Introduction to Cloud Computing

MapReduce Algorithms

15-319, spring 2010

18th Lecture, March 18th

Majd F. Sakr

Lecture Goals

- Look At Examples of Algorithms Implemented in MapReduce
- Understand methods of designing algorithms in MapReduce

Lecture Outline

- **Algorithms**
  - Sorting
  - Searching
  - Indexing
- **Design Strategies**

Thinking in MapReduce

- **As with functional programming, a change in mindset is required when developing algorithms for MapReduce**
- **We focus on data and not instructions**
  - Express your computation in terms of maps and reduces
- **Hadoop programs represent Data flow rather than a procedure.**
- **Take advantage of implicit operations in MapReduce to do difficult tasks**

Sorting in MapReduce

- **Remember that MapReduce has a sort/shuffle stage after the map operation and before the reduce operation.**
- **The Shuffle stage sorts map outputs by key and sends it to the reducers**
- **Preparing the Input**
  - Set of Files with one value per line
  - Mapper key is file name, line number
  - Mapper value is the contents of the line
- **Write pseudocode (including the mapper and reducer code) to sort n values stored in a text file, one value per line.**
  - Try It!

Sorting Algorithm

- **Take advantage of the reducer property – (key,value) pairs are sorted by key before sending to the individual reducers.**
- **Mapper: Use a mapper that transforms each value to a key**
  - Map(k,v) -> (v,_)
- **Reducer: Identity Reducer**
  - Reduce (k’,_) -> (k’,””)
  - You may use the reducer to handle duplicates/erronous input pairs as well.

Sorting with a global order

- **(key,value) pairs inside a reducer are sorted by key for a particular reducer. There is only a local order**
- **For a globally ordered sort either:**
  - Use one reducer (slow)
  - Or choose a correct hash function in the partitioner such that **`K1 < K2 => hash(K1) < hash (K2)`**

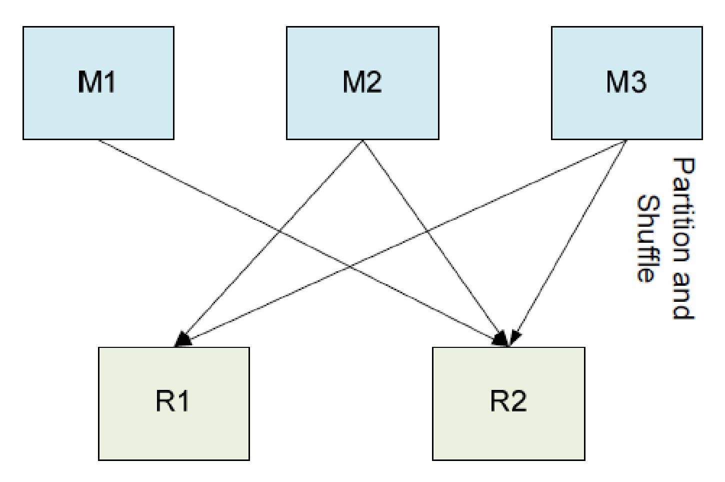

Figure © Cloudera, Inc.

Program Structure in Hadoop

```
Set input files, key,value types
Set output files, key,value types
Set Mapper -> map(k,v) -> (v,_)
Set Reducer -> reduce(k’,_) -> (k’, “”)
Set HashPartitioner to appropriate hash function
Run Job!
```

Sorting : Conclusions

- **One of the easiest things to do in Hadoop! The framework takes care of everything for you.**
- **You specify how the input looks like, and how you want the output format to be.**
- **Also one of the fastest distributed sorting algorithms – a 910 node Yahoo! cluster won the TeraSort challenge by sorting a terabyte of data in 209 seconds.**
  - 1800 maps and 1800 reduce operations
  - Extra optimizations to reduce the intermediate writes to disk
  - Packaged in Hadoop as the TeraSort example.
- **Bottleneck in Distributed Sorting: I/O**
  - How fast can you move data around?

