Ira A. Fulton Schools of Engineering
Arizona State University

CSE 531: Distributed and Multiprocessor Operating Systems

Client-Centric Consistency Project

Purpose

The goal of this project is to implement the client-centric consistency model on top of the gRPC Project. Your job is to implement the essential functions that enforce the client-centric consistency, specifically Read your Write consistency of the replicated data in the bank.

Objectives

Learners will be able to:

- Implement the essential functions that enforces the client-centric consistency.
- Enforce the Read your Write policy, which extends the implementation of previous interfaces.
- Determine the problem statement.
- Identify the goal of the problem statement.
- List relevant technologies for the setup and their versions.
- Explain the implementation processes.
- Explain implementation results.

Technology Requirements

- Access to Github
- Python
- gRPC
- You are required to use the files in the "Projects Overview and Resources" page in the *Welcome and Start Here* module of the course.

Directions

Part 1: Written Report

Your written report must be a single PDF with the correct naming convention: *CSE 531_Your Name_Client-Centric Consistency_Written Report*.

Using the provided *Learner Template_CSE 531_Your Name_Client-Centric Consistency_Written Report*, compose a report addressing the questions:

1. What is the problem statement?
2. What is the goal of the problem statement?
3. What are the relevant technologies for the setup and their versions?
4. What are the implementation processes?
5. What are the implementation results and their justifications?

*Learners may add subheadings on the template to purposefully call attention to specific, organized details.

Part 2: Project Code

Major tasks:

1. Extend the implementation of Branch.Withdraw and Branch.Deposit interface to enforce Ready your Writes policy.

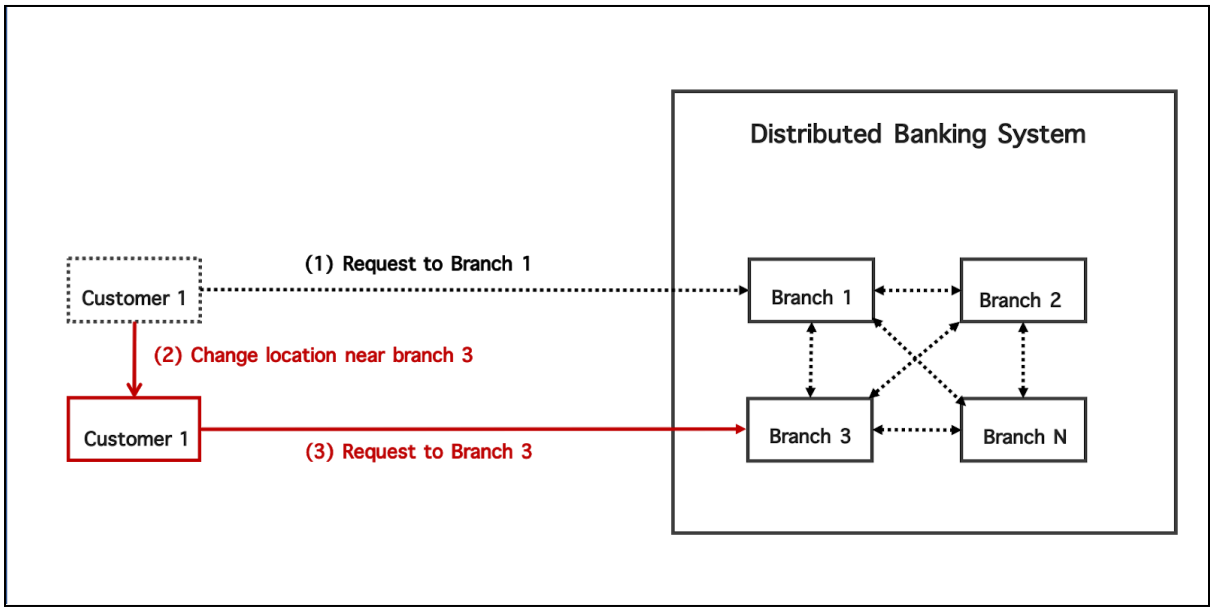

Diagram A: The customer changes the branches while submitting request to the Bank

