Lecture 18: MapReduce

Principles of Computer Systems

Spring 2019

Stanford University

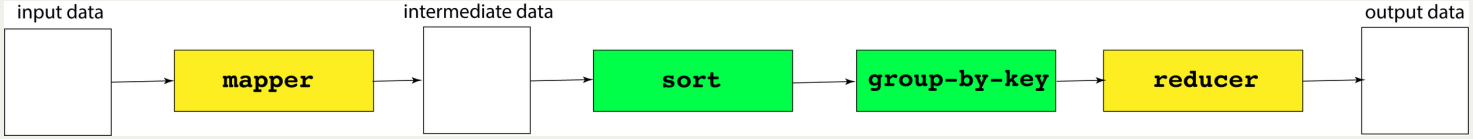
Computer Science Department

Lecturer: Chris Gregg

PDF of this presentation

Lecture 18: MapReduce

- MapReduce is a parallel, distributed programming model and implementation used to process and generate large data sets.
  - The **map** component of a MapReduce job typically parses input data and distills it down to some intermediate result.
  - The **reduce** component of a MapReduce job collates these intermediate results and distills them down even further to the desired output.
  - The pipeline of processes involved in a MapReduce job is captured by the below illustration:



  - The processes shaded in yellow are programs specific to the data set being processed, whereas the processes shaded in green are present in all MapReduce pipelines.
  - We'll invest some energy over the next several slides explaining what a mapper, a reducer, and the group-by-key processes look like.

Lecture 18: MapReduce

- Here is an example of a map executable—written in Python—that reads an input file and outputs a line of the form **<word> 1** for every alphabetic token in that file.

```
import sys
import re

pattern = re.compile("^[a-z]+$") # matches purely alphabetic words
for line in sys.stdin:
    line = line.strip()
    tokens = line.split()
    for token in tokens:
        lowercaseword = token.lower()
        if pattern.match(lowercaseword):
            print '%s 1' % lowercaseword
```

- The above script can be invoked as follows to generate the stream of words in Anna Karenina:

```
myth61:$ cat anna-karenina.txt | ./word-count-mapper.py
happy 1
families 1
are 1
... // some 340000 words omitted for brevity
to 1
put 1
into 1
```

Lecture 18: MapReduce

- **`group-by-key`** contributes to all MapReduce pipelines, not just this one. Our **`group-by-key.py`** executable—presented on the next slide—assumes the mapper's output has been sorted so multiple instances of the same key are more easily grouped together, as with:

```
myth61:$ cat anna-karenina.txt | ./word-count-mapper.py | sort
a 1
a 1
a 1
a 1
a 1 // plus 6064 additional copies of this same line
...
zigzag 1
zoological 1
zoological 1
zoology 1
zu 1
myth61:$ cat anna-karenina.txt | ./word-count-mapper.py | sort | ./group-by-key.py
a 1 1 1 1 1 // plus 6064 more 1's on this same line
...
zeal 1 1 1
zealously 1
zest 1
zhivahov 1
zigzag 1
zoological 1 1
zoology 1
zu 1
```

Lecture 18: MapReduce

- Presented below is a short (but dense) Python script that reads from an incoming stream of key-value pairs, sorted by key, and outputs the same content, save for the fact that all lines with the same key have been merged into a single line, where all values themselves have been collapsed to a single vector-of-values presentation.
  - The implementation relies on some nontrivial features of Python that don't exist in C or C++. Don't worry about the implementation too much, as it's really just here for completeness.
  - Since you know what the overall script does, you can intuit what each line of it must do.

```
from itertools import groupby
from operator import itemgetter
import sys

def read_mapper_output(file):
    for line in file:
        yield line.strip().split(' ')

data = read_mapper_output(sys.stdin)
for key, keygroup in groupby(data, itemgetter(0)):
    values = ' '.join(sorted(v for k, v in keygroup))
    print "%s %s" % (key, values)
```

Lecture 18: MapReduce

- A reducer is a problem-specific program that expects a sorted input file, where each line is a key/vector-of-values pair as might be produced by our **`./group-by-key.py`** script.

```
import sys

def read_mapper_output(file):
    for line in file:
        yield line.strip().split(' ')

for vec in read_mapper_output(sys.stdin):
    word = vec[0]
    count = sum(int(number) for number in vec[1:])
    print "%s %d" % (word, count)
```

- The above reducer could be fed the sorted, key-grouped output of the previously supplied mapper if this chain of piped executables is supplied on the command line:

```
myth61:$ cat anna-karenina.txt | ./word-count-mapper.py | sort \
                               | ./group-by-key.py | ./word-count-reducer.py
a 6069
abandon 6
abandoned 9
abandonment 1
...
zoological 2
zoology 1
zu 1
```

Lecture 18: MapReduce Assignment (last assignment!)