Searching in MapReduce

- **Pop quiz**
  - You care given a **file** with text and you would like to search for a particular **pattern**. Given the file and pattern, write pseudocode to a MapReduce algorithm that outputs each instance of **pattern** in **file**

Searching in MapReduce : Algorithm

- **Input**
  - A set of files containing lines of text
  - A search pattern to find
- **Mapper Input**
  - Mapper key is file name, line number
  - Mapper value is the contents of the line
  - Search pattern is sent as special parameter to the mapper
- **Algorithm**
  - Mapper
    - Given (filename, some text) and “pattern”, if “text” matches “pattern” output (filename, _)
  - Reducer:
    - Identity function

Searching Program Structure

```
Set input files, key,value types
Set output files, key,value types
Send pattern as a special argument to the
  mappter
Set Mapper -> map(k,v)=
     if(v matches pattern)
          output(filename,_);
Set Reducer -> Identity Reducer

Run Job!
```

Search Algorithm : Observations

- **We effectively implement parallel search using the MapReduce Framework**
- **We use the Identity Reducer**
- **Optimization: If we find a file to be interesting, we need to emit it only once.**
  - Use a combiner to fold multiple hits in a file to a single (key, value) pair
  - You can reduce network bandwidth and get a much faster search engine

Indexing

- **Indexing is a common operation in web search engines**
  - Build in index of words from a set of documents and the files they belong to
  - A sample indexing project is given to you to build the index of Shakespeare’s work
- **Makes it much easier to locate a particular item/document based on its content**
- **Inverted Indexing: Map each word to the name of the file it appears in**

Inputs and Algorithm

- **Inputs:**
  - Set of files containing lines of text
  - Mapper Key: Filename, line number
  - Mapper Value: Contents of the Line
- **Algorithm:**
  - Mapper: For each word in (file,words) map to (word,file)
  - Reducer: Identity Reducer