1. Description

The customer described in (1) of diagram A, accesses the banking system by connecting to one of the replicas in a transparent way. In other words, the application running on the customer's mobile device is unaware of which replica it is actually using. Assume the customer performs several update operations and then disconnects. Later the customer accesses the banking system again possibly after removing to a different location or using a different access device. At that point the customer may be connected to a different replica than before as shown in (2), (3) of Diagram A. However if the updates performed previously have not yet been propagated, the customer will notice inconsistent behavior. In particular, the customer would expect to see all previously made changes, but instead it appears as if nothing at all has happened. This problem can be alleviated by introducing client-centric consistency. In essence, client-centric consistency provides guarantees for a single client concerning the consistency of accesses to a data store by that client.

1.1. Read your Writes

A data store is said to provide read-your-write consistency, if the following condition holds: The effect of a write operation by a process on data item x will always be seen by a successive read operation On x by the same process. Suppose the customer deposits $100 in an empty bank account, and is planning to first check if the bank account has $100 and then withdraws $100. The customer issues sequential order of requests(deposit->query->withdraw) to the server. The customer may fail to withdraw the money from the bank because the query request can return $0. This annoying problem arises because the query request contacted a replication which the new deposit request had not yet propagated.

1.2. Implementation

Unlike the gRPC Project, where the customer communicates with only a specific branch with the same unique ID, this project allows the customer to communicate with different branches for query, withdrawal, and deposit operations.

The Branch processes generate the IDs for write operations (deposit and withdraw) requested by the customer. The Customer process obtains the IDs of the write operations performed by the Branch processes. Both the Customer process and the Branch process maintain these sets of write IDs, i.e., writesets. When the Customer sends a request to a Branch, it also sends its writeset. The Branch compares the received writeset to its own writeset, and uses them to enforce client-centric consistency in the banking system.

File structure should remain the same as the gRPC Project. The server and client files should also follow execution as stated in the gRPC Project.

1.2.1. Input and Output

The input file contains one Customer and multiple Branch processes. The format of the input file follows the gRPC Project, you will be using the destination parameter “dest” which has a value of a unique identifier of some branch.

```
[ // array of processes
{ // Customer process #1
        "id" : {a unique identifier of a customer},
        "type" : “customer”,
        "events" : [{“id“ : {unique identifier of an event}, "interface" : {query | deposit | withdraw},
        "money" : {an integer value}, “branch” : {a unique identifier of the branch} }]}
{ // Branch process #1
        "id" : {a unique identifier of a branch},
        “type” : “branch”
        “balance” : {the initial amount of money stored in the branch}
}
{ ... } // Branch process #2
{ ... } // Branch process #3
{ ... } // Branch process #N
```

```
]
```

Output file format for the Read your Writes:

```
[ // array of customers
   { // Customer process #1
        “id” : { a unique identifier of a customer }
        “recv” : [a list of successful returns of the events from the customer process]
   }
]
```

1.2.2. Read your Writes Example

The test-cases for the Read your Writes consistency implementation will generate Customer processes that perform a sequence of write and read operations on different Branch processes.

Example of the input file

```
[
  {
    "id" : 1,
    "type" : "customer",
    "events" :  [{"id" : 1, "interface" : "deposit", "money" : 400, "branch" : 1}, {"id" : 2, "interface":"query",
"branch" : 1}, {"id" : 3,"interface" : "query", "branch" : 2 }]
  },
  {
    "id" : 1,
    "type" : "branch",
    "balance" : 0
  },
  {
    "id" : 2,
```

```
   "type" : "branch",
   "balance" : 0
  }
]
```

The customer will perform write and read operations to different Branch processes. The read operation performed by the customer should reflect the correct result of write operations that the customer performed before.