- Now that you know a bit about the MapReduce algorithm, let's chat about your final assignment for the quarter. It will be due on **Wednesday, June 5th at midnight**, but there will not be a late day penalty for handing it in by Thursday at midnight (handing it in Friday will incur the usual 90% max, and Saturday will be the 60% max).
- The assignment implements the MapReduce algorithm using the **`myth`** machines as workers.
- We have given you a robust start to the program, but it will involve understanding a fair amount of code.
- As with the proxy-server assignment, there are four individual tasks -- handle each one in turn for the best flow through the assignment.
- You will be adding a **`ThreadPool`** to the assignment, and you will be fully implementing the "reduce" part of the assignment. Some of the map part of the assignment has been written for you, but you need to do a bit more work there, as well.

Lecture 18: MapReduce Assignment

- All of the output for your assignment will go into a files directory that you create as follows:

```
myth57:$ make directories
```

- You only have to run this once for the assignment. Before you run a test, you should clear this directory, as follows:

```
make filefree
```

- There are *five* executables that will be created when you run **`make`**:

```
myth57:$ find . -maxdepth 1 -type f -executable
./mr
./mrm
./mrr
./word-count-mapper
./word-count-reducer
```

- **`word-count-mapper`** and **`word-count-reducer`** are already written for you. You will modify files that will contribute to **`mr`**, **`mrm`**, and **`mrr`**, (map-reduce, map-reduce-mapper, and map-reduce-reducer, respectively).

Lecture 18: MapReduce Assignment

- There are, as you can imagine, many parameters you can have (number of mappers, number of mappers, number of reducers, paths, etc., so there is a configuration file that you will use. One example is as follows:

```
myth57:$ cat odyssey-full.cfg
mapper word-count-mapper
reducer word-count-reducer
num-mappers 8
num-reducers 4
input-path /usr/class/cs110/samples/assign8/odyssey-full
intermediate-path files/intermediate
output-path files/output
```

- See the assignment details for a description of each line. You can modify the keys and values as you wish to test.
- If you look at the **`samples/odyssey-full`** directory, you will see twelve files, which comprise of the full Odyssey book in twelve parts:

```
myth57:$ ls -lu samples/odyssey-full
total 515
-rw------- 1 poohbear operator 59318 Mar  7 09:31 00001.input
-rw------- 1 poohbear operator 43041 Mar  7 09:31 00002.input
-rw------- 1 poohbear operator 42209 Mar  7 09:31 00003.input
...
-rw------- 1 poohbear operator 42157 Mar  7 09:31 00011.input
-rw------- 1 poohbear operator 41080 Mar  7 09:31 00012.input
```

Lecture 18: MapReduce Assignment

- Let's see an example run with the solution executables:

```
myth57:$ make filefree
rm -fr files/intermediate/* files/output/*

myth57:$ ./samples/mr_soln --mapper ./samples/mrm_soln --reducer ./samples/mrr_soln --config odyssey-full.cfg

Determining which machines in the myth cluster can be used... [done!!]
Mapper executable: word-count-mapper
Reducer executable: word-count-reducer
Number of Mapping Workers: 8
Number of Reducing Workers: 4
Input Path: /usr/class/cs110/samples/assign8/odyssey-full
Intermediate Path: /afs/.ir.stanford.edu/users/c/g/cgregg/cs110/spring-2019/assignments/assign8/files/intermediate
Output Path: /afs/.ir.stanford.edu/users/c/g/cgregg/cs110/spring-2019/assignments/assign8/files/output
Server running on port 48721

Received a connection request from myth59.stanford.edu.
Incoming communication from myth59.stanford.edu on descriptor 6.
Instructing worker at myth59.stanford.edu to process this pattern: "/usr/class/cs110/samples/assign8/odyssey-full/00001.input"
Conversation with myth59.stanford.edu complete.
Received a connection request from myth61.stanford.edu.
Incoming communication from myth61.stanford.edu on descriptor 7.

... LOTS of lines removed

Remote ssh command on myth56 executed and exited with status code 0.
Reduction of all intermediate chunks now complete.
/afs/.ir.stanford.edu/users/c/g/cgregg/cs110/spring-2019/assignments/assign8/files/output/00000.output hashes to 13585898109251157014
/afs/.ir.stanford.edu/users/c/g/cgregg/cs110/spring-2019/assignments/assign8/files/output/00001.output hashes to 1022930401727915107
/afs/.ir.stanford.edu/users/c/g/cgregg/cs110/spring-2019/assignments/assign8/files/output/00002.output hashes to 9942936493001557706
/afs/.ir.stanford.edu/users/c/g/cgregg/cs110/spring-2019/assignments/assign8/files/output/00003.output hashes to 5127170323801202206

... more lines removed
```

