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 Project 1: gRPC. The student's job is to implement the essential functions that enforce the client-centric consistency, specifically Monotonic Write consistency and Read your Write consistency of the replicated data in the bank.

Objectives

Students will be able to:

- Implement the essential functions that enforces the client-centric consistency.
- Enforce the Monotonic Write policy, which extends the implementation of previous interfaces.
- 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

- The basic templates for Customer and Branch processes are provided as “Customer.py” and “Branch.py” can be found within the project description in the course.
- Personal computer with 8 GB RAM or higher. Must be able to install virtual machines on this computer (VMware).

Directions

Part 1: Written Report

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

Using the provided Student Template_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?

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

Part 2: Project Code

The project code may be submitted as a .zip file of your source code or as a text file with a source repository link to your source code.

Major tasks:

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

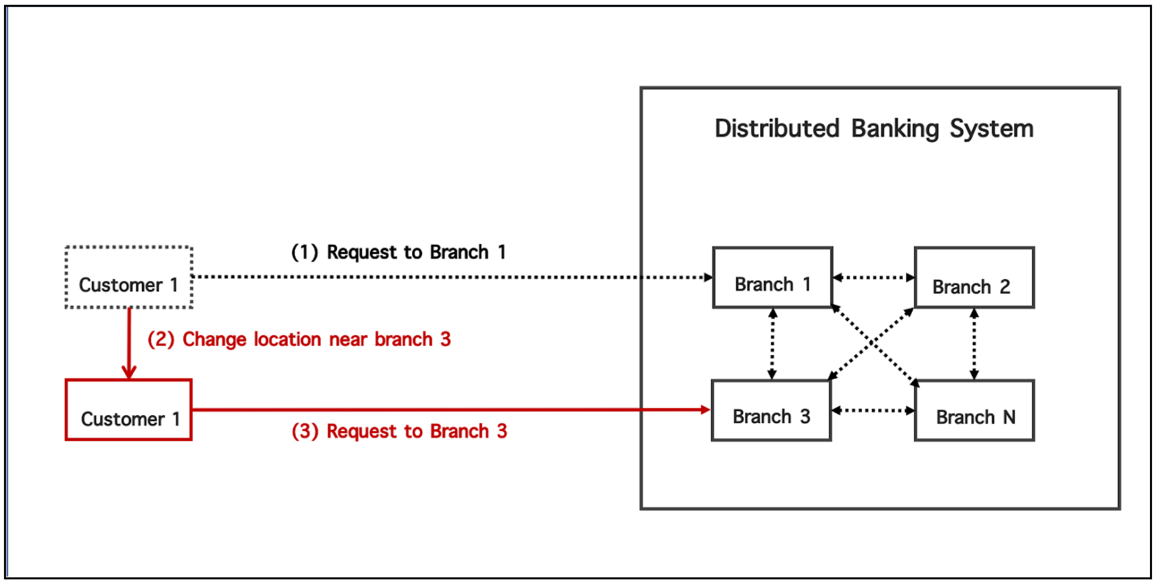

Diagram A: The customer changes the branches while submitting request to the Bank

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. Monotonic Writes

In many situations, it is important that write operations are propagated in the correct order to all copies of the data store. This property is expressed in monotonic-write consistency. In a monotonic-write consistency store, the following condition holds: A write operation by a process on a data item x is completed before any successive write operation on x by the same process.

Suppose that a client has an empty bank account and deposits $100 in location A. Suppose that the customer also withdraws while in location B. How and when the deposit of $100 in location A will be transferred to location B is left unspecified. In this case, if the deposit request has not yet been received by the server in location B, then the withdrawal request will fail.

2. Read your Writes

A data store is said to provide read-your-writes 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 the bank account has $100 and then withdraw $100. The customer issues sequential order of requests (deposit -> query -> withdrawal) 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.

2. Implementation

Unlike Project 1, where the customer communicates with only a specific branch with the same unique ID, Project 3 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.

3. Input and Output

The input file contains one Customer and multiple Branch processes. The format of the input file follows Project 1, 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 or a branch},
    "type" : “customer”,
    "events" :  [{"interface" : {query | deposit | withdraw}, "money" : {an integer value}, “id“ :
{unique identifier of an event}, “dest” : {a unique identifier of the branch} }]
  }
  { // Branch process #1
    "id" : {a unique identifier of a customer or a branch},
    “type” : “branch”
    “balance” : {replica of the amount of money stored in the branch}
  }
  { … } // Branch process #2
  { … } // Branch process #3
  { … } // Branch process #N
]
```

Output file format for the Monotonic Writes

```
[ // array of customers
   { // Customer process #1
     “id” : { a unique identifier of a customer #1  }
      “balance” {the final result of the balance for customer #1}
   }
]
```

Output file format for the Read your Writes

```
[ // array of customers
  { // Customer process #1
    “id” : { a unique identifier of a customer #1  }
     “balance” : [the list of results of the read operations performed by customer #1]

