Project 4

COMP 443 – Programming Languages
Spring 2016

Due Tuesday, April 19 by 11:55 pm

Overview

For this project, you will be using actors to create a concurrent version of a web search engine.

You can choose to work in a group of two people on this project or you can complete it yourself. You should put a clear comment at the top of your Main.scala that indicates who was in your group (which could just be your own name). All of the work on this project should be your team's own work. Do not show code to anyone outside your group, nor look at anyone else's code for this project. You can discuss general concepts with classmates (e.g., How do I create an Actor?), but not how they pertain to this project.

Learning Objectives

Completing this project will give you experience with using the Akka actor framework in Scala.

Setup

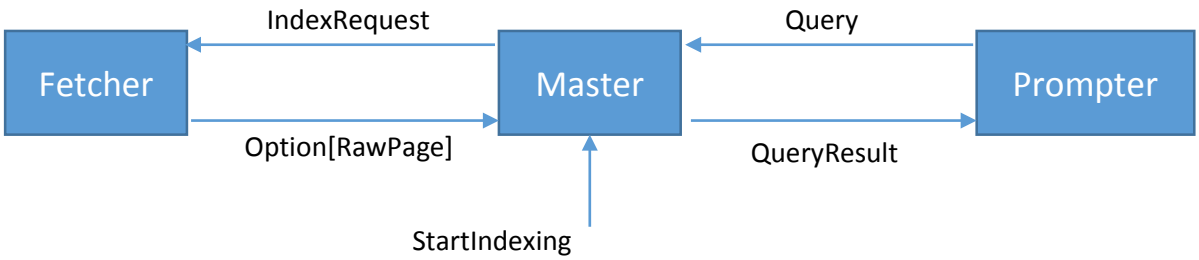
Download the starter code from the class website.

To complete the project, you only need to complete the TODO sections in Actors.scala and create a Main.scala (described below).

Points for each part are indicated in square brackets next to the instructions.

Actors

Here are the actors and messages we will use in this project:



There will be one Master actor.

- When it receives the StartIndexing message [3], it should create one Fetcher actor for each url in the message [8], sending each respective actor an IndexRequest for that url [8]. The Master

should also create one Prompter actor [5], sending it a dummy QueryResults message with 0 total pages [3]. This dummy message causes the Prompter to issue the first prompt for a Query.

- When it receives a Query message [3]...
  - If there are no terms in the message, shut down the actor system [5]
  - Otherwise, compute the number of indexed pages overall and the fraction of those pages that contain all of the terms in the Query [8]. (The Page class has a method to make this easy.) Those two values should be wrapped in a QueryResults message and sent back to whoever sent the Query message [5].
- When it receives an Option[RawPage]: This code is provided for you. See the comments for details.

There are several Fetcher actors, each of which retrieves the source of web pages. You are to write the Fetcher class so it inherits from Actor [3]. When it receives the IndexRequest message [3] it should...

- Use a statement like val src = scala.io.Source.fromURL(url).getLines.mkString("\n") to get the HTML source for the web page. [3]
- If the previous statement throws a java.io.IOException, catch it [5] and send a None message to the sender of the IndexRequest message [3].
- Otherwise (no IOException), send Some(RawPage(...)) as a message back to the sender of the IndexRequest message, passing constructor arguments to RawPage as appropriate [8].

There is one Prompter actor that asks the user for queries. The full code for Prompter is provided.

Main

In a file Main.scala, create a Main object [3] that contains a main method [3]. Your main method should create an actor system [8] and a Master actor [5]. Send the Master actor a StartIndexing message [5] with the collection of command line arguments given to main (i.e., the args in def main(args: Array[String]): Unit).

Tips

I had trouble with the readLine prompt in the REPL in Windows; it would accept some of my characters but not others. But I had no problems when compiling the scala code and then running the compiled bytecode. Here is how you can do that:

- Run scalac *.scala from the directory with your Scala files. If successful, it will create several .class files. These are Java bytecode, compiled for the Java Virtual Machine.
- Run the program as follows:
  scala -J-Xmx2g Main http://www.gcc.edu http://www.bbc.com http://www.biblegateway.com
  - The -J-Xmx2g tells the Java Virtual Machine that it can use to up 2GB of memory. This makes it less likely to run into OutOfMemory errors.
  - The Main is the name of the object containing the main method you want to run.
  - The urls are the command line arguments for main.

Actor-based programs can be difficult to debug. Fortunately, the Akka actor framework makes it easy to get log messages about actor events. In the application.conf file from the starter pack, you will see several lines of commented configuration options. If you want to see log messages, uncomment the loglevel assignment, in addition to the following steps:

- If you want to see log messages about actor startup, creation, shutdown, etc., uncomment the configuration line that sets lifecycle = on.

- If you want to see log messages about all of the messages the actors receive, uncomment the configuration line that sets receive = on. In addition, for each actor class that you want to display the log messages, you need to replace the "def receive = {" line with "def receive = akka.event.LoggingReceive {". That is, add akka.event.LoggingReceive.

There can be lots of messages, so if you turn on logging, you should also set Master.maxPages to a smaller number.

Handin

Zip your .scala files and submit the zip file on myGCC. **Do not include any .class files. [3]**

Grading

Since this project is smaller in scope than the other projects, it will be worth a smaller fraction of your overall course grade.