MapReduce

Kai Zhao

Review

Common Big Data Challenges

Volume:

- Too much data for processing → reduce processing time by using distributed & parallel computing
  - Performing word counting on 300 articles simultaneously
- Solution: **Distributed & Parallel Computing**

Parallel Computing

- Make use of **parallel architectures** (e.g. multi-core, multi-processor, clusters of machines, etc.) to improve computing performance
    - Using multiple cores to speedup video processing (Facetime HD!)
    - Using multiple processors in web servers
    - Using GPUs to increase rendering frame rate in games
    - Using clusters to run simulations
- Have a long history in High Performance Computing
    - Similar to streaming, getting popular in the Big Data era

Python’s Core HOF

- map(): applies a function over an iterable to produce a new iterable

```
map(lambda x: int(x)+1, ['0', '1']) -> [1, 2]
```

- filter(): return only values that satisfy a predicate in the new iterable

```
filter(lambda x: x<1, [0, 1]) -> [0]
```

- reduce(): accumulate a sequence of values from left to right

```
reduce(lambda x, y: x+y, xrange(5), 100) -> 110
```

- sorted(): return a new sorted list using a comparator function

```
sorted(xrange(5))                          -> [0,1,2,3,4]
sorted(xrange(5), cmp=lambda x,y:y-x)      -> [4,3,2,1,0]
```

Outline

- MapReduce
- Designing MapReduce Algorithms

MapReduce

Why is MapReduce?

- By definition, big data is too large to handle by conventional means. Sooner or later, you just can't scale up anymore
  - machines cannot grow too large...
- But we can add more computers!
- MapReduce paradigm allows us to scale out - even with commodity hardware
  - first proposed in the big data world by Google Jeffrey Dean and Sanjay Ghemawat.
  - MapReduce: Simplified Data Processing on Large Clusters. 2004.

What is MapReduce?

- A programming paradigm (for big data processing)
  - Data is split into distributable chunks
  - Perform a series of transformations on those chunks
  - Transformations are run in parallel on the chunks (data parallelism)
- MapReduce is scalable by adding more machines to process chunks
  - scaling out on commodity hardware
- The foundation for Hadoop, which is an implementation of MapReduce

What is MapReduce?

- A programming paradigm that process data in 2 phases/operations: map(), and then reduce()
  - instead of map(), filter(), reduce(), sort() like in the previous class, we are limited to only map() and reduce()
- In a nutshell — that's it:
  - Provide a data collection (separable records)
  - Apply a user-defined map function on each data record
  - Then reduce the mapped output with another user-defined function

What is MapReduce?

- MapReduce works on (key,value) pairs

INPUT: list of key-value pairs of (k1,v1)

MAP: (k1,v1) ➔ [list of (k2,v2)]

SHUFFLE: combine (k2,v2) ➔ (k2, [list of v2])

REDUCE: (k2, [list of v2]) ➔ (k3,v3)

OUTPUT: list of (k3,v3)

Fixed pipeline

What is MapReduce?

- MapReduce works on (key,value) pairs

INPUT: list of key-value pairs of (k1,v1)

MAP: (k1,v1) ➔ [list of (k2,v2)]

SHUFFLE: combine (k2,v2) ➔ (k2, [list of v2])

REDUCE: (k2, [list of v2]) ➔ (k3,v3)

OUTPUT: list of (k3,v3)

User-defined functions executed in parallel

Input

- Data must be separable into records
  - Lines of text
  - Rows of tables
  - CSV: yes — JSON, XML: no
- Key/value pairs
  - Key = line number, record index
  - Value = text string, row data
- Keys are mostly ignored in many cases (e.g. just passing text lines)

Map Phase

- Transform each input record using a user-defined function
  - **(k1,v1) ➔ [(k2,v2),...]** — could be an empty list
- one-to-many relationship
  - vs. one-to-one of Python's map() higher order function
  - Python's filter() is part of the Map phase
- Mainly process by streaming data chunks
  - **[(k1,v1)] ➔ [(k2,v2),...]**
  - generators ➔ generators