- There is a plethora of communication between the machine we run on and the other myths.
- Output ends up in the **`files/`** directory.

Lecture 18: MapReduce Assignment

- The map phase of **`mr`** has the 8 mappers (from the .cfg file) process the 12 files processed by **`word-count-mapper`** and put the results into **`files/intermediate`**:

```
myth57:$ ls -lu files/intermediate/
total 858
-rw------- 1 cgregg operator 2279 May 29 09:29 00001.00000.mapped
-rw------- 1 cgregg operator 1448 May 29 09:29 00001.00001.mapped
-rw------- 1 cgregg operator 1927 May 29 09:29 00001.00002.mapped
-rw------- 1 cgregg operator 2776 May 29 09:29 00001.00003.mapped
-rw------- 1 cgregg operator 1071 May 29 09:29 00001.00004.mapped
...lots removed
-rw------- 1 cgregg operator  968 May 29 09:29 00012.00027.mapped
-rw------- 1 cgregg operator 1720 May 29 09:29 00012.00028.mapped
-rw------- 1 cgregg operator 1686 May 29 09:29 00012.00029.mapped
-rw------- 1 cgregg operator 2930 May 29 09:29 00012.00030.mapped
-rw------- 1 cgregg operator 2355 May 29 09:29 00012.00031.mapped
```

- If we look at **`00012.00028`**, we see:

```
myth57:$ head -10 files/intermediate/00012.00028.mapped
thee 1
rest 1
thee 1
woes 1
knows 1
grieve 1
sire 1
laertes 1
sire 1
power 1
```

- This file represents the words in **`00012.input`** that hashed to 28 modulo 32 (because we have 8 mappers * 4 reducers)
- Note that some words will appear multiple times (e.g., "thee")

Lecture 18: MapReduce Assignment

- If we look at `00005.00028`, we can also see "thee" again:

```
myth57:$ head -10 files/intermediate/00005.00028.mapped
vain 1
must 1
strand 1
cry 1
herself 1
she 1
along 1
head 1
dayreflection 1
thee 1
```

- This makes sense because "thee" also occurs in file `00005.input` (these files are not reduced yet!)
- "thee" hashes to 28 modulo 32, so it will end up in any of the `.00028` files if occurs in the input that produced that file.
- To test a word with a hash, you can run the hasher program, located here.

```
myth57:$ ./hasher thee 32
28
```

Lecture 18: MapReduce Assignment

- Let's test the starter code:

```
myth57:~$ make directories filefree
// make command listings removed for brevity
myth57:~$ make
// make command listings removed for brevity
myth57:~$ ./mr --mapper ./mrm --reducer ./mrr --config odyssey-full.cfg --map-only --quiet
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00001.mapped hashes to 2579744460591809953
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00002.mapped hashes to 15803262022774104844
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00003.mapped hashes to 15899354350090661280
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00004.mapped hashes to 15307244185057831752
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00005.mapped hashes to 13459647136135605867
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00006.mapped hashes to 2960163283726752270
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00007.mapped hashes to 3717115895887543972
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00008.mapped hashes to 8824063684278310934
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00009.mapped hashes to 673568360187010420
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00010.mapped hashes to 9867662168026348720
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00011.mapped hashes to 5390329291543335432
/afs/ir.stanford.edu/users/c/g/cgregg/assign8/files/intermediate/00012.mapped hashes to 13755032733372518054
myth57:~$
```