Indexing Data Flow

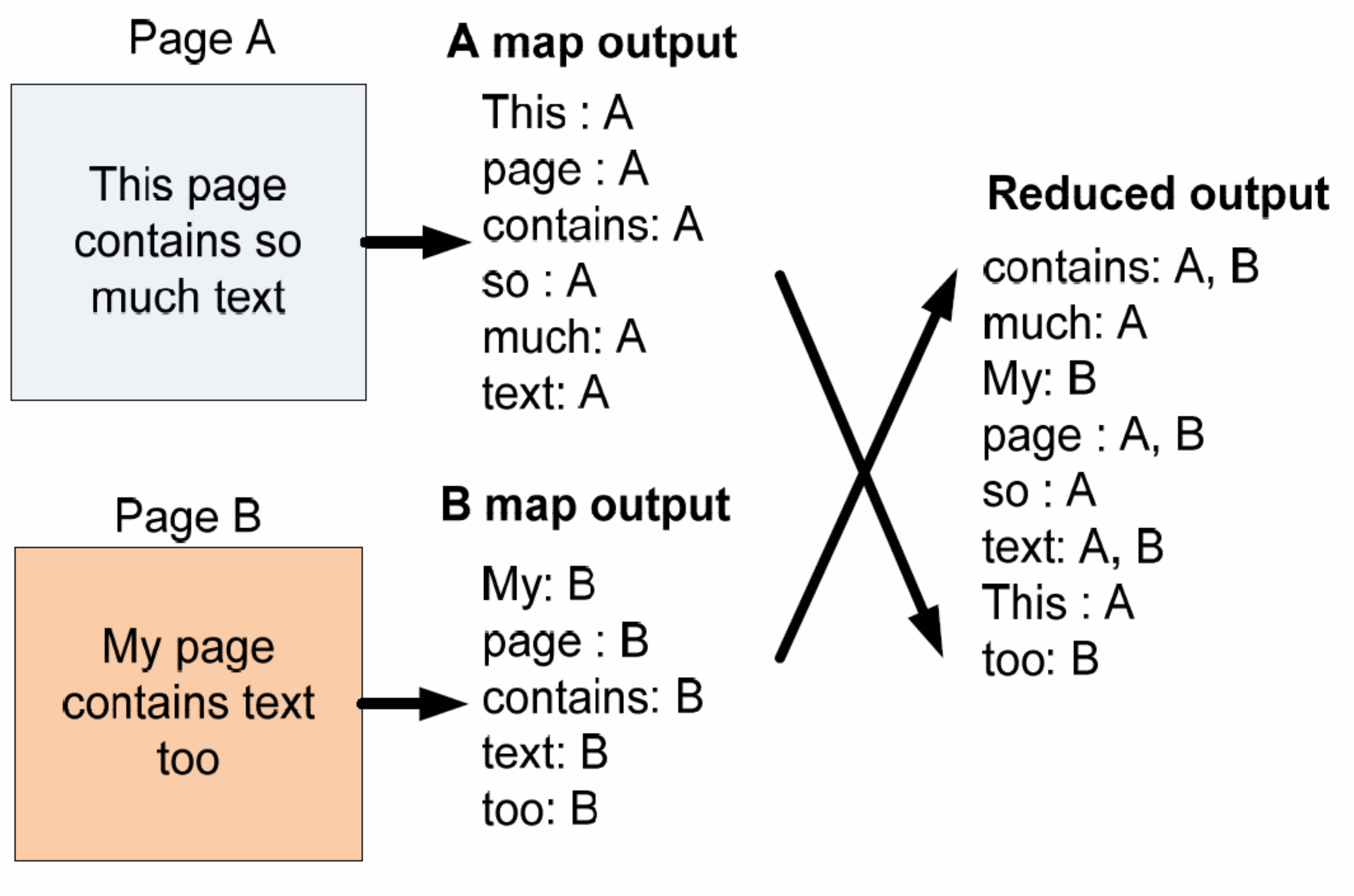

Figure © Cloudera, Inc.

Inverted Index Program Structure

```
Set input files, key,value types
Set output files, key,value types

Set map(pageName,pageText)=
    foreach word w in pageText:
      emit(w,pageName)

Set Reducer -> reduce(word,values)=
  foreach pageName in values
      AddToOutputList(pageName)
  emit(PageListForWord)

Run Job!
```

The MapReduce Library

- **You don't need to write commonly used mappers and reducers yourself all the time**
- **Hadoop Includes commonly used mapper and reducer implementations**
  - `TokenCounterMapper` – Emits (string token, 1) for each token in a line
  - `InverseMapper` – Swaps keys and values
  - `RegexMapper` – Looks for a match with a Regular expression in the input value, emits the match with count 1
  - `IntSumReducer, LongSumReducer` – Does sum operation on list of values belonging to one key

Program Design Strategies I

- **Express your computation in terms of simple maps and reduces**
  - The format of your data and the Key/Value pairs are very important and will essentially dictate your program code
- **Use Multiple Jobs if necessary**
- **Take advantage of the framework to do key tasks like sorting/shuffling**
- **Reuse Mapper and Reducer code and make simple modifications to suit your application**
  - Eg: Inverted Index algorithm is very similar to wordcount, you can reuse/modify existing mappers/reducers

Program Design Strategies II

- **Synchronization Issues**
  - You get one opportunity to synchronize across a single job during the shuffle/sort stage
  - Beyond this, your mappers and reducers run in isolation
    - No opportunity to re-order the execution
    - Deciding where each mapper/reducer runs etc.
- **For more complex problems**
  - Multiple Jobs
  - Create your own data structures to be used as input key/value pairs
  - Advanced MapReduce Techniques
    - Use State inside mappers/reducers
    - Control Sorting Order for intermediate and key space partitioning