  }
]
```

1. Monotonic Writes Example

The test-cases for the Monotonic Write consistency implementation will generate Customer processes that perform write operations on different Branch processes.

Example of the input file

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

The customer will perform write operations to different Branch processes, and read the final result of the balance from the last Branch process that it requested the write operation. The balance of the customer should reflect the correct number of withdrawal and the deposits.

Expected output file:	**Wrong output file:**
[{"id": 1, "balance": 0 }]	[{"id" : 1, "balance" : 400 }]

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" :  [{"interface":"deposit", "money":400, "id":1, "dest":1 }, {"interface":"query", "id": 2, "dest": 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 operations 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, "balance": 400 }]	[{"id": 1, "balance": 0 }]

Submission Directions for Project Deliverables

You *must* submit your deliverables in the designated submission space in the course. Students may **not** email or use other means to submit the project for course team review and grading. Students should review the academic integrity and plagiarism policies prior to beginning this project.

Your Client-Centric Consistency Project includes **two (2)** deliverables. You may submit the code an unlimited number of times prior to the deadline and you may submit the written report one (1) time.

1. **Client-Centric Consistency Project Written Report:** Your written report must be a single PDF with the correct naming convention: Your Name_Client-Centric Consistency_Written Report.
2. **Client-Centric Consistency Project Code:** The project code may be submitted as a .zip file of your source code or as a text file with a source repository link to your source code.

Final submissions missing either of the deliverables will be graded based on what was submitted against the rubric criteria. Please review the rubric for how your Client-Centric Consistency Project will be graded.

There is an automatic 20% grade penalty for each day late past the deadline.

Rubric

Rubrics communicate specific criteria for evaluation. Prior to starting any graded coursework, you are expected to read through the rubric, so you know how you will be assessed. You are encouraged to self-assess your work and make informed revisions before submitting. Engaging in this learning practice will support you in developing your highest quality deliverables. *Points may be deducted at the discretion of the course team for disorganized submissions that convolute the grading process.*

Component	No Attempt	Undeveloped	Developing	Approaching	Proficient	Exemplary
Client-Centric Consistency Project Written Report Part 1: Determine the problem statement	Provided no response.	Provided an incoherent problem statement with no connection to the project description. Unclear understanding of the project.	Provided a somewhat coherent problem statement with little or misguided connections with the project description. It *may* demonstrate surface-level understanding of the project description, is too vague, or includes irrelevant information.	Provided a basic, understandable problem statement that loosely connects with the project description. It *may* demonstrate mid-level understanding of the project description with a reasonable approach.	Provided a clear problem statement that directly connects with the project description. Mid-high level understanding of the project description is demonstrated with an appropriate approach.	Provided a focused problem statement that distinctly and in meaningful ways connects with the project description. High-level understanding of the project description is demonstrated with a logical approach.
Client-Centric Consistency Project Written Report Part 1: Identify the goal of the problem statement	Provided no response.	Provided an inaccurate goal of the problem statement.		Provided a mostly accurate goal of the problem statement.		Provided a relevant and accurate goal of the problem statement.
Client-Centric Consistency Project Written Report Part 1: Explain the 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. [Example for students: Either of the policies are not enforced.]	Provided a reasonable explanation and implementation. Steps may be illogical or missing. [Example for students: Either of the policies are not enforced.]	Provided a logical explanation and implementation. Steps are logical, but may be disjointed or unrelated to one another. [Example for students: Both of the policies are not enforced, but there may be some inconsistency with the extension.]	Provided a sound explanation and implementation. All steps are accurate, related, and working correctly. [Example for students: Implemented the essential functions and enforced the client-centric consistency.]

Client-Centric Consistency Project Written Report Part 1: Explain implementation 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.
Component	**No Attempt**	**Undeveloped**	**Developing**	**Approaching**	**Proficient**	**Exemplary**
Project Code	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 some of the functions as described in the final report. Project code is thrown together. Sparse 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 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	**No Attempt**	**Undeveloped**	**Developing**	**Approaching**	**Proficient**	**Exemplary**
Professional Presentation	Provided no submission.	Did not use the provided template and much proofreading is needed. Little or no content-specific vocabulary is used or the attempt has staggering inaccuracies. Numerous imperfections are present and interfere with meaning.	Minimally utilized the provided template with little effectiveness in presentation and proofreading. Little or no content-specific vocabulary is used or the attempt has inaccuracies. Some imperfections are present and interfere with meaning.	Utilized the provided template with some effectiveness in presentation and proofreading. It may include some content-specific vocabulary. Few imperfections *may* be present, but may interfere with meaning.	Utilized the provided template with effective presentation and proofreading. Included relevant content-specific vocabulary. Minor imperfections *may* be present, but do not impede meaning.	Fully utilized the provided template with professional level presentation and proofreading. Included purposeful academic vocabulary. Imperfections are not present or are so slight, they are inconsequential.