Map/Reduce Problem Solving

Sharma Chakravarthy
Information Technology Laboratory
Computer Science and Engineering Department
The University of Texas at Arlington, Arlington, TX 76009
Email: sharma@cse.uta.edu
URL: http://itlab.uta.edu/sharma

Acknowledgements

- These slides are put together from a variety of sources (both papers and slides/tutorials available on the web)

- Mostly I have tried to: provide my perspective, emphasize aspects that are of interest to this course, and have tried to put forth a consolidated view of Map/Reduce

Approach to M/R problem solving

1. Do not start writing code when you are given a problem to solve using map/reduce (e.g., project 3) or even a PL
2. Try to understand the problem in terms of M/R (or PL) framework

   a. Come up with an algorithm FIRST
3. Think of all three levels we have discussed for problem at hand
   a. Do you have 1 input or multiple inputs?
   b. Do you need any other data/file/info in addition to input?
   c. Can you solve it in one m/r job or do you need more than 1? And why?
   d. Is this an iterative m/r job with one or more jobs in each iteration
   e. Is there anything to be done in between iterations?

Approach to M/R problem solving (2)

1. Understand the input and what should happen at each step under your control (input, output)
   a. Map code
   b. Combiner code (if needed)
   c. Reducer code
2. Understand what happens in between your steps and how that contributes to the overall solution?
   a. In fact, the above has to be part of your overall solution!
   b. Your solution will not be correct without the whole thing working correctly!
3. What should be done in each part of your code and why?
4. I am not even considering optimization here!

Concrete steps to M/R problem solving

1. What is the input, in terms of key value?
2. What should be done in the mapper?
   - Specify key value output for each input record
3. Is a combiner needed? If so, what should it do?
   - How does it transform the key, value pair produced by the mapper?
   - What should it produce as key, value pair?
4. How many reducers are needed? Based on what?
5. Should we use default partitioning?
   - Default partitioning is done on the key values produced by the mapper. # partitions is based on the number of specified reducers.
6. Should we provide a custom partitioning?
   - Why? If so, what should it be? How do you decide that?
7. What is received by the reducer? How do we process the key, value-list for each key produced in the mapper
   - This needs a good understanding of what happens during shuffle!

Application 1: Word Count

This is the “Hello World” equivalent!

A simple Map/Reduce program can be written to determine count of words in a set of files. For example, if we had:

foo.txt: Sweet, this is the foo file, sweet

bar.txt: This is the bar file, very sweet

We would expect the output to be:

sweet	3
this	2
is	2
the	2
foo	1
bar	1
file	2
Very	1

Word Count example

- Input: Large number of text documents
- Task: Compute word count across all the documents
- Solution
  - Mapper:
    - Input: A line from the text
    - Output: For every word in a document, output (word (as key), "1“ (as value))
    - For the word “hello”, output <“hello”, “1”>
  - Mapper will generate several similar outputs
  - Reducer: (after all mapper tasks are done)
    - What comes to the reducer is <key, value-list>
    - As many of the above as the number of unique words (why?)
    - <“hello”, [“1”, “1”, ….. “1”] and so on
    - For each <key, value-list>, reduce code can count the number of elements in the list and output <“Hello”, “54”>

Word Count Solution (in Java)

```
//Pseudo-code for "word counting"
map(String key, String value):
    // key: document name,
    // value: document contents
        for each word w in value:
          EmitIntermediate(w, "1");

reduce(String key, Iterator values):
// key: a word
// values: a list of "1"s
    int word_count = 0;
    for each v in values:
        word_count += ParseInt(v);
    Emit(key, AsString(word_count)); //No types, just strings
```

Analysis

- Is this a good word count implementation?
- If not, why?
- Let us take a closer look at the mapper output
- Mapper produces repeating key, value pairs
  - <“hello”, “1”>, <“the”, “1”>, <“hello”, “1”> <“the”, “1”>
  - When mapper sorts on key and groups, you get
    - <“hello”, [“1”, “1”, …]> say 35 in the list
    - <“the”, [“1”, “1”, …]> say 44 in the list
    - …
  - This list from (EACH) mapper is sent to the reducer
  - Reducer merges and applies the previous algorithm
- Is this a good implementation?
  - Will a combiner be beneficial? How?