- If we look in **`files/intermediate`**, we see files without the secondary split:

```
myth57:~$ $ ls -l files/intermediate/
total 655
-rw------- 1 cgregg operator 76280 May 29 10:26 00001.mapped
-rw------- 1 cgregg operator 54704 May 29 10:26 00002.mapped
-rw------- 1 cgregg operator 53732 May 29 10:26 00003.mapped
-rw------- 1 cgregg operator 53246 May 29 10:26 00004.mapped
-rw------- 1 cgregg operator 53693 May 29 10:26 00005.mapped
-rw------- 1 cgregg operator 53182 May 29 10:26 00006.mapped
-rw------- 1 cgregg operator 54404 May 29 10:26 00007.mapped
-rw------- 1 cgregg operator 53464 May 29 10:26 00008.mapped
-rw------- 1 cgregg operator 53143 May 29 10:26 00009.mapped
-rw------- 1 cgregg operator 53325 May 29 10:26 00010.mapped
-rw------- 1 cgregg operator 53790 May 29 10:26 00011.mapped
-rw------- 1 cgregg operator 52207 May 29 10:26 00012.mapped
```

- It turns out that "thee" is only in 11 of the 12 files:

```
$ grep -l "^thee " files/intermediate/*.mapped \
          | wc -l
11
```

- also, **`files/output`** is empty:

```
myth57:$ ls -l files/output/
total 0
```

Lecture 18: MapReduce Assignment

- **Task 1**: Read and understand these files:
  - **`mrm.cc`**
  - **`mapreduce-worker.h/cc`**
  - **`mapreduce-mapper.h/cc`**
  - **`mr-messages.h/cc`**
  - **`mapreduce-server.h/cc`**
- **`mrm.cc`** is the entry point, and is the client of the MapReduce server. The server invokes the program remotely (i.e., on a different **`myth`** machine), and it then reaches out to the server for an input file to process, notifies the server when it has succeeded (or failed), and also sends progress messages back to the server.
- The reason this works across the **`myth`**s is because of the shared AFS file system -- each separate computer has the same files.
- **`mapreduce-mapper`** provides a class, **`MapReduceMapper`**, that keeps track of the hostname and virtual pid of the server, and it relies on a custom protocol for the communication.
- **`MapReduceMapper`** *subclasses* another class, **`MapReduceWorker`**, which also is a base class for **`MapReduceReducer`** .
- **`map-reduce-server.cc`** provides a single mapper, and launches a server to respond to the mapper.

Lecture 18: MapReduce Assignment

- **Task 2:** Spawn multiple mappers
- How do you span multiple mappers?
  - You use a **`ThreadPool`**!
  - You'll need to modify **`spawnMappers`** so it will create **`num-mappers`** of them.
  - You should add a **`ThreadPool`** to **`MapReduceServer`**. There is already a method called **`orchestrateWorkers`** which uses a **`handleRequest`** method, which you can wrap into a **`.schedule`** call.
  - Make sure you provide enough **`mutex`** support so that the threads don't corrupt each other.

Lecture 18: MapReduce Assignment

- **Task 3**: Hashing keys and creating multiple intermediate files
  - At this point, your program still only creates the **`000xx.mapped`** files.
  - You will need to use the **`hash<string>`** class (see the **`hasher`** example for usage) to hash each key.
  - You will need to update **`buildMapperCommand`** to add another argument, which will be the number of hash codes used by each mapper when generating the intermediate files for each input file.
  - You will also need to update **`mrm.cc`** to accept another argument in its **`argv`**, and this means you'll need to modify the **`MapReducerMapper`** constructor likewise.
  - The number of hash codes should be the number of mappers times the number of reducers (this is arbitrary, but helps to surface concurrency issues).

Lecture 18: MapReduce Assignment

- **Task 4**: Implement **`spawnReducers`**
  - This is an open-ended part of the assignment -- if you understand the purpose of MapReduce, it should be clear what needs to get done.
    - Each reducer needs to collate the collection of intermediate files of keys with the same hash code, sort it, and then group the sorted collation by key, then invoke the reducer executable to produce output files.
    - You may use the python programs located at **`/usr/class/cs110/lecture-examples/map-reduce/`** (e.g., **`group-by-key.py`**, **`word-count-reducer.py`**) to do some of this work for you. To run them, you could use the subprocess function we've created in class, but it is going to be easier to use the **`system`** function, which is already used in parts of the assignment. See the **`system-example.cc`** example to see how easy it is to use.
    - Hint 2 in this section is important:

      Once the MapReduce job has transitioned to the reduce phase, you should rely on the server to respond to reducer client requests with file name **`patterns`** instead of actual file names. The server, for instance, might send an absolute file name pattern ending in **`files/intermediate/00001`** as an instruction to the reducer that it should gather, collate, sort, and group all intermediate files ending in **`.00001.mapped`** before pressing all of it through the reduce executable to generate **`files/output/00001.output`**.