INF5570 - Distributed Objects
Home Exam 2

Department of Informatics
University of Oslo

April 11th, 2019

Foreword

This is the second, and last home exam in IN5570, Spring 2019. You must pass both home exams to be admitted to the final, oral exam. The home exams are graded, and each makes up ¼ of your final grade; the oral exam makes up ½ of your grade. Resubmission is not possible, but you will receive feedback on your work, in due time before the oral exam.

Task

Create a basic, distributed framework for Primary-Copy (object) Replication. The framework should allow the user to replicate a given object over a given number of machines (nodes), as well as access, and update the replicated objects via their replicas.

To offer such functionality you could create an operation `replicate[X,N]`, where `X` is the object that should be replicated, and `N` is the number of machines to replicate the object over. The framework should make a best-effort to maintain `N` replicas of the given object. However, the number of possible replicas is always bound by the number of available machines. You are allowed to require that the given object has an operation `clone[]`, which returns a clone of the original object.

With Primary-Copy Replication, all updates must be performed on the *primary copy*. If a replica that is not the primary copy wants to perform an update, the update must be delegated to the primary copy. How these replicas address the primary copy, and how a user program accesses a replica, is up to you define. (NB! Emerald has no built-in one-to-many reference mechanism.) In classical Primary-Copy Replication, when the primary copy has updated its state, it must make sure that all replicas also update their state, before any further updates to the primary copy can take place.

Your framework should keep track of the available nodes. When new machines are detected, they become eligible to host replicas. A node can host the replicas of multiple objects, but must not host multiple replicas of the same object. When a machine becomes unavailable, the framework should automatically recreate the replicas hereby lost. If a primary copy becomes unavailable, the framework should choose a replica to act as a new primary copy. How exactly you elect the new primary copy, is up to you to decide. A replica need not keep track of all of its replicas; you can leave this accounting to the framework.

Overall, you may consider making an implementation that conforms to the type given below. It is not mandatory that you do so; this is merely a suggestion. Justify your design in your report. It is OK to use type assertions (**view** ... **as** ...) in your implementation. You are free to use any design patterns that you deem relevant. Consider using the Observer Design Pattern.

```
const PCRType <- typeobject PCRType
  operation replicate[X : t, N : Integer]
  forall t
  suchthat
    t *> typeobject ot
      op clone -> [result : t]
    end ot
  operation replicas[X : t] -> [Array.of[rt]]
  forall t
  where
    rt <- typeobject rt
      operation read -> [o : t]
      operation write[o : t]
    end rt
end PCRType
```

Testing

You must also test the framework with at least two tests:

1. One where you will maintain a name server with a lookup operation with a given name, returns the matching object.
2. One where you will maintain a time server as the same as you did during your earlier assignment. The primary replica will maintain the time. If N is larger than the number of available machines, there should be a replica on each machine as far as it goes.

Use PlanetLab, together with local Docker instances, to test your application.

Level of Ambition and Evaluation

A well-documented, simple solution will be evaluated as better than an advanced solution that works only partially, or is poorly documented. Beware that main goal of the exam is building a distributed set of cooperating objects and that the replication is merely used here to illustrate the main goal. Thus you should not worry about creating the best possible replication scheme; simple solutions will suffice.

Delivery

1. A short (3 - 6 pages) report that contains:
   - An analysis of the most significant decisions you made regarding the program. First and foremost design decisions including any design patterns that you have chosen to use, if any.
   - A description of the main classes/objects in the program.
   - A description of how you have tested the program and why you have chosen to test as you do. For example, you should mention if you have tested the program in several parts and you should describe the resulting output briefly. Include an honest appraisal of how much of the program works.
2. The source code.
3. A Makefile, optional.
4. Output from your test runs.

The deadline for the assignment is **May** 14^{th}**, 2019 at 23:59 CEST (21:59 UTC)** Deliver your assignment in devilry: let the zip/tar file have the name <username>.<zip/tartype>. <username> is your user name/e-mail on uio.no.

Last, but not Least

Good luck! And have fun! :)