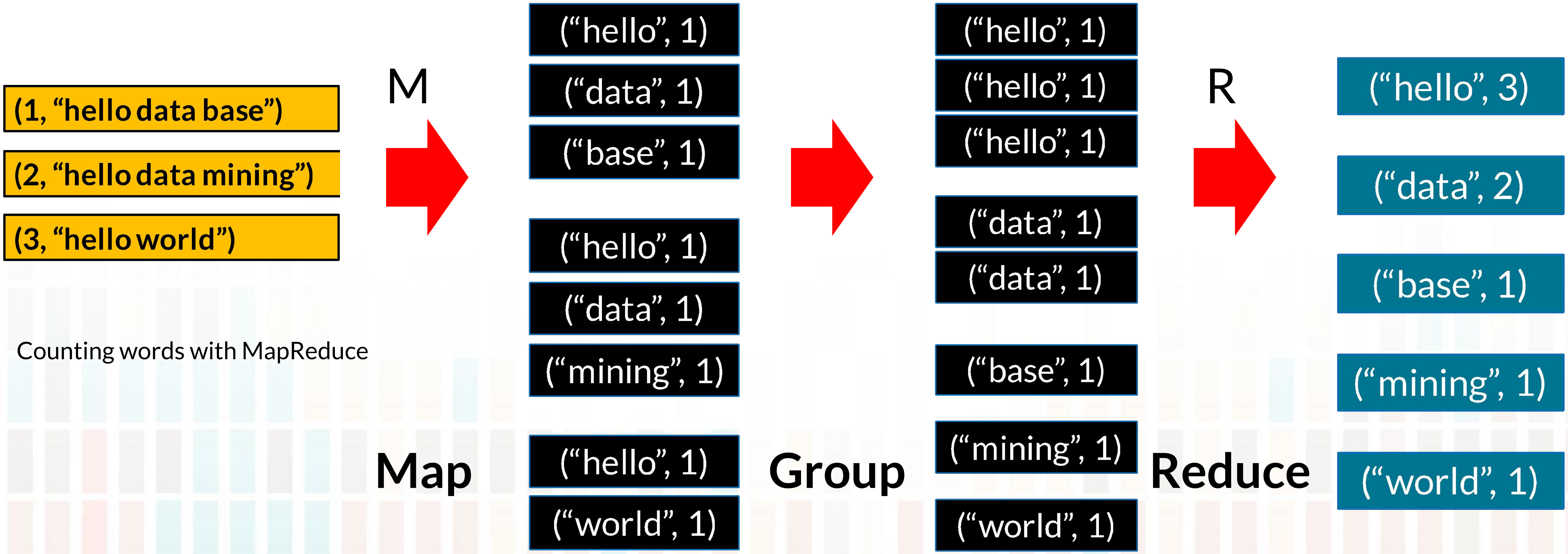
Example: Generating Word Cloud

- Application: Generating word cloud by counting words in database

Generating word cloud by counting words in database

The Standard Word Counting Example

- **function** M(k, v): **for each** word w **in** v: emit (w, 1)
- **function** R (k, values): emit (k, sum(values))



Counting words with MapReduce

Relational Operation with MapReduce?

- $\sigma_{\text{brewer}=\text{“Boston Beer”}}\text{Beers}$
- **function** M(k, v):. **if** v = “Boston Beer”: **emit** (k, v)
- **function** R (k, v): emit (k, v)

Key	Value
“Sam Adams”	“Boston Beer”
“Bud”	”AB InBev”
“Bud Lite”	”AB InBev”
“Coors”	“Coors”

M

(“Sam Adams”, “Boston Beers”)

(“Sam Adams”, “Boston Beers”)

R

(“Sam Adams”, “Boston Beers”)

Map **Group** **Reduce**

Selection operation with MapReduce

Food for Thought

*Can we use MapReduce to perform **θ-join** over relations organized as key-value data?*

$$\textbf{Brewers} \bowtie_{\text{Brewer.key=Price.key AND brewer=“AB InBev” AND price<5.0}} \textbf{Price}$$

Brewer

Key	Value
“Sam Adams”	(brewer, “Boston Beer”)
“Bud”	(brewer, ”AB InBev”)
“Bud Lite”	(brewer, ”AB InBev”)
“Coors”	(brewer, “Coors”)

Brewer relation in the key-value model

Price

Key	Value
“Sam Adams”	(price, 5.0)
“Bud”	(price, 3.0)
“Bud Lite”	(price, 6.5)
“Coors”	(price, 2.5)

Price relation in the key-value model