Assignment 3

Published 9. oktober 2012 by Carøe, Michael Kirkedal
Mandatory Yes
Deadline 25. oktober 2012 16:00
Assessment Bestået/ej bestået (Passed/Failed)
Use groups Learner-defined groups
Assignments

Remember that song?

Objective

The objective of this assignment is to gain hands-on programming experience with Erlang and to g
The goal is to implement a simple map-reduce framework, and then use this framework to proces
The assignment consists of two parts, and an optional third part:

- Part 1: Implementing a Map-Reduce framework, that is independent of mxm.
- Part 2: Use your framework to implement some simple algorithms processing the mxm data
- Part 3: Suggestions for various extensions. This part is **optional**, it will have no influence w

What to hand in

You should hand in two things:

1. Your code.
2. A short report explaining the code, and an assessment of the quality of code including what

Scope

The following topics are not within the scope (of Part 1 and 2) of this assignment:

- Error handling
- Absolute performance

Part 1: Map-Reduce Framework

As entry-point for the framework we only want to communicate with a single process, which in tur
Thus, in the framework there are three kinds of processors:

- A single *coordinator* that serves as the entry-point for the framework.
- A fixed size pool of *mappers* that takes care of the mapping phase.
- A single *reducer* that takes care of the reducing phase.

API for the Coordinator

The API for the coordinator should be exposed from at module called mr. The module should expo

- A function start(N) for starting a coordinator with N mappers and a single reducer. The coor
- A function job(Pid, MapFun, RedFun, Initial, Data) for starting a new Map-Reduce job, where
  The arguments should have the following types (here using Haskell type-syntax):

  ```
  MapFun  :: a -> bRedFun  :: b -> res -> resInitial :: resData    :: [a]
  ```

- A function stop(Pid) for stopping the coordinator and all worker processes for that coordinat

Example

The following example shows how to use the framework for adding and factorial of the numbers 1

```
test_sum() ->    {ok, MR}  = mr:start(3),    {ok, Sum} = mr:job(MR,
```

Implementation

We encourage (but don't require) you to base your solution on the provided skeleton implementat
Use the following rough sketch for the implementation of job:

- The coordinator sends the relevant functions and initialisation data to the mappers and red
- The coordinator sends the data asynchronously to the mappers, for instance by using the s
- When a mapper receive some data it processes the data and asynchronously sends the res
- When the reducer receive some data from a mapper it computes a new intermediate result

Part 2: MusicXMatch Dataset

The *musiXmatch dataset* (mxm) is the official collection of lyrics from the Million Song Dataset (M

Getting the data

- The cut down test version of the bag-of-words: http://labrosa.ee.columbia.edu/millionsong/s
- The full version of the bag-of-words: http://labrosa.ee.columbia.edu/millionsong/sites/defaul
- The matching with MSD: http://labrosa.ee.columbia.edu/millionsong/sites/default/files/Addit

If you have wget on your machine you can use the commands:

```
wget http://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFile
```

You can use the provided read_mxm module to read in the bag-of-words datasets.

Tasks

In all the following tasks you should use the mr framework from Part 1, and the bag-of-words data

1. Compute the total number of words in all songs together.
2. Compute the average number of different words in a song *and* the average total number of
3. Make function grep that for a given word can find the MSD track IDs for all songs with that
4. Compute a reverse index, that is a mapping (as a dict) from words to songs where they occ
5. Discuss (shortly) advantages and disadvantages of using a reverse index over the grep fun

Assessment

Your hand-in should contain a short section giving an overview of your solution, including **an asse**

Hints

- What kind of data (state) should each process keep track of?
- Start by thinking about which messages the different kinds of processes should send to eac
- If you want to start with Part 2 before making Part 1, it's relatively straightforward to impler
- Make yourself familiar with the manual page for lists and the manual page for dict for inforr
- While developing the program I've found it helpful to use io:format to print various informat

  ```
  io:format("~p is coordinator with the mappers ~p ~n", [self(), Mappers]),
  ```

  (Yes, it's a side-effect but it's useful.) See the manual page for the io module for more inforr
- If you want to time a function, the timer:tc function comes handy.

Part 3: Extensions

Here are some suggestions for extensions to the assignment, in no particular order.

- **Contest:** Suggest interesting queries/computations you can perform over the dataset. The
- The framework only have a single reducer, extend the framework so that there is a fixed pc
- Currently the initial reading in and splitting of data is done sequentially. Change that so tha
- No error handling is required for Part 1 and 2. Use the techniques (to be) taught in the cour
- Change the API so that the coordinator also takes a list of Erlang nodes as argument, and th

mr_skel.erl
read_mxm.erl

Submit answer Deadline 25. oktober 2012 16:00