CIS 560 – Database System Concepts

Lecture 31

NoSQL - MapReduce

November 18, 2013

Credits for slides: Wisdom, Alberton, Pokorný, Hoekstra.

Reminders

- Exam 2 (assignments 6-9) – 11/20
  - Assignments 6-9
  - Lecture notes 18-28
  - Textbook 17.1-17.4, 18.1-18.3, 18.8, 14.1-14.2, 15.1-15.6, 16
  - Feel free to bring one page of notes (front and back)
- Project - DB implementation and queries due 11/22
- Quiz from NoSQL lectures – 12/06

Where we are

- Today: MapReduce framework
  - MapReduce: Simplified Data Processing on Large Clusters. Jeffrey Dean and Sanjay Ghemawat. OSDI'04.
  - Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer, 2010.

    http://lintool.github.io/MapReduceAlgorithms/
- Next: Hive, Pig Latin
  - Hive – A Petabyte Scale Data Warehouse Using Hadoop
  - Pig Latin: A Not-So-Foreign Language for Data Processing

NoSQL – Two Main Incarnations

- NoSQL framework - MapReduce
  - Originally from Google, open source Hadoop
    - No data model, data stored in files
    - User provides specific functions
    - System provides data processing “glue”, fault-tolerance, scalability
- NoSQL ``databases”

Typical Large-Data Problem

Map

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Reduce

Key idea: provide a functional abstraction for these two operations

(Dean and Ghemawat, OSDI 2004)

Example: Word Count

- We have a large file of words, one word to a line
- Count the number of times each distinct word appears in the file
- Sample application: analyze web server logs to find popular URLs