An Optimization: The Combiner

- A combiner is a local aggregation function for repeated keys produced by same mapper
- For associative operations, such as sum, count, and max, reducer can be used as a combiner for local aggregation
- Decreases size of intermediate data shuffled!
- For the word Count problem,
  - <“hello”, [“1”, “1”, …]> with 35 “1”s can be reduced to
  - <“hello”, “35”> and
  - <“the”, [“1”, “1”, …]> with 44 “1”s can be reduced to
  - <“hello”., “44”>

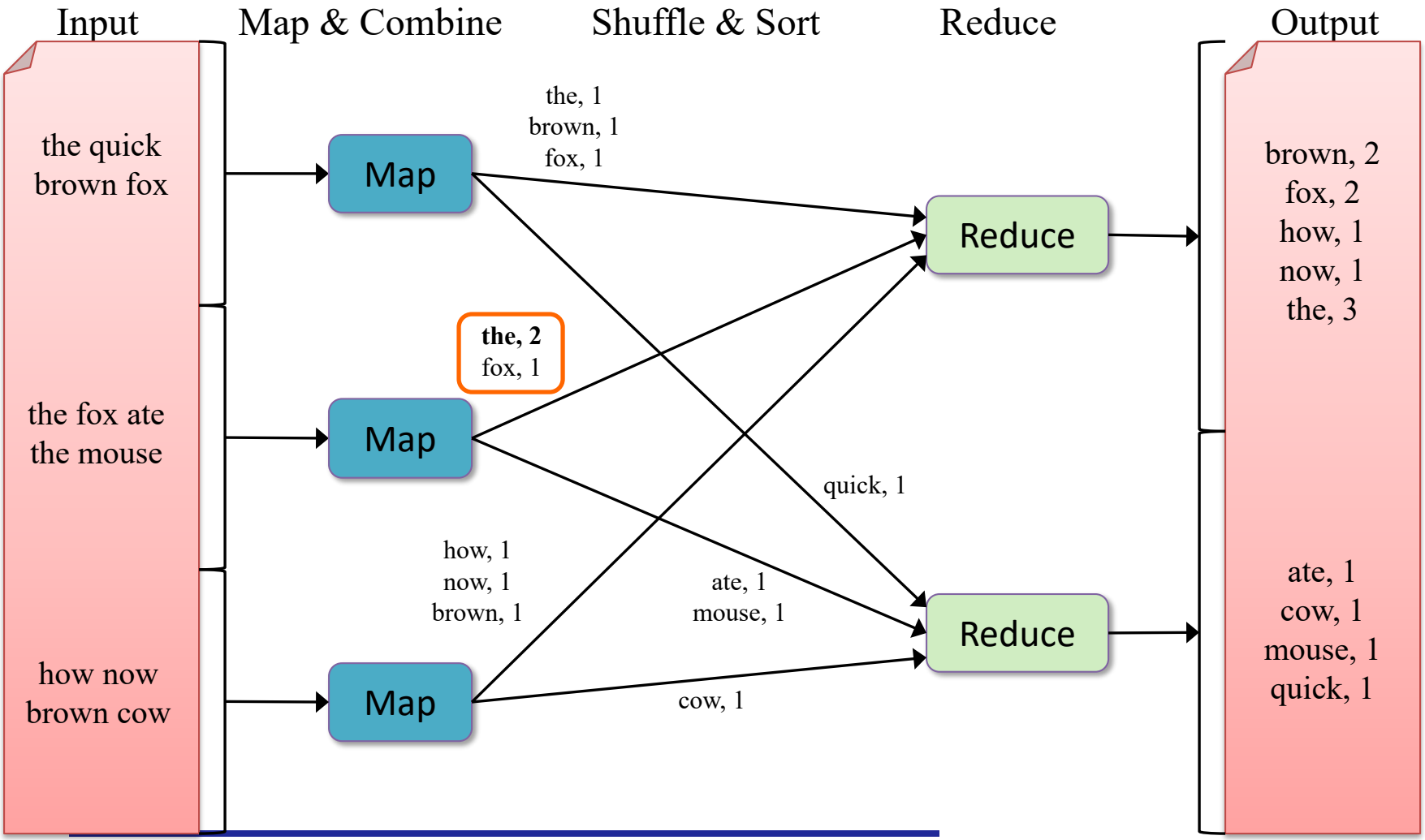
Combiner

- Word count is a prime example where a Combiner is useful. The Word Count program emits a (*word*, 1) pair for every instance of every word it sees. So if the same document contains the word "cat" 300 times, the pair ("cat", 1) is emitted three hundred times; all of these are then sent to the Reducer.
- By using a Combiner, these can be condensed into a single <"cat", “300”> pair to be sent to the Reducer. Now each node only sends a single value (a string of 3 chars instead of hundreds of chars) to the reducer for each word
  - drastically reducing the total bandwidth required for the **shuffle process**, and speeding up the job.

Combiner

- The best part is that we may not need to write any additional code to take advantage of this!
- If a reduce function is both *commutative* and *associative*, then it can be used as a Combiner as well.
- The Combiner should be an instance of the *Reducer* interface. If your Reducer itself cannot be used directly as a Combiner because of commutativity or associativity, you might still be able to write a third class to use as a Combiner for your job.
- Commutative means A o B is the same as B o A
- Associative means (A o B) o C is the same as A o (B o C)
- Count is both commutative and associative (as are +, max)
- -, avg are not associative, concat is not commutative!
  - Keep this in mind when using reduce code for combiner code!

Word Count with Combiner



Homework 1

- Write a Map/Reduce program that **counts the words in each document separately**. Assume, document name is the key and all words are values as input. For example, if we had:

foo.txt: sweet this this is very sweet

bar.txt: this sweet bar is very very sweet

- The output should be:
  - Foo.txt, sweet 2
  - Foo.txt, this 2
  - Foo.txt, is 1
  - Foo.txt, very 1
  - bar.txt, this 1
  - bar.txt, sweet 2
  - bar.txt, bar 1
  - bar.txt, is 1
  - bar.txt, very 2

Word frequency example

- Used for decrypting, comparison of documents etc.

- Input: Large number of text documents
- Task: Compute word frequency (ratio) across all the document
