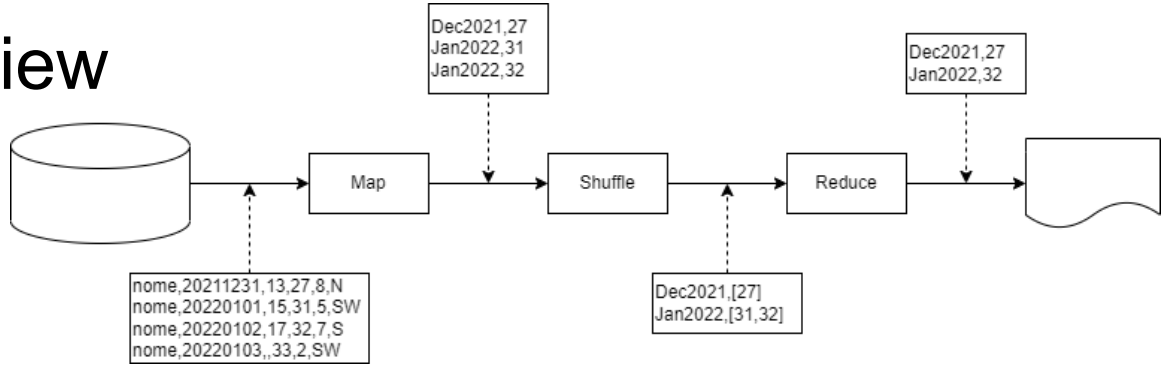
CS4650 Topic 4: Map-Reduce

Analyzing Large Datasets

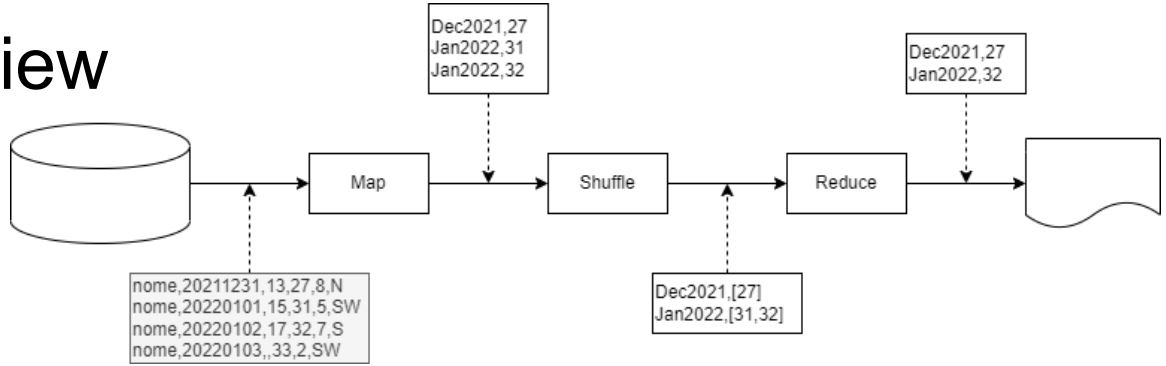
- Suppose we have a very large dataset, and we want to ask questions like:
  - What is the largest temperature during each month of each year?
  - What is the average amount of time spent during homework per class?
  - What is the total sales per product per month?
  - What is the average and maximum life expectancy for people of a given height?
- Further suppose that the dataset is too large to fit on one computer?
  - Use HDFS, for example
- How do we extract the relevant information from the dataset?
  - Map
- How do we distill the information to answer the questions?
  - Reduce
- Today we will look at the MapReduce tool