Shuffle and Sort Phase

- For each key **k2**, collect all **v2** from outputs of all map processes — shuffling into **(k2, [v2])**
  - Done by distributed partitioning and local sort
- Outputs are **(k2, [v2])** being distributed to “reducers”
- **[(k2,v2)] ➔ (k2, [v2])**
- **(k2,v2)** are intermediate key/value pairs

Reduce Phase

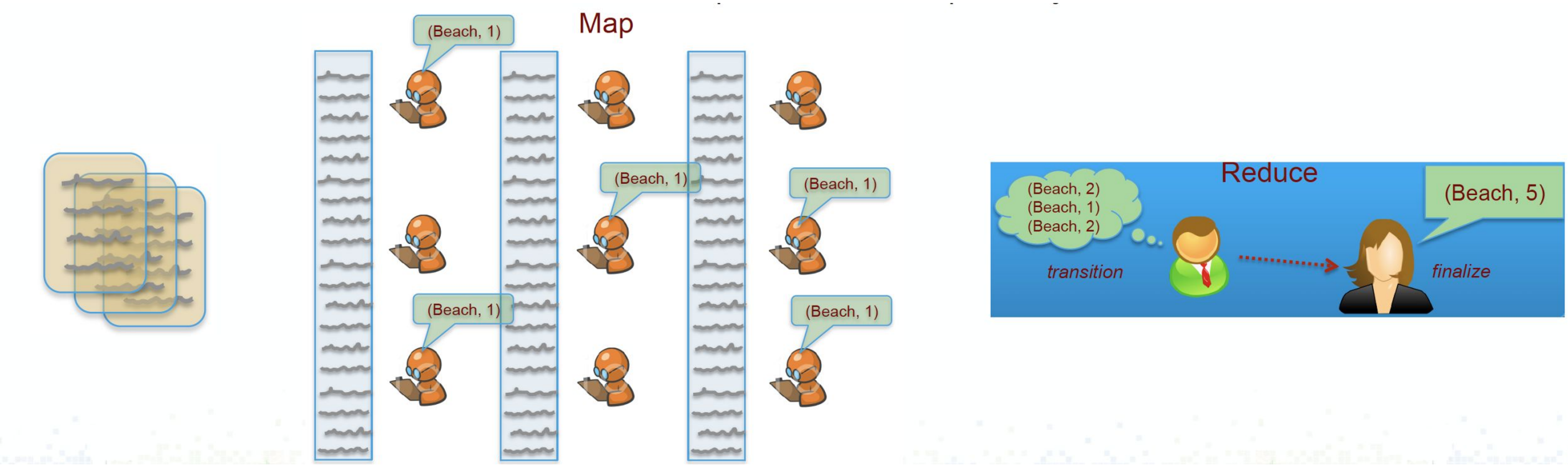
- Transform the output of the Shuffle phase using a user-defined function
  - **(k2,[v2]) ➔ (k3, v3)**
- Equivalent to reduceByKey() ~ groupByKey() x reduce()
  - vs. Python's reduce()
- • Also process in streaming data chunks
  - **[(k2, [v2])] ➔ [(k3, v3)]**
- This phase is optional, can be omitted.

The Classic Example: WordCount

- The “Hello World” of MapReduce
- Given a set of documents, compute the frequency of each word
- INPUT (k1, v1): (document: line number, text line)
- OUTPUT (k3, v3): (word, frequency)
- INTERMEDIATE (k2, v2): (word, 1)
    - Transform each line to a list of words and their frequencies (=1)
    - Combine all tuples by words, and add all the frequency together