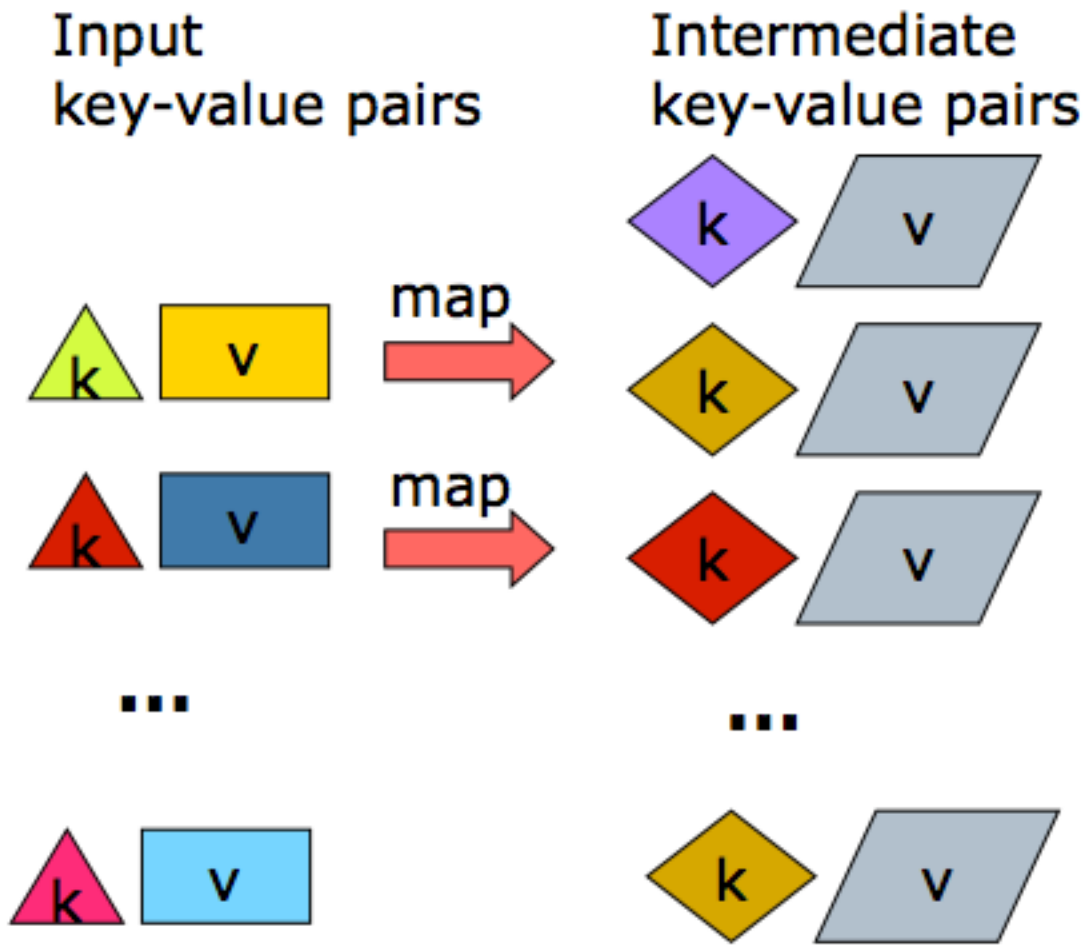
Map and Reduce Functions

- Input: a set of key/value pairs
- User supplies two functions:
  - map(k,v) → list(k1,v1)
  - reduce(k1, list(v1)) → v2
- (k1,v1) is an intermediate key/value pair
- All values with the same key are sent to the same reducer
- Output is the set of (k1,v2) pairs