MapReduce Overview



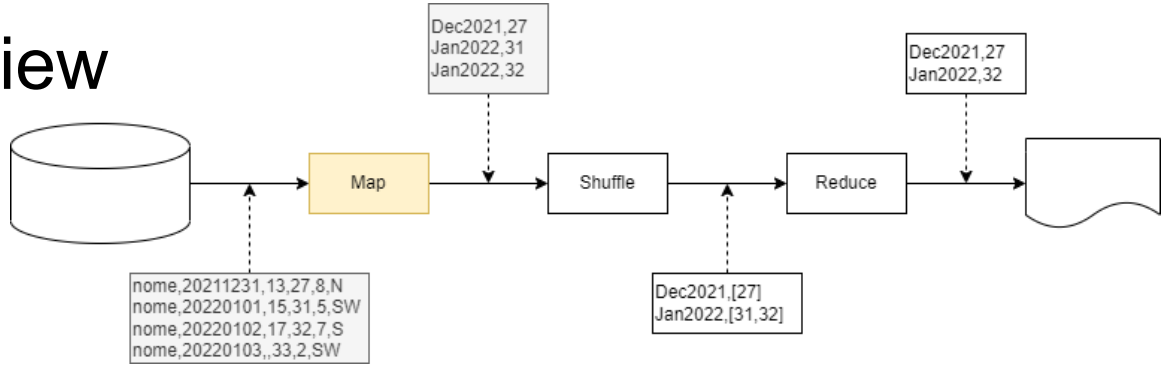
- This diagram shows the data flow through the MapReduce System:
  - The data is filtered and cleaned up by the Map program
  - The data is collated by the Shuffle program
  - The data is condensed by the Reduce program
- There may be additional pieces in the system

MapReduce Overview



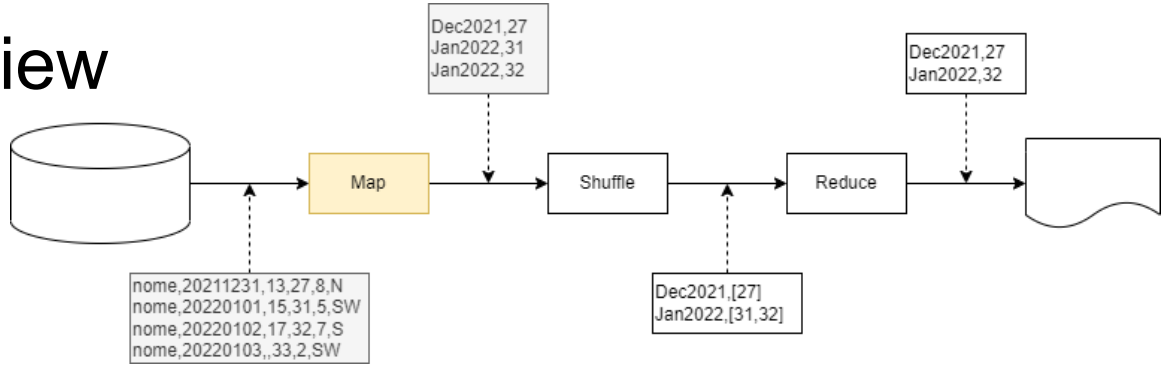
- This example considers weather data collected from various automated weather stations (the diagram only shows data from Nome).
- Each line of text in the dataset represents the values for one station, one day.
- The line gives the station name, the date, the lowest recorded temperature, the highest recorded temperature, the wind speed, and wind direction.
- This dataset uses the CSV (*comma separated values*) representation.
- In our final report, we want the highest recorded temperature for each month.

MapReduce Overview



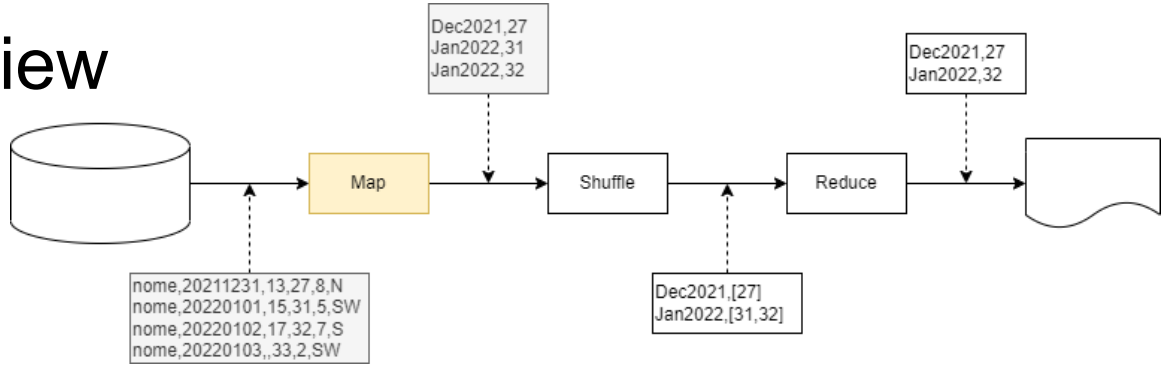
- The *Map* program extracts the relevant information from the dataset.
- Map considers each line, one at a time.
- If a line has valid data, a corresponding line is written to the output file.
- If a line has invalid data, or has missing data, the Map program might either attempt to correct the data, or it may simply discard the line.
    - In this example, the last line is missing one of the temperature values, so the Map program discards that line.

