Big Data Systems Assignment 1

This assignment is due on Dec 7, 23:59. While individual hand-ins will be accepted, we strongly recommend that this assignment be solved in groups of maximum two students.

A well-formed solution to this assignment should include a PDF file with answers to all questions posed in the programming part of the assignment. In addition, you must submit your code along with your written solution. Evaluation of the assignment will take both into consideration. Note that all homework assignments have to be submitted via Absalon in electronic format. It is your responsibility to make sure in time that the upload of your files succeeds. While it is allowed to submit scanned PDF files of your homework assignments, we strongly suggest composing the assignment using a text editor or LaTeX and creating a PDF file for submission. Email or paper submissions will not be accepted.

Learning Goals

This assignment targets the following learning goals:

- Understand the actor-oriented programming model in order to effectively develop programs on top of actor frameworks, such as Microsoft Orleans;
- Understand the use of logs as an application programming paradigm and learn to design an application following the OLEP approach.

Programming Task

In this programming task, we provide you with a small banking application that implements traditional bank account functionalities (e.g., withdraw and transfer) using a conventional concurrency control protocol, in this case, 2-Phase Locking.

We ask you to understand the implementation of this application and map this application scenario to follow an OLEP approach. In other words, it is expected from you that you transform the bank application provided to you into an OLEP-based application. Keep in mind that this mapping would require you to design the necessary actors and their behavior appropriately using the actor and the log abstractions to communicate operations to be performed over (privately encapsulated) data.

Furthermore, it is expected that you make use of good system's design practices, particularly those learned throughout the *Advanced Computer Systems* (ACS) course, such as strong modularity, error handling, and techniques for performance.

Preliminaries

1. **Scenario Application.** In this assignment, you build upon the sample Orleans project *BankAccount*. Make sure to clone or refer to this repository to understand the application requirements expected for a banking system in your assignment.

2. **Orleans Streams.** As a log abstraction, it is expected that you employ *Orleans Streams* to enable asynchronous event processing across the actors of your application. A sample project with a simple solution employing Orleans Streams can be found under Files (Streaming.zip)
3. **Workload Generation.** You are expected to generate the workload for this assignment. That is, the accounts participating in the bank operation as well as the amount involved in the operation. The sample Orleans project *BankAccount* provides an example (check BankClient/Program.cs) on how such workload can be generated.

Questions & Answers Regarding the Programming Task

1. Can I transfer money from account X to the same account X? No. Make sure the workload generated does not allow such an operation.
2. Can a client account have a negative balance? No. The application should never allow such a thing to occur.
3. Does the workload generated should only create operations that lead to non-negative balance in the accounts? No. The workload generated should also create operations that, if processed, will lead to negative balance in the application states. The constraint on non-negative balance should be enforced by the program logic.
4. Can I take advantage of libraries (i.e., Nuget packages) provided by the .NET framework? Yes, but make sure to make wise decisions and report the reason for opting for such a package in your solution.
5. Do I need to target the best performance possible? Although it is recommended that you reason about designs that allow for increased throughput, **we recommend you to come up with a design that fulfills the application requirements first**, which is the expected outcome of this assignment. Remember that optimizing for performance earlier in the system's project may lead to increased (and unexpected) efforts to reach the requirements.
6. Is it allowed to start all accounts with a positive balance? Yes.
7. Should I rely on a durable event distribution system (e.g., Apache Kafka or Apache Pulsar)? Not necessarily. You can assume an in-memory stream provider is sufficient for this assignment. Refer to the Simple Message Stream Provider for further information and to the Orleans Streams project provided with this assignment.
8. Should I use one log per bank account as in the OLEP paper? Not necessarily. For this assignment you can use a single log per operation.
9. Should I target a multi-server deployment? No. A single-server deployment is sufficient for the purpose of this assignment.

Questions for Discussion

In addition to the implementation above, reflect about and provide answers to the following questions in your solution text:

1. Why does the Online Event Processing (OLEP) approach fit the application scenario presented in this assignment? In particular, what problems does OLEP address that makes it a fit for this application scenario?
2. Do you believe the actor model is a natural fit for OLEP-based applications? Justify your answer.
3. What are the relationships and differences between BASE and OLEP?
4. Explain briefly how your application would differ if designed following the BASE model instead of the OLEP approach.
5. Suppose that instead of only having transfers from one account to another, the application requirements change to reflect the need to support account transfers from multiple sources. That is, a transaction may now involve multiple source accounts transferring money to a single account. Does your current OLEP application design safeguard atomicity in this case? If yes, justify why this is the case. If not, explain the necessary measures to adapt the design to the new application scenario.