Expected output file:	**Wrong output file:**
[{"id": 1,"recv": [{"interface": "deposit","branch": ,"result": success}]}, {"id": 1,"recv": [{"interface": "query","branch": 1,"balance": 400}]}, {"id": 1,"recv": [{"interface": "query","branch": 2,"balance": 400}]},]	[{"id" : 1, "balance" : 400 }]

Formatting Specifications

Naming convention for your project files and usage with the input file:

- The server file should be named server.py, all required servers should be able to start with this command **python server.py input.json** (execution command to start servers)
- Client file should be named client.py, client file should execute with **python client.py input.json**.

Protobuf: proto file should be inside the protos folder (e.g., **protos/banks.proto**). The expected file structure (excludes files generated after running proto file) should be like this:

```
├──protos
├── ├──banks.proto
├──input.json
├──server.py
├──client.py
├──customer.py
├──branch.py
├──output.json
```

Submission Directions for Project Deliverables

Part 1: Written Report

You are given a limited number of attempts to submit your best work. The number of attempts is given to anticipate any submission errors you may have in regards to properly submitting your best work within the deadline (e.g., accidentally submitting the wrong paper). It is not meant for you to receive multiple rounds of feedback and then one (1) final submission. Only your most recent submission will be assessed.

You must submit your Client-Centric Consistency Project Written Report deliverable in its submission space in the course. Learners may not email or use other means to submit any assignment or project for review, including feedback, and grading.

The Client-Centric Consistency Project Written Report includes one (1) deliverable:

- **Written Report:** Your written report must be a single PDF with the correct naming convention: *CSE 531_Your Name_Client-Centric Consistency_Written Report.*

Part 2: Project Code

You are given an unlimited number of attempts to submit your best work. You must submit your Client-Centric Consistency Project Code deliverable through Gradescope. Carefully review submission directions outlined in this overview document in order to correctly earn credit for your work. Learners may not email or use other means to submit any assignment or project for review, including feedback, and grading.

Submitting to Gradescope

Your submission will be reviewed by the course team and then, after the due date has passed, your score will be populated from Gradescope into your Canvas grade.

1. Go to the Canvas Assignment, "**Submission: Client-Centric Consistency Project Code**".
2. Click the "**Load Submission…in new window**" button.
3. Once in Gradescope, select the project titled, "**Submission: Client-Centric Consistency Project Code**", and a pop-up window will appear.
4. In the pop-up,
    a. Submit a single ZIP file.

    b. Click "**Upload**" to submit your work for grading.

5. You will know you have completed the assignment when feedback appears for each test case with a score.

6. If needed: to resubmit the assignment in Gradescope:

    a. Click the "**Resubmit**" button on the bottom right corner of the page and repeat the process from Step 3.

The Client-Centric Consistency Project includes one (1) deliverable:

- **ZIP File:** Your ZIP file must contain your Client-Centric Consistency Project code files and final output. The **code files** must follow the naming conventions outlined in the "Formatting Specifications" section of this Overview Document. The **final output** must be a single JSON file with the correct naming convention: **output.json**.
    - Zip your files by selecting all of them together (your code files should be in the root folder).

Evaluation

Project deliverables will be evaluated based on criteria and will receive a total score.

Evaluation details vary depending on whether the component is auto-graded or course team-graded, so review this section carefully so you understand how you earn credit for each portion of your work.

Review the course syllabus for details regarding late penalties.

Test Case

This component of the project is **auto-graded** and worth **40%** of your project grade.

Your code must be able to handle following:

- **Read your Writes:** The code must be able to perform the Read your Writes model and ensure client-centric consistency.
    - **Example of Read your Writes:** Customer 1 performs deposit on branch 1 with 400. Then he moves to branch 2 and makes a query transaction. The result was 400. If it was 0, that means there is no consistency in the system and the Read your Writes model fails.

Your grade for this portion will be assigned based on the **percentage** of returned balance values that are **correct**. This schema for calculating the grade is also provided in the **rubric**:

- About **20%** of the return balance values are correct.
- About **40%** of the returned balance values are correct.
- About **60%** of the returned balance values are correct.
- About **80%** of the returned balance values are correct.
- **All** of the returned balance values are correct.