MapReduce Overview



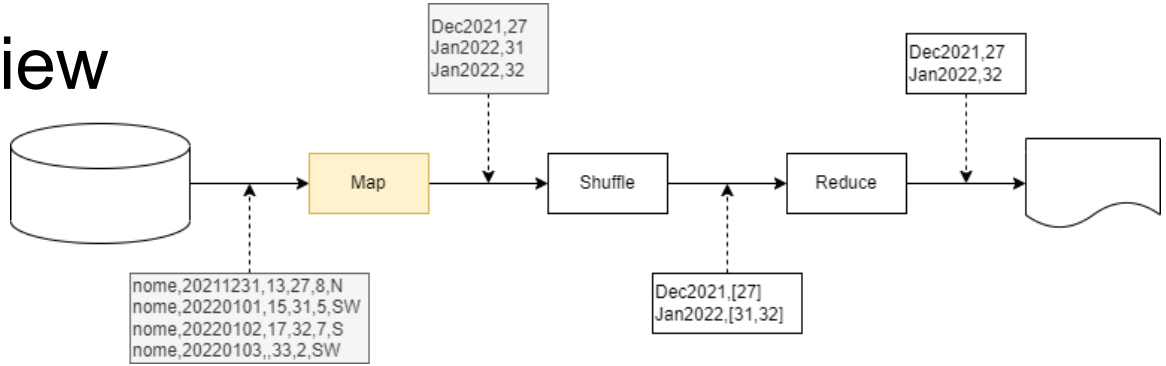
- The Map program also discards all of the irrelevant data. For this application, Map discards the location, the low temperature, the wind direction and wind speed.
- All that we need is the date and the highest temperature reading.

MapReduce Overview



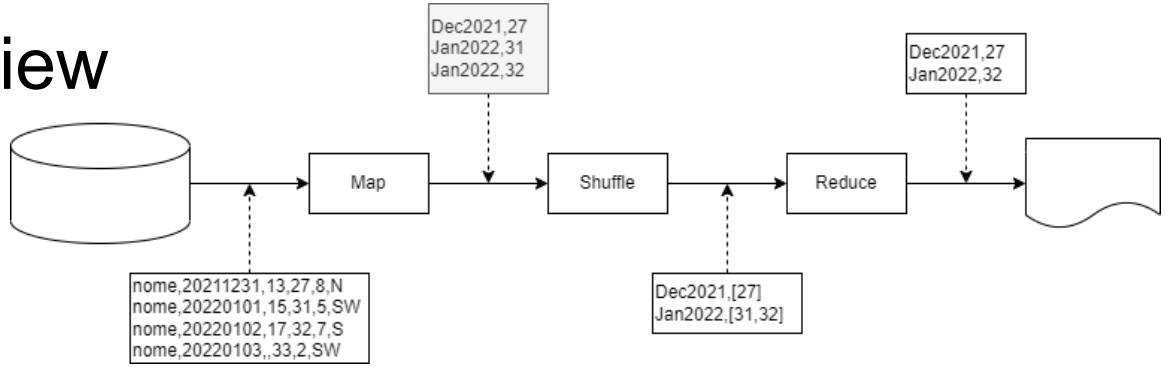
- The Map program might convert one or more value into a more useful representation.
- In this case, we convert the date from an integer of the form *yyyymmdd* to a 3-character month name followed by the year, but the day is discarded.