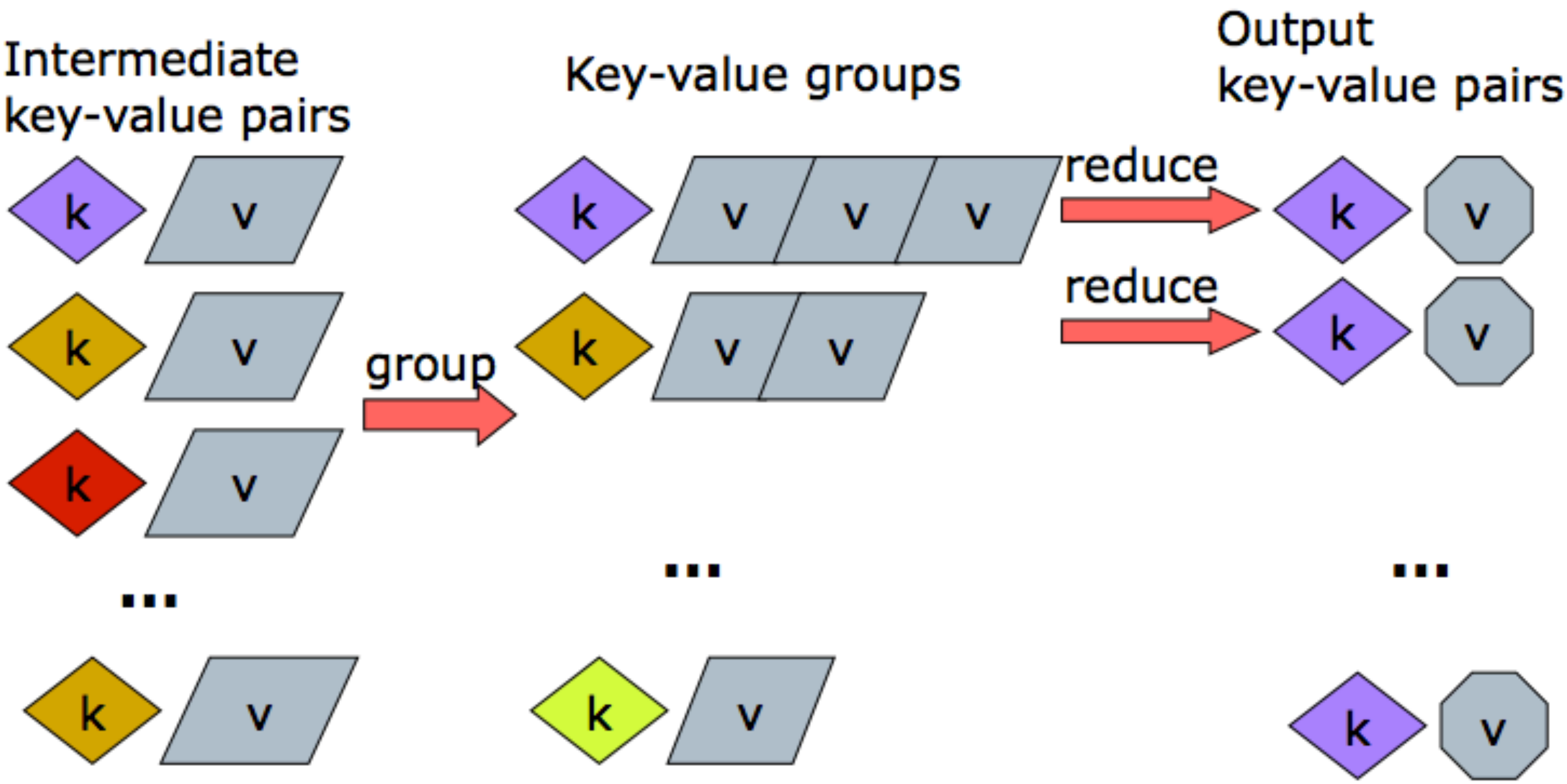
Example: Count word occurrences

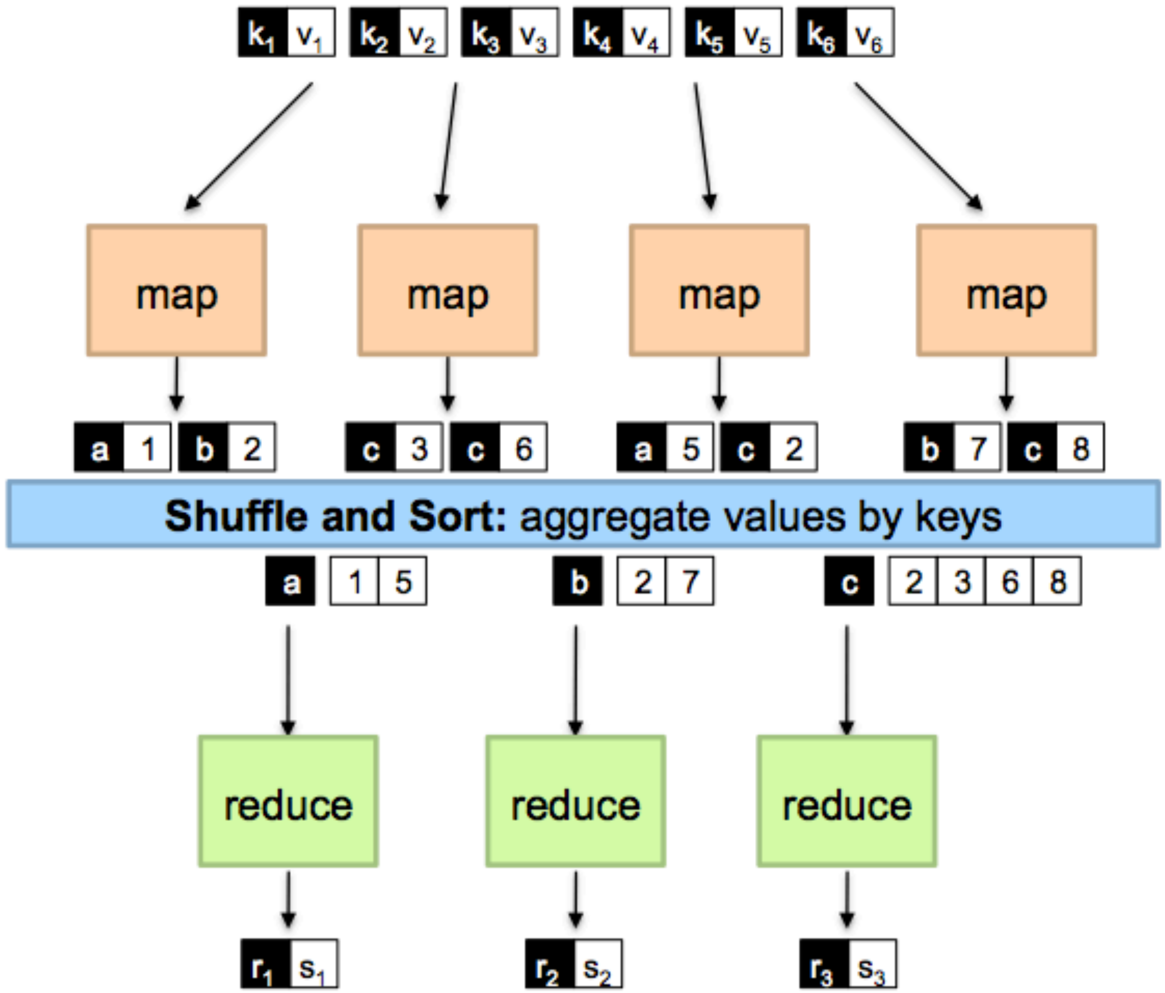
```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, 1);

reduce(String output_key, Iterator<int>
  intermediate_values):
  // output_key: a word
  // output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
    result += v;
 Emit(result);
```

