CS 61A Week 13

Topic: Analyzing evaluator, MapReduce

Reading: Therac paper in course reader.
analyzing evaluator: `~cs61a/lib/analyze.scm`

Homework:

A&S exercises 4.22, 4.23, 4.24

MapReduce exercises:

1(a). Google uses MapReduce for a variety of search-engine-related tasks involving processing large data sets. An example of such a task is building an inverted word index. For each word in a given set of documents, we want to generate a key-value pair, where the key is the word and the value is all documents in which the word appears. Write a procedure that takes a directory name as argument and returns the inverted index stream.

1(b). The list above will include words like "a," "so," "the," etc. Suppose now that we only want "important words." We can use word length as a measure of importance (a real search engine would use a more complex heuristic). Write a procedure that returns an index where all the words are N or more letters long, taking the filename and N as arguments.

2. Google also manages the GMail e-mail service, and they would like to filter out as many spammers as possible. In this problem you will implement a simple spam filtering idea with `mapreduce` so that it can be efficiently applied to the multitudes of e-mail messages in GMail.

We want to produce a "blacklist" of e-mail addresses that are spamming GMail users. Spam messages are sent to many addresses at once, and so a spam message can be identified by having one of the most commonly-used subject lines. If an address sends many messages with the most common subject lines, it is likely a spammer address. We would like to find the top ten addresses that have sent the most messages with frequently-recurring subject lines.

The input data is a collection of e-mail records in the file `/sample-emails`. (These are simulated messages, not actual GMail data!) Each e-mail record is a list with the format

```
(from-address to-address subject body)
```

where each element is a double-quoted string. The mapper function will be applied to each e-mail record. An small example set of e-mail records is below.

```
("cs61a-tb" "cs61a-tc" "mapreduce" "mapreduce is great! lucky students!")
("bot1337" "cs61a-ta" "free ipod now!" "buy herbal ipod enhancer!")
("bot1338" "cs61c-tf" "free ipod now!" "buy herbal ipod enhancer!")
...
```

Problem continues on next page...

Week 13 continued:

2(a). Our first step is to produce a table with subject lines as keys and counts of occurrences as values. Identify the intermediate key-value pairs and write the mapper and reducer functions.

2(b). From our tabulation of subject line occurrences in 2(a), we want to find the most common subject lines in the table. We can perform a sort by using the fact that MapReduce sorts by intermediate keys into the reducer groups. Identify the intermediate key-value pairs and write the mapper and reducer functions.

2(c). Finally, assuming you've moved the ten most common subject lines into a list, we want to make a table with from-addresses as keys and counts of e-mails sent with common subject lines as values. You don't need to sort the table, since the procedures would be identical to those in 2(b). Identify the intermediate key-value pairs and write the mapper and reducer functions.

3. Sometimes either the map or reduce operation is trivial, such as an identity function. Is MapReduce still the best way to handle those cases, or would it be better to provide separate `map` and `groupreduce` functions as we did for the non-parallel toy version? Why?

4. Could you use MapReduce to perform a parallelized Sieve of Eratosthenes? If so, describe the process briefly. If not, why not? (Don't try to solve this problem by quizmanship; there's a case to be made for both answers!)

Note: Part I of project 4 is also due next week.

Unix feature of the week: `sort`, `cut`, `paste`, `join`

Emacs feature of the week: `C-x C-o`, `M-\` (delete blank space)