MapReduce Overview



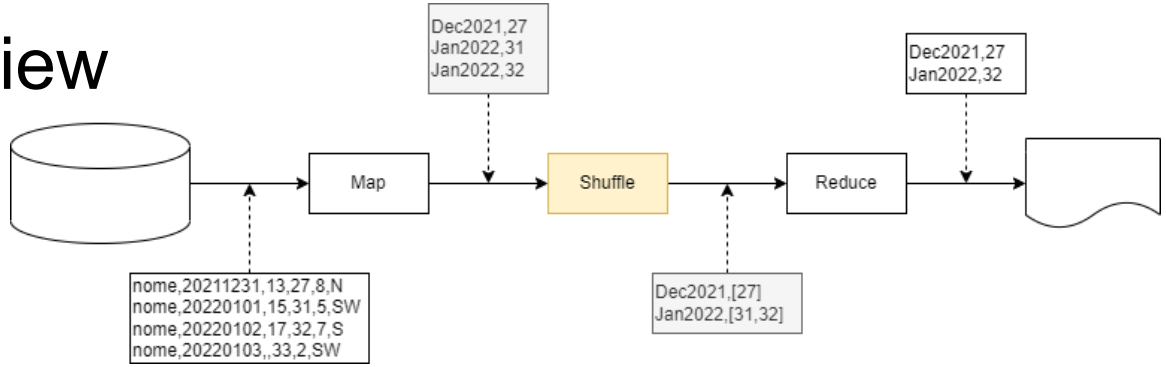
- You may wonder how the Map program knows how to do all of this processing.
- The answer is actually quite simple: Map is a program that you write to solve this particular problem.

MapReduce Overview



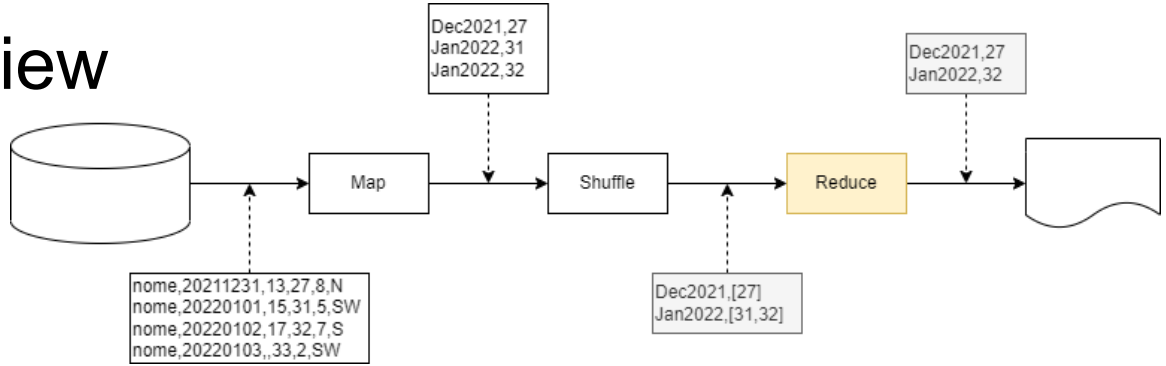
- Note in the output of the Map there are two lines that have the same date: Jan2022. It is OK to have duplicate lines at this point, and in fact this is expected.
- Each line of the output has two quantities:
  - The first is the key. For this example, the date is the key, and we are going to compute a resulting value for each key.
  - The second is the value. For this example, the value is a single number. For other examples, other types of data, or even compositions of data can be used as the value.

MapReduce Overview



- The *Shuffle* program sorts all of the lines of the Map output data, gathering all of the values associated with each key.
- In the Shuffle output file, there is one line for each key. The line gives the key, then gives an array of all of the values that were associated with that key.
- Unlike the Map program, the same Shuffle program is used for each MapReduce.

MapReduce Overview



- The last program in the ensemble is the *Reduce* program.
- This program considers each line of the Shuffle output, one at a time.
- It looks at all the values that have been given for that key, then performs the desired computation.
  - For this example, it keeps the largest of the values.
- The Reduce output file contains one line for each key, giving the final result for that key.
- Yes, you will also write the Reduce program for your application.