MapReduce: The Map Step



MapReduce: The Reduce Step




k1 v1 k2 v2 k3 v3 k4 v4 k5 v5 k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Remember that…

- Barrier between map and reduce phases
  - But we can begin copying intermediate data earlier
- Keys arrive at each reducer in sorted order
  - No enforced ordering *across* reducers

MapReduce

- Programmers specify the map and reduce functions.
- The execution framework handles everything else…

- Not quite…usually, programmers also specify:

combine (k1,list(v1)) →v2

partition (k1, number of partitions) → partition for k1

Importance of Local Aggregation

- Ideal scaling characteristics:
  - Twice the data, twice the running time
  - Twice the resources, half the running time
- Why can’t we achieve this?
  - Synchronization requires communication
  - Communication kills performance
- Thus… avoid communication!
  - Reduce intermediate data via local aggregation
  - Combiners can help

Combiners

- Often a map task will produce many pairs of the form (k,v1), (k,v2), … for the same key k
  - E.g., popular words in Word Count
- Can save network traffic (time) by pre-aggregating at mapper
- Mini-reducers that run in memory after the map phase
  - combine(k1, list(v1)) → v2
  - Usually same as reduce function
- Works only if reduce function is commutative and associative

Mappers

- Java object that implements the Map method.
- A mapper object is initialized for each map task (associated with aparticular sequence of key-value pairs called an input split).
- A hook is provided in the API to run programmer-specified code.
- Mappers can read in “side data”, providing an opportunity to load state, static data sources, dictionaries, etc*. These method calls occur in the context of the same Java object, therefore it is possible to preserve state across multiple input key-value pairs within the same map task.*
- The Map method is called on each key-value pair by the execution framework.
- After all key-value pairs in the input split have been processed, the mapper object provides an opportunity to run programmer-specified termination code.

Reducers

- Java object that implements the Reduce method
- A reducer object is initialized for each reduce task.
- The Hadoop API provides hooks for programmer-specified initialization and termination code.
- For each intermediate key in the partition (defined by the partitioner), the execution framework repeatedly calls the Reduce method with an intermediate key and an iterator over all values associated with that key.
- Since this occurs in the context of a single object, it is possible to preserve state across multiple intermediate keys (and associated values) within a single reduce task.

Example: Word Count

- We have a large file of documents, one document to a line
- Count the number of times each distinct word appears in the file

Word Count: Baseline

class Mapper
    method Map(docid a, doc d)
        for all term $t \in$ doc d **do**
            Emit(term t, count 1)

class Reducer
    method Reduce(term t, counts $[c_1, c_2, \ldots]$)
        $sum \leftarrow 0$
        for all count $c \in$ counts $[c_1, c_2, \ldots]$ **do**
            $sum \leftarrow sum + c$
        Emit(term t, count s)

What's the impact of combiners?

Word Count: Version 1

```
class Mapper
    method Map(docid a, doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1                    ▷ Tally counts for entire document
        for all term t ∈ H do
            Emit(term t, count H{t})
```

Are combiners still needed?

Word Count: Version 2

class MAPPER
    method INITIALIZE
        $H \leftarrow$ new ASSOCIATIVEARRAY
    method MAP(docid a, doc d)
        for all term $t \in$ doc d **do**
            $H\{t\} \leftarrow H\{t\} + 1$ $\triangleright$ Tally counts *across* documents
    method CLOSE
        for all term $t \in H$ **do**
            EMIT(term t, count $H\{t\}$)

Key: preserve state across input key-value pairs!

Are combiners still needed?

Design Pattern for Local Aggregation

- "In-mapper combining"
  - Fold the functionality of the combiner into the mapper by preserving state across multiple map calls
- Advantages
  - Speed
- Disadvantages
  - Explicit memory management required
  - Potential for order-dependent bugs

MapReduce Example: Web log analysis (1)

Each record: UserID, URL, timestamp, additional-info

Task: Count number of accesses for each domain (inside URL)

MapReduce Example: Web log analysis (2)

Each record: UserID, URL, timestamp, additional-info

Task: Total "value" of accesses for each domain based on additional-info

MapReduce Example: Web log analysis (3)

Each record: UserID, URL, timestamp, additional-info

Task: Find all pairs of UserIDs accessing same URL

MapReduce Example: Web log analysis (4)

Each record: UserID, URL, timestamp, additional-info

Separate records: UserID, name, age, gender, …

Task: Total "value" of accesses for each domain based on user attributes