CS 677 Distributed Operating Systems

Spring 2013

Programming Assignment 3: Angry birds : Replication, Fault Tolerance and Cache Consistency

Due: Tue Apr 30 2013

- You may work in groups of two for this lab assignment.
- Purpose of this project: to familiarize you with the concepts of Replication, Fault Tolerance and Cache Consistency.
- You can be creative with this project. You are free to use any programming languages (C, C++, Java, etc.) and any abstractions such as sockets, RPCs, RMIs, threads, events, etc. that might be needed. You have considerable flexibility to make appropriate design decisions and implement them in your program.

- **A: The problem**

  - A follow-up on the previous story.

    The pigs are now smart enough to avoid bird attacks to an acceptable degree. Their success has meant prosperity in the pig world and their numbers have increased. In order to remain effective against the birds, they now decide to have two coordinators to share the important task of spreading information about the bird attacks and maintaining reports of active and dead pigs. The task of the coordinator pig is now replicated on two pigs – pig Oink and pig Doink.

    Each of the two coordinators is responsible for a mutually exclusive set of pigs. The two decide before-hand which pigs they will be responsible for. The two coordinators are also responsible for maintaining the town's records in a central register in the mayor's office. One game now consists of multiple iterations of bird launches. Each coordinator, at the end of each iteration, updates the town's register with information about the pigs that it is responsible for. Because the mayor's office is far away, they choose to maintain a notebook (local cache) in their own homes which is a local copy of the town register. They need to work out a suitable agreement on how to ensure that the town's register is maintained correctly when changes are made to their local notebooks. They have asked you for suggestions on how to do this correctly.

Finally, Oink and Doink occasionally fall asleep and fail to respond to bird launches. When one falls asleep, the other is responsible for all the pigs and for maintaining the town register. This is an example of fault tolerance.

In addition to the pigs and the bird, you need to add a third entity to your system: a database server. Your database can simply consist of a file that contains information about the pigs' location and whether it is alive or dead (the town register). The database "server" is a process that can access and update this file. Coordinator pigs modify this information at the end of each game iteration via the database server. As mentioned earlier, the coordinators cache this file information locally. Cache consistency needs to be addressed when information is updated by the coordinators.

- This project is based on project 2. Assume that the number of pigs *N* is specified beforehand (*N* should be configurable in your system). As usual, each peer is both a client and a server. Each peer should be able to accept multiple requests at the same time. We will ignore ordering based on logical clocks in this assignment (you may still use it, but it is no longer required). Assume that each game now consists of multiple bird launches (say 10-20).

  First construct a connected network. Assume that a list of all *N* peers and their addresses/ports are specified in a configuration file.

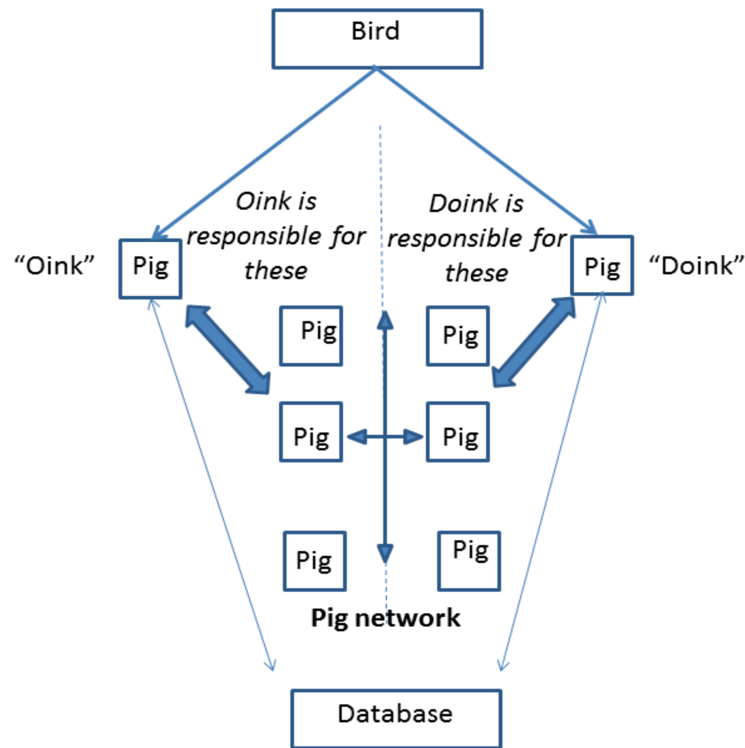
  Once the network is formed, randomly pick two pigs to be coordinators (leader election is no longer required). Then randomly assign each pig to one of the two coordinators. When a bird launches, it informs both coordinators of its landing location and speed. Each coordinator then sends bird approaching and status requests to the pigs that are assigned to it and eventually request status from those pigs.