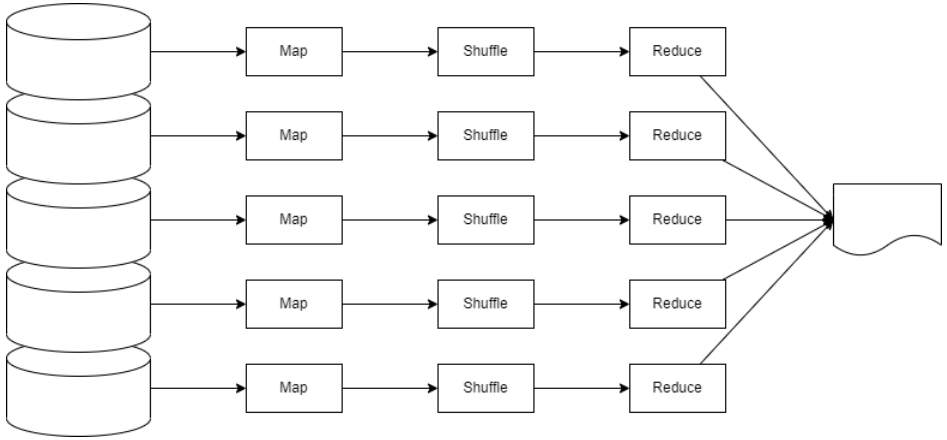
Small Data vs Big Data

- The Map/Shuffle/Reduce collection works great with small data (data that fits on one computer).
- For really big data sets, this gets to be problematic!
  - The mapper has to work on a lot of data
  - The shuffle has to work on a lot of data
  - The reducer has to work on a lot of data
- Since this data does not fit on one machine, if the mapper is on one machine, then all of the data must be sent to that machine for the mapper to do its thing.
- This gets worse for Shuffle, since it has to merge all of the data, if this is too much for one machine, what can Shuffle do?

Divide and Conquer

- One technique that is frequently used in Computer Science is *divide and conquer*: A large problem is divided into smaller problems, each of those smaller problems is solved, then the results are merged to produce the final result.
- With HDFS, the dataset is naturally divided and distributed to multiple machines.
- Perhaps we can run Map/Shuffle/Reduce on these multiple machines.

Divide and Conquer

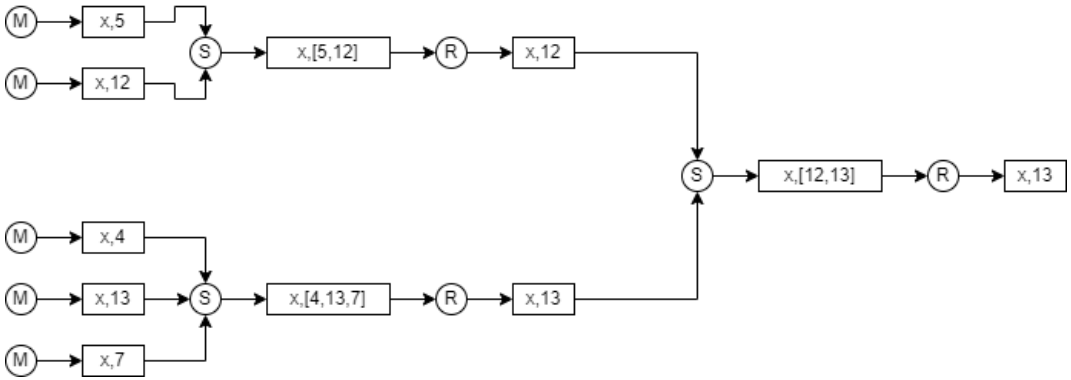


- Here we show the data distributed over five machines. Each machine runs Map, Shuffle, and Reduce, then the final results are combined to produce the final result.
- This mostly works, but partly doesn't work.
  - The Map program works one line at a time, so it can easily be split and run in parallel.
  - The final Reduce wants to operate on the final data, as it has to consider data from across the dataset, so it cannot inherently be run in parallel.
  - Shuffle has the same problem as Reduce.

The Problem With Reduce: Largest

- The problem with Reduce may be compounded because of the algorithm being computed.
- Consider the case of keeping the largest value for any key:
  - The source data for one key might be spread across multiple machines.
  - The Reduce on each of those machines will produce an output line for that key.
    - This result will be the largest value for the key *as far as this machine knows, for its part of the data.*
    - In other words, this is a *partial result*.
  - If we do a straight merge of the files, there will be multiple lines for that key, with all of the partial results from each of the machines.
  - We could run another Shuffle/Reduce operation on that machine to clean up the duplicates.
  - This would work for an algorithm like 'maximum' or 'sum' or 'count', where the partial result has the same representation as the final result.