  - Need to compute total count as well as individual count

- You need the total count before you can compute total word count of each word in order to compute frequency

- This cannot be done with a single map reduce task without additional support (why?)

Word frequency example

- A naive solution with basic Map/Reduce model requires two Map/Reduce jobs
  - Map/Reduce job 1: count number of all words in these documents
    - Use combiners
    - The reducer just needs to sum up all word counts
  - Map/Reducer job 2:
    - count number of each word as we did in the previous example
    - The reducer, in addition to the input, reads the total word count and uses for the denominator in every reducer for generating word frequency
    - Alternatively, the Map/Reducer job 1 can generate word count for each word and total word count and map/reduce job 2 can use them for computing word frequency
- Requires the output of one Map/Reduce job to be fed into another Map/Reduce job
  - Chaining of M/R jobs

Word frequency example

- Can we do better?
- Two nice features of Google's MapReduce implementation can help!
  - Ordering guarantee of reduce key
  - Auxiliary functionality: EmitToAllReducers(k, v)
- A nice trick: To compute the total number of words in all documents
  - Every map task sends its total world count with key "“ to ALL reducers
  - Key "" will be the first key processed by reducer (due to ordering guarantee)
    - Sum of its values → total number of words!
- Requires only 1 map and 1 reduce instead of 2 chained map/reduce jobs

Word frequency solution: Mapper with combiner

```
map(String key, String value):
// key: document name, value: document contents
    int word_count = 0;
    for each word w in value:
        EmitIntermediate(w, "1");
        word_count++;
    EmitIntermediateToAllReducers("", AsString(word_count));
combine(String key, Iterator values):
// Combiner for map output
// key: a word, values: a list of counts
   int partial_word_count = 0;
   for each v in values:
      partial_word_count += ParseInt(v);
      Emit(key, AsString(partial_word_count));
```

Word frequency solution: reducer

```
reduce(String key, Iterator values):
// Actual reducer, key: a word
// values: a list of counts
    if (is_first_key):
        assert("" == key); // sanity check
        total_word_count = 0;
        for each v in values:
            total_word_count += ParseInt(v)
    else
        assert("" != key); // sanity check
        int word_count = 0;
        for each v in values:
            word_count += ParseInt(v);
    Emit(key, AsString(word_count /
total_word_count));
```