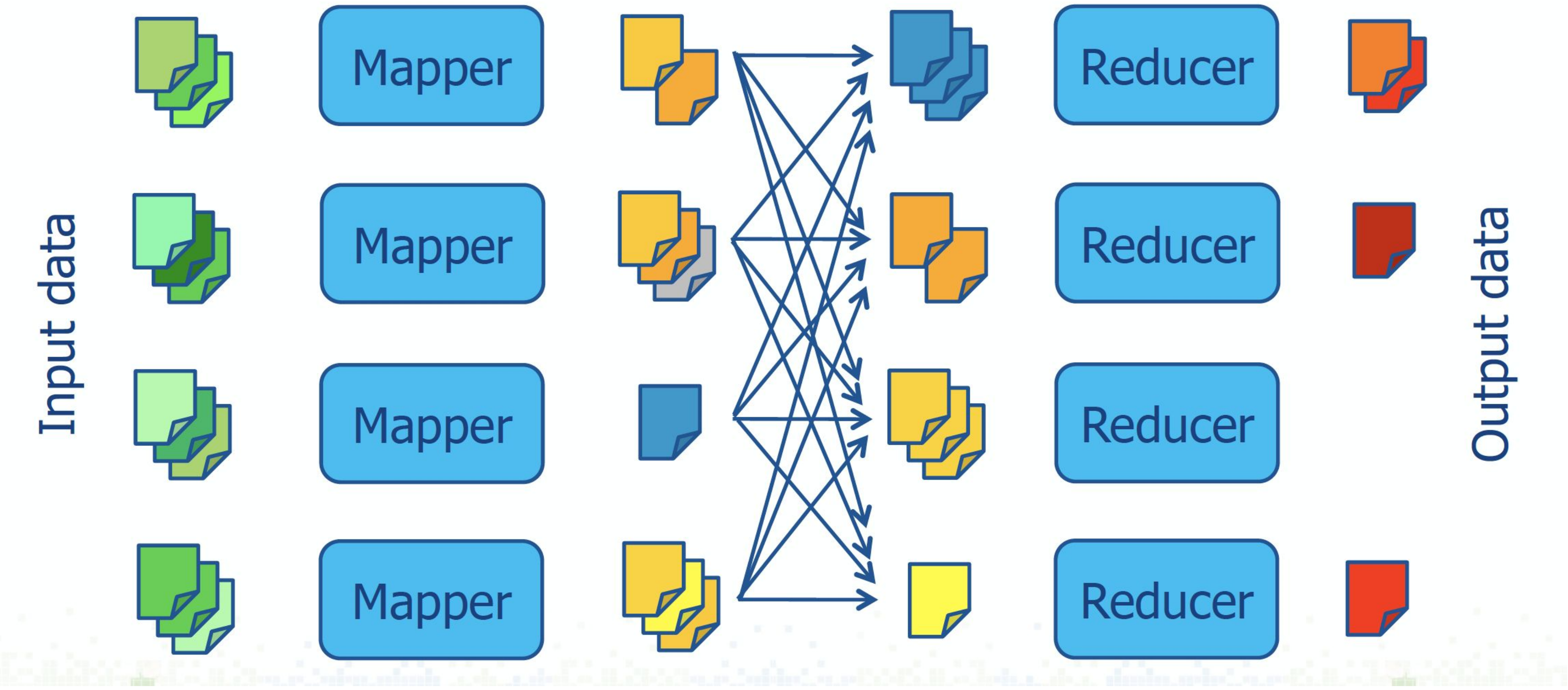
The Classic Example: WordCount

- The “Hello World” of MapReduce
- Given a set of documents, compute the frequency of each word



MapReduce Dataflow

Intermediate (key,value) pairs



The Shuffle

When to use MapReduce?

- Data parallelism problems (“embarrassingly parallel”)
  - Word count
  - Distributed grep
  - Reverse index
  - Document OCR (Optical character recognition)
- Data must be splittable into chunks and records

Designing MapReduce Algorithms

Designing MapReduce Algorithms

- Key decision: What should be done by map, and what by reduce?
  - map can do something to each individual key-value pair, but it can't look at other key-value pairs
- Example: Filtering out key-value pairs we don't need
  - map can emit more than one intermediate key-value pair for each incoming key-value pair
- Example: Incoming data is text, map produces (word,1) for each word

Designing MapReduce Algorithms

- Key decision: What should be done by map, and what by reduce?
  - reduce can aggregate data; it can look at multiple values, as long as map has mapped them to the same (intermediate) key
  - Example: Count the number of words, add up the total cost, ...
- Need to get the intermediate format right!
  - If reduce needs to look at several values together, map must emit them using the same key!