The Problem With Reduce: Largest

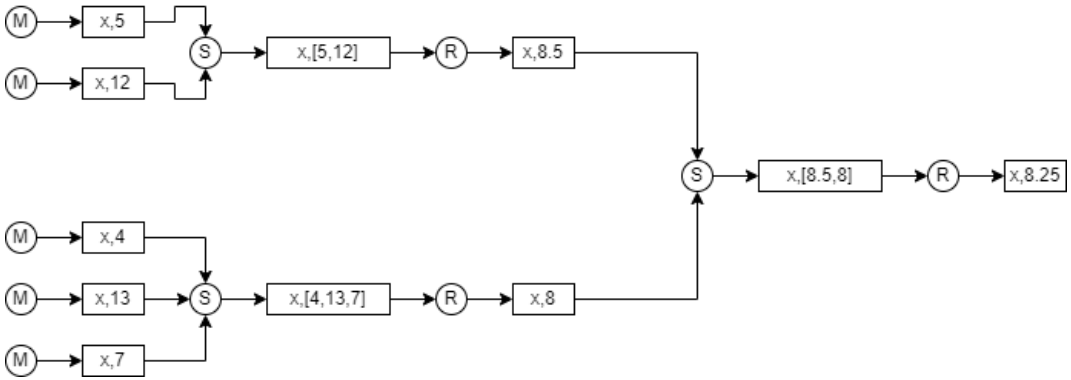


- For computing the largest, each machine computes its own result, an *intermediate result*.
- A final machine merges all of these results, using the same Shuffle/Reduce code, to produce the final result.

The Problem With Reduce: Average

- Suppose our computation was to determine the *average value* for each key.
- We could try the same approach to compute the final result.
- *FYI, for the values 5, 12, 4, 13, and 7, the average is 8.2.*

The Problem With Reduce: Average

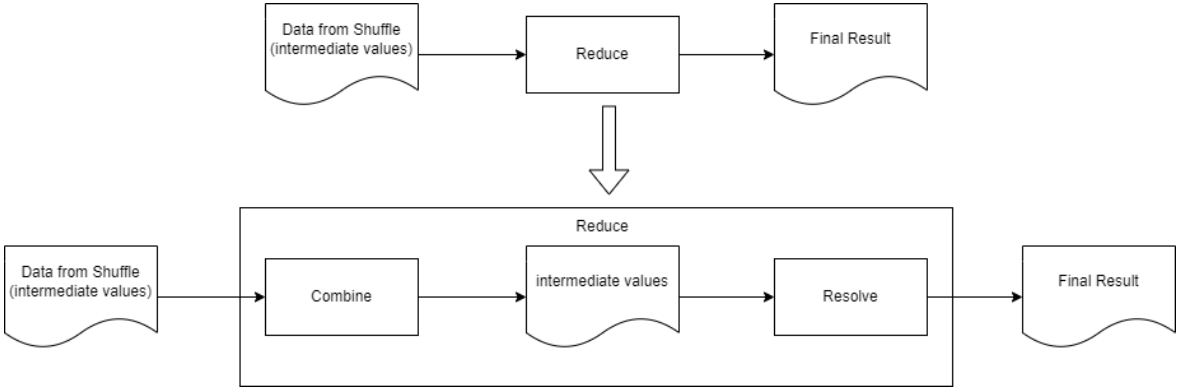


- This computes the average as 8.25, but the actual average is 8.2.
- The results would be even more off if, for example, there was only one line in the top machine but four lines in the bottom machine. *Try it!*