  Implement the process that acts as a database server. It should minimally support a lookup for a pig's location and status, and updating of this information. Implement simple locking to prevent the two coordinators from concurrently modifying the database.

  Since coordinators cache pig information, choose a suitable cache consistency algorithm based on the ones discussed in class. Strong consistency is needed, since the pig town register is very important for pig town planners. The consistency can be achieved either using *push* or *pull* and the technique can be stateful or stateless. You have considerable flexibility to make appropriate design decisions.

  For simplicity, the database itself can be a single text file with one entry (row) per data item.

- A pictorial representation of the system is as shown in the figure below.



- The second part of the assignment involves fault tolerance. Sometimes one of the coordinators becomes inactive and does not do its task of sending bird approaching messages. In this case, the other coordinator has to send the bird approaching message to all pigs in the network. You can implement this by assuming that when a coordinator receives the information from the bird, it checks to see if the other coordinator is active or not (*are you there?*). If the other coordinator is inactive, the active coordinator takes over responsibility for all pigs. You may assume that other coordinator stays active or inactive until that iteration of the game is complete.
- No GUIs are required. Simple command line interfaces are fine.
- **Optional Extra Credit Question:** In this optional component, we will use cloud computing to dynamically replicate the coordinator. Initially, start with one coordinator (one EC2 server). Monitor the load on the server (in terms of number bird launches per minute). If the number of launches exceeds a (small) threshold, have the first coordinator dynamically start a new EC2 server and run a second coordinator on that machine and hand off responsibility for some pigs to that coordinator. The bird must be informed of this new replica so that it can inform both coordinators of its landing coordinates. No fault tolerance functionality is needed in this extra credit question (Side Note: although not required, do think of a design as to how a failure of one coordinator allows you to dynamically start a different coordinator on a separate EC2 server and add a small note to the design document as to how you may have implemented such a functionality).

- **B. Evaluation and Measurement**

1. Deploy at least 6 peers. They can be setup on the same machine (different directories) or different machines. You are free to develop your solution on any platform, but please ensure that your programs compile and run on the edlab machines (See note below).
2. Do a simple experiment study to evaluate the behavior of your system. Compute the average score for the game after running many bird launches in each game. Also, observe the performance change due to the replication (you may simulate a one coordinator version of your system by letting one coordinator handle all the pigs and the other coordinator be inactive all the time). Make necessary plots to support your conclusions.

- **C. What you will submit**

· When you have finished implementing the complete assignment as described above, you will submit your solution in the form of a zip file that you will upload into moodle.
· Each program must work correctly and be **documented**. The zip file you upload to moodle should contain:

1. An electronic copy of the output generated by running your program. Print informative messages when a pig receives and sends key messages and the score. Do not print multiple flooding messages since it will clutter your output.
2. A separate document of approximately two pages describing the overall program design, a description of "how it works", and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made). You also need to describe clearly how we can run your program - if we can't run it, we can't verify that it works.
3. A program listing containing in-line documentation. **Important** – Please list which classes and methods in your code implement each of the requirements of the project.
4. A separate description of the tests you ran on your program to convince yourself that it is indeed correct. Also describe any cases for which your program is known not to work correctly.
5. Performance results.

- **D. Grading policy for all programming assignments**

1. Program Listing
   works correctly ------------- 50%
   in-line documentation -------- 15%
2. Design Document
   quality of design and creativity ------------ 15%
   understandability of doc ------- 10%

3. Thoroughness of test cases ---------- 10%
4. Grades for late programs will be lowered 12 points per day late.

- **Note about edlab machines**

- Read more about edlab and how to access it here

· Although it is not required that you develop your code on the edlab machines, we will run and test your solutions on the edlab machines. Testing your code on the edlab machines is a good way to ensure that we can run and grade your code. Remember, if we can't run it, we can't grade it.

· There are no visiting hours for the edlab. You should all have remote access to the edlab machines. Please make sure you are able to log into and access your edlab accounts.

· IMPORTANT - No submissions are to be made on edlab. Submit your solutions only via moodle.