Filtering with MapReduce

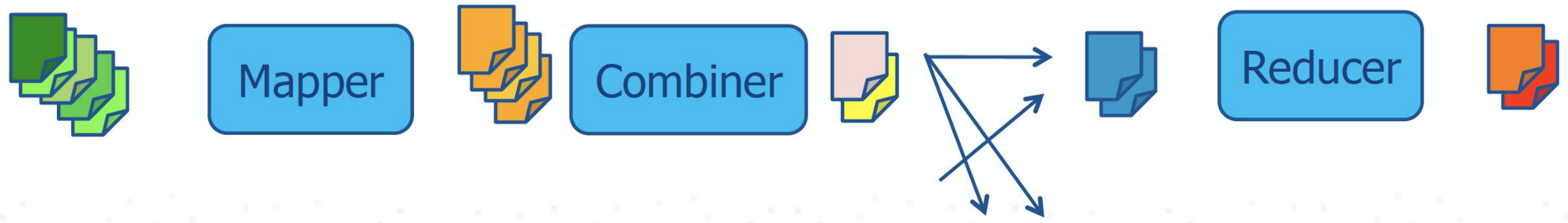
- Goal: find lines/files/tuples with a particular characteristic
- Examples:
  - grep Web logs for requests to a specific domain
  - find in the Web logs the hostnames accessed by a particular IP
  - locate all the files that contain the words 'Apple' and 'Jobs'
- Generally: map does most of the work, utilizing one-to-many relationships to discard unqualified records. reduce may simply be the identity

Aggregation with MapReduce

- Goal: compute the maximum, the sum, the average, ..., over a set of values
- Examples:
  - Count the number of requests to a website
  - Find the most popular domain
  - Average the number of requests per page per Web site
- Often: map may be simple or the identity, reduce does the aggregation

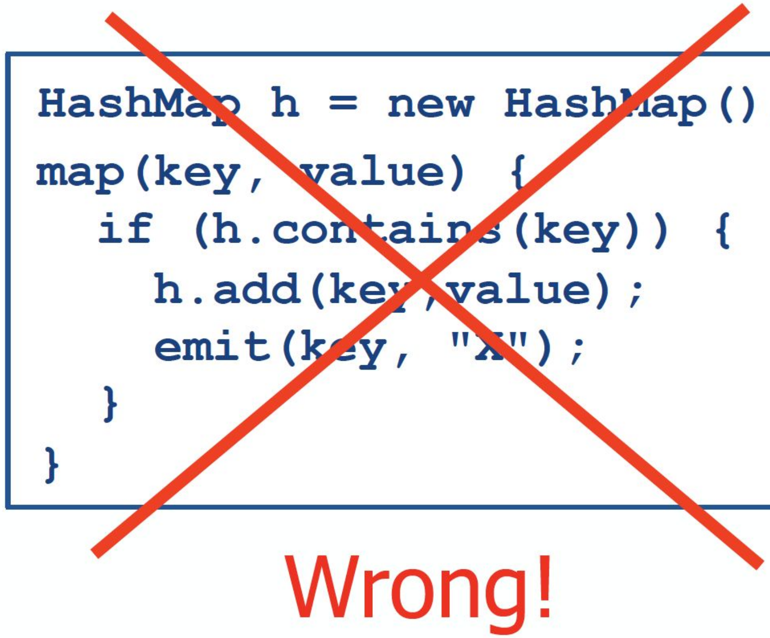
Combiners

- Certain functions can be decomposed into partial steps:
  - Can take counts of two sub-partitions, sum them up to get a complete count for the partition
  - Can take maxes of two sub-partitions, max them to get a complete max for the partition
- Multiple map jobs on the same machine may write to the same reduce key



Common Mistakes to Avoid

- Mapper and reducer should be stateless
-
- Don't use static variables - after map +
- reduce return, they should remember
- nothing about the processed data!
-
- Reason: No guarantees about which
- key-value pairs will be processed by
- which workers!



Common Mistakes to Avoid

- Mapper must not map too much data to the same key
- In particular, don't map everything to the same key!! Otherwise the reduce worker will be overwhelmed!
- It's okay if some reduce workers have more work than others

```
map(key, value) {
  emit("FOO", key + " " + value);
}
```

Wrong!