Computing the Average

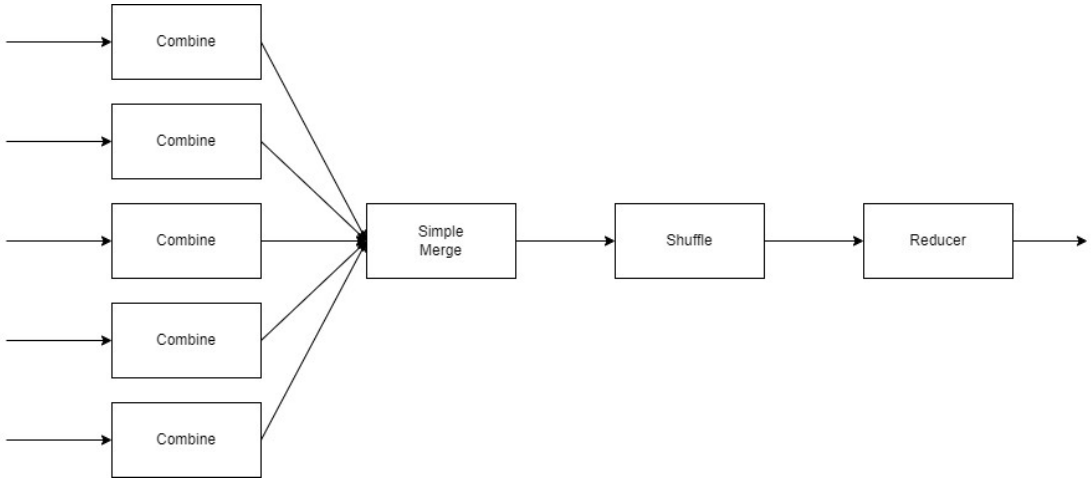
- The problem is that for the *Average* computation, the output of Reduce is not a good intermediate representation for partial results.
- To correctly compute *Average*, we want two intermediate numbers, the sum of the values and the number of the values.
- When we are done, the result is sum / number. So the final output of Reduce divides the total sum by the total number.
- But it would be OK to distribute the computation of the sum and distribute the computation of the number.

Computing the Average



- This suggests that we can conceptually split the Reduce operation into a *Combine* operation (which computes the partial, intermediate results), and a *Resolve* operation (which computes the final result).

Computing the Average



- For the *Average* calculation, we then have a new function, a *Combine* routine.
- We can even build a tree of Combine functions, if we split the process among many computers.

Intermediate Values

- We are close to wrapping up all of these details.
- For the *Largest* calculation, our 'intermediate representation' is simply a number, the largest found so far. In addition, our Combine function is identical to our Reduce function.
- For the *Average* calculation, our 'intermediate representation' will be a pair of numbers, [sum, count]. And our Combine function is similar to, but not identical to, the Reduce function.

Computing the Average

- The Map function will therefore generate the following line for each valid input line:

  key, [value, 1]

- The *key* is the key, as before.
- The *value* is actually a pair of numbers, the sum so far and the count so far. The sum for each individual line is simply the value for that line, and the count is 1.

Datatypes

- Since the Mapper and Reducer are custom programs for the specific application, we should document what the various values are that each application requires as input and provides as output.
- The Map function will output using our intermediate representation.
- The Reduce function will take as input the Shuffled version of the intermediate representation (where the value associated with each key is a *list of the intermediate representations*).
- The Combine function will take the same input as the Reduce function and produce the same output as the Map function.
- *Let's look at some examples!*

What is the largest temperature during each month of each year?

- Our first question was computing the largest temperature during each year.
- Here is an example of one output line from Map:
  "Jan2022,17"
- Here is an example of one input line for Reduce:
  "Jan2022,[17,12,15]"
- Here is an example of one output line from Reduce:
  "Jan2022,17"
- The Combine function is identical to the Reduce function.

What is the average amount of time spent during homework per class?

- We are computing an average, so our intermediate form needs a sum and a count.
- A typical output line from Map and from Combine would be:
  "CS4650,[2.5,1]"
- A typical input line to either Combine or Reduce is
  "CS4650,[[2.5,1],[3.7,2],[1.2,1]]"
- A typical output line from Reduce is
  "CS4650,2.87"

What is the total sales per product per month?

- What would be a good key for this situation?
- Can we use the same form for intermediate results as final results?

What is the total sales per product per month?

- What would be a good key for this situation?

  The key should be a combination of the product identifier (name?),

  month, and year
- Can we use the same form for intermediate results as final results?