Code

This component of the project is **course team-graded** and worth **40%** of your project grade.

Review the **rubric** for how your code will be graded.

Project deliverables missing any part of the project will be graded based on what was submitted against the rubric criteria. Missing parts submitted after the deadline will not be graded.

Report

This component of the project is **course team-graded** and worth **20%** of your project grade.

Review the **rubric** for how your written report will be graded.

Project deliverables missing any part of the project will be graded based on what was submitted against the rubric criteria. Missing parts submitted after the deadline will not be graded.

Rubric

Rubrics communicate specific criteria for evaluation. Prior to starting any graded coursework, learners are expected to read through the rubric so they know how they will be assessed. You are encouraged to self-assess your responses and make informed revisions before submitting your final report. Engaging in this learning practice will support you in developing your best work.

Component	Criteria	No Attempt	Undeveloped	Developing	Approaching	Proficient	Exemplary

Test Case	Check if the correct balance is reflected across branches. All transactions should show consistency across all branches.	Provided no response.	About 20% of the returned balance values are correct.	About 40% of the returned balance values are correct.	About 60% of the returned balance values are correct.	About 80% of the returned balance values are correct.	All of the returned balance values are correct.
Component	**Criteria**	**No Attempt**	**Undeveloped**	**Developing**	**Approaching**	**Proficient**	**Exemplary**
Code	The code contains all necessary components, can perform Read Your Writes, and functions correctly.	Provided no response.	Provided project code that is syntactically or semantically invalid.	Provided project code is functional. Project code performs a few of the functions as described in the final report. Project code is not engineered or designed. No documentation is provided.	Provided project code is functional. Project code performs most of the functions as described in the final report. Project code is thought out and engineered. Project code provides documentation and code comments (where appropriate).	Provided project code is functional. Project code performs most of the functions as described in the final report. Project code is thought out and engineered. Project code provides documentation and code comments (where appropriate).	Provided project code that is functional. Project code performs the functions as described in the final report. Project code is excellently engineered. Project code provided helpful documentation and code comments (where appropriate).
Component	**Criteria**	**No Attempt**	**Undeveloped**	**Developing**	**Approaching**	**Proficient**	**Exemplary**
Report	Problem statement and goal	Provided no response.	Provided an incoherent problem statement and goal with no connection to the project description.	Provided a somewhat coherent problem statement and Goal with little or misguided connections with the project description.	Provided a basic, understandable problem statement and goal that loosely connects with the project description.	Provided a clear problem statement and goal that directly connects with the project description.	Provided a focused problem statement and goal that distinctly and in meaningful ways connects with the project description.
	Setup	Provided no response.	Provided an incomplete explanation of the setup.	Provided a reasonable explanation of the setup, but missing steps.	Provided a complete explanation of the setup, but some steps are not working.	Provided a complete and understandable explanation of the setup. All steps are working.	Provided a complete and clear explanation of the setup. All steps are accurate and working correctly.

	Implementation Processes	Provided no response.	Provided an incomplete explanation and/or one or more of the major tasks may be missing.	Provided a general explanation and implementation . Inaccuracies may be present.	Provided a reasonable explanation and implementation . Steps may be illogical or missing.	Provided a logical explanation and implementation . Steps are logical, but may be disjointed or unrelated to one another.	Provided a sound explanation and implementation . All steps are accurate, related, and working correctly.
	Results	Provided no response.	Provided incorrect or fake results. The explanation is fully incorrect with no merit in approach or connection with the results.	Provides some results and/or unrelated results. The explanation is somewhat incorrect with little merit in approach or connection with the results.	Provides mostly correct results. The explanation is reasonable with some ambiguity.	Results are accurate. The explanation is logical with no ambiguity.	Results are accurate. The explanation is well-developed with strong justifications directly related to the implementation results.