Join using Map/Reduce

- An important and expensive operation in RDBMSs
  - Supported in SQL (transparent to user)
- Joins two input relations on attribute
- Consider equi-join on 2 attributes

Table 1: (SSN, {name; town; state})

123456:(John Smith;Sunnyvale, CA)

123457:(Jane Brown;Mountain View, CA)

123458:(Tom Little;Mountain View, CA)

Table 2: (SSN, {year, income})

123456:(2007,$70000),(2006,$65000),(2005,$6000),...

123457:(2007,$72000),(2006,$70000),(2005,$6000),...

123458:(2007,$80000),(2006,$85000),(2005,$7500),...

- Task: Compute average income in each city in 2007 or each city for each year
- Relations are not ordered!

Analysis of the problem

- Can this be done with ONE map/reduce job?
- If so, what do you need at the reducer?
- Can that be emitted/generated by the mapper?
- In order to compute the average, you need somethings like
  - <city2007, [list of incomes]>
  - <city2008, [list of incomes]>
  - Study the inputs to figure this out!
- If you look at the input, it should be evident that it is not possible
  - Because city info is not there in table 1
- Then look for what can be output from mapper and how it can be used in a second map/reduce job for solving the problem

Computing averages

Mapper 1a

Input: SSN → personal info

emit: <ssn, city>

Mapper 1b:

input: SSN → annual incomes

emit: <SSN, 2007 income>

reducer 1:

input: SSN, {city, 2007 income}

Output: (SSN, [City, 2007 income])

Mapper 2:

Input: SSN → [city, 2007 income]

Output: (City, 2007 income)

reducer 2:

Input: City → 2007 incomes

output: (City, AVG (2007 Incomes))

Reducer-side Join using Map/Reduce

- Note that join requires one map/reduce task
- Average computation requires another
- Each input (table) need to be processed separately by a mapper
  - Cannot run the *same* mapper code in all mappers
- This example is a simple one (why?)
  - The join attribute is unique!
  - Also, equality join
- Why does that matter?
  - If not, the input to the reducer is more complicated
  - It will NOT be a pair of values
  - Tagging is required to differentiate
- How about non equi-joins?

Doing Join using Map/Reduce

- There are two approaches
  - Mapper-side join
  - Requires only mapper
  - Uses Hash join
- Reducer-side join
  - requires one map/reduce task
  - Mapper separates tuples on join key attributes
  - Tagging tuples may be needed
    - Depends on whether the join attribute is unique or not
  - Typically used for equi-join
  - Does not use hash join directly, but relies on the hashing during shuffling
- Non equi-joins are more complicated!

non-key Join example

sid	sname	rating	age
22	dustin	7	45
28	yuppy	9	35
31	lubber	8	55
44	guppy	5	35
58	rusty	10	35

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

sid	sname	age	bid
28	Yuppy	35	103
28	yuppy	35	103
31	lubber	55	101
31	Lubber	55	102
31	Lubber	55	101
58	rusty	35	103

[22, <r1,dustin,45>] [28, <r2,103>]
[28, <r1,yuppy,35>] [28, <r2,103>]
[31, <r1,lubber,55>] [31, <r2,101>]

At reducer:

[22, {<r1,dustin,45>}]
[28, {<r1,yuppy,35>,<r2,103>,<r2,103>}]
[31, {<r1,lubber,25>,<r2,101>, …, …}]

Another Example

Suppose, we have the following input (can be any delimiter separated)

Id	Name	Age	Gender	Salary
1201	gopal	45	Male	50,000
1202	manisha	40	Female	51,000
1203	khalil	34	Male	30,000
1204	prasanth	30	Male	30,000
1205	kiran	20	Male	40,000
1206	laxmi	25	Female	35,000
1207	bhavya	20	Female	15,000
1208	reshma	19	Female	14,000
1209	kranthi	22	Male	22,000
1210	Satish	24	Male	25,000
1211	Krishna	25	Male	25,000
1212	Arshad	28	Male	20,000
1213	lavanya	18	Female	8,000