  Yes, because sum is like larger, it is associative. So the value is simply the sum so far.

What is the average and maximum life expectancy for people of a given height?

- What would be our key?
- What would the final result look like?
- What would an intermediate result look like?

What is the average and maximum life expectancy for people of a given height?

- What would be our key?

  The key would simply be a height.
- What would the final result look like?

  The final result would be the pair [average,maximum]
- What would an intermediate result look like?

  For the intermediate, we would need the triple [sum,count,maximum]

In Class Exercise

- Consider a large dataset that gives the multiple-choice answer for each question of every student's SAT test. The result we want is, for each question on the SAT, what was the most common answer (If you want, you can assume that all questions have 4 possible answers).
- In your group, consider what the key would look like, what the intermediate result should look like, and write some pseudocode that describes the algorithm for the Reduce and the Combine functions.

Coding the Map and Reduce Programs

- The Map and Reduce functions can be coded in a number of languages.
- Naturally, writing code in some languages is easier than writing code in other languages.
- Conversely, some programs run more efficiently when coded in one language rather than another.
- Therefore, there is a tradeoff to be made!
- We will explore Java and Python implementations of the Max Temperature system.

Coding the Map and Reduce Programs

- Datasets come in a variety of formats.
- To simplify our discussions above, we used CSV format for the dataset.
- In the book, they used a single text string to represent one line of data.
- The text string is about 105 characters long.
- Specific substrings give values of interest:
  - One substring gives the observation date
  - Another substring gives the air temperature
  - One character gives a 'quality code', indicating whether the air temperature value is accurate, missing, probably unreliable and so on.
- To accept a line, the temperature value should not be '9999' (an indicator of a bad value), and the quality should be 0, 1, 4, 5, or 9.

Java Map Boilerplate

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper
        extends Mapper<LongWritable, Text, Text, IntWritable> {
        …
}
```

Java Map Code

```
public class MaxTemperatureMapper
        extends Mapper<LongWritable, Text, Text, IntWritable> {
        private static final int MISSING = 9999;
        @Override
        public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        String line = value.toString();
        String year = line.substring(15, 19);
        int airTemperature;
        if (line.charAt(87) == '+') {
        airTemperature = Integer.parseInt(line.substring(88, 92));
        } else {
        airTemperature = Integer.parseInt(line.substring(87, 92));
        }
        …
```

Java Map Code (cont.)

```
                String quality = line.substring(92, 93);
                if (airTemperature != MISSING && quality.matches("[01459]"))
{
                        context.write(new Text(year), new
IntWritable(airTemperature));
                }
        }
}
```

Java Reduce Boilerplate

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer
        extends Reducer<Text, IntWritable, Text, IntWritable> {
        …
}
```

Java Reduce Code

```
public class MaxTemperatureReducer
        extends Reducer<Text, IntWritable, Text, IntWritable> {
        @Override
        public void reduce(Text key, Iterable<IntWritable> values, Context
context)
                throws IOException, InterruptedException {
                int maxValue = Integer.MIN_VALUE;
                for (IntWritable value : values) {
                        maxValue = Math.max(maxValue, value.get());
                }
                context.write(key, new IntWritable(maxValue));
        }
}
```

Python Map Code

```
#!/usr/bin/env python


import re
import sys


for line in sys.stdin:
        val = line.strip();
        (year, temp, q) = (val[15:19], val[87:92], val[92:93])
        if (temp != "+9999" and re.match("[01459]", q)):
                print "%s\t%s" % (year, temp)
```

Python Reduce Code

```
#!/usr/bin/env python

import sys

(last_key, max_val) = (None, -sys.maxint)
for line in sys.stdin:
        (key, val) = line.strip().split("\t")
        if last_key and last_key != key:
                print "%s\t%s" % (last_key, max_val)
                (last_key, max_val) = (key, int(val))
        else:
                (last_key, max_val) = (key, max(max_val, int(val)))

if last_key:
        print "%s\t%s" % (last_key, max_val)
```

Conclusion

- We now have the pieces.
- How do we control this? How do we make this work?
- We would like to try a small example on our personal computer, to make sure the code works.
- We would then like to scale this up to work for large examples, without rewriting code?
- Who we gonna call?… see you in the next topic!