Desired output:

age range up to 20

Female 15000

Male 40000

age range between 21 and 30

Female 35000

Male 31000

age range above 30

Female 51000

Male 50000

- **We have to compute highest salaried employee by gender in different age groups: up to 20, between 21 to 30, and above 30**
- **Essentially compute Histograms for Male and Female separately**

Approach (steps to follow)

- What should be done in the mapper? How many mappers?
  - Key: gender
  - Value: entire record (or needed portions)
- Do we need a combiner?
  - For this problem we **do not** need a combiner! (why?)
- Should we use the default partitioning?
  - Default partitioning will partition the key, namely, gender. We also want partitioning on age for histogram
- Should we provide a custom partitioning?
  - Yes, but on what?
  - Read the key value pair. Access the age and create p partitions (3 in our case) based on the age groups given (if 5 age groups, need 5 partitions)
- What should be done in the reducer?
  - How to choose the number of reducers?

Mapper

- For this input record
  - 1201 gopal 45 Male 50,000
  - Mapper emits <“Male”, “45, 50,000”>
  - Custom partitioner will take this as input and generate
    - <maleAbove30, “50000”>
- Similarly, for this input record
  - 1208 reshma 19 Female 14,000
  - mapper emits <“Female”, “19, 14000”>
  - Custom partitioner will take this as input and generate
    - <FemaleUnder20, “14000”>
- Mapper will sort and group all newly generates <kay, value> into
  - <“maleabove30”, value-list> and <“femaleUnder20”, value-list>

Histogram details

- The mapper output key space was only 2
- If default partitioning is used, we ONLY get 2 partitions as male and female
- But, we need to compute histograms which requires age
- There is NO combiner
- Hence a custom partition need to be used
- Custom partitioner takes mapper emits and creates as many partitions as needed based on the age range. They bare sorted and grouped to get
  - <MaleAbove30, “50000”, “30000”, “30000”>
  - <MaleUnder20, “40000”, …>
  - <MaleBetween, …>
- These are shuffled (sent to reducers -- can be 1 or 2 or 3 in this example)
- No combiner in this example

Reducer

- With the partitioning, max number of reducers that can be used?
  - 3
- Each partition is on age range and can be sent to a different reducer
- How do the key value pairs look like in each partition?
  - Partition 0:
    - <FemaleUnder20 [15000, 14000, 8000]>
    - MaleUnder20, [15000]>
  - Partition 1:
    - <FemaleBetween21-30, [35000]>
    - <MaleBetween21-30, [25000, …]> total 5
  - Partition 2: similar

Approach

- What does the reducer do?
  - Reducer 0 takes
    - <FemaleUnder20 [15000, 14000, 8000]> and computes highest value in the list
      - <Female Under 20, 15,000> as output
    - <Male under 20, 40000 > as output
  - Compute the highest for each <key, list> coming to that reducer!
    - Female Bhavya, 15,000; Male, Kiran, 40,000
  - Reducer 1:
    - Female, Laxmi, 25, female, 35000
    - Male Satish, 24, male, 25000
  - Reducer 2:
  - You can see the code for this example at https://www.tutorialspoint.com/map_reduce/map_reduce_partitioner.htm

Approach

- Final output will be in 3 files each generated by a reducer.
  - Reducer 0:
    - Female <(bhavya, 20, female, 15000), {reshma, 19, female, 14000)
      (lavanya, 18, female, 8000)>
    - Male <(kiran, 20, male, 40000)>
- Output in Part-00000 (age range up to 20)
  - Female 15000
  - Male 40000
- Output in Part-00001 (age range between 20 and 30)
  - Female 35000
  - Male 31000
- Output in Part-00002 (age range above 30)
  - Female 51000
  - Male 50000

Homework

- Think of an alternative approach to do this problem using map/reduce
- Assume mapper uses age range for generating its key, value output
- Could be a test problem

Summary

- Additional features such as pipes and streaming are available in Hadoop.
- If you are familiar with C++ or Java, it is not very difficult to understand the basic concept and use it
- Of course, if you want to use advanced features, you need to learn them
- Much easier than using a DBMS for some jobs where the data is in free format; will discuss more of this later!

Questions !