Ira A. Fulton Schools of Engineering
Arizona State University

CSE 531: Distributed and Multiprocessor Operating Systems

# gRPC Project

## Purpose

The goal of this project is to build a distributed banking system that allows multiple customers to withdraw or deposit money from multiple branches in the bank. We assume that all the customers share the same bank account and each customer accesses only one specific branch. In this project, we also assume that there are no concurrent updates on the bank account. Each branch maintains a replica of the money that needs to be consistent with the replicas in other branches. The customer communicates with only a specific branch that has the same unique ID as the customer. Although each customer independently updates a specific replica, the replicas stored in each branch need to reflect all the updates made by the customer.

## Objectives

Learners will be able to:

- Define a service in a .proto file.
- Generate server and client code using the protocol buffer compiler.
- Use the Python gRPC API to write a simple client and server for your service.
- Build a distributed system that meets specific criteria.
- 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

## Helpful Resources

- gRPC Quick Start Guide

## Part 1: Written Report

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

Using the provided *Learner Template_CSE 531_Your Name_gRPC_Written Report*, compose a report addressing the questions:

1. What is the problem statement and goal?
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 results and their justifications?

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

## Part 2: Project Code

### Version Requirements for Libraries

grpcio==1.64.1

grpcio-tools==1.64.1

protobuf==5.27.2

### Major Tasks

1. Implement the customer and branch processes using Python processes
2. Implement the communication between the processes using gRPC

3. Implement the interfaces (Query, Deposit, and Withdraw) that are used between the customer processes and the branch processes
4. Implement the interfaces (Propogate_Deposit and Propogate_Withdraw) that are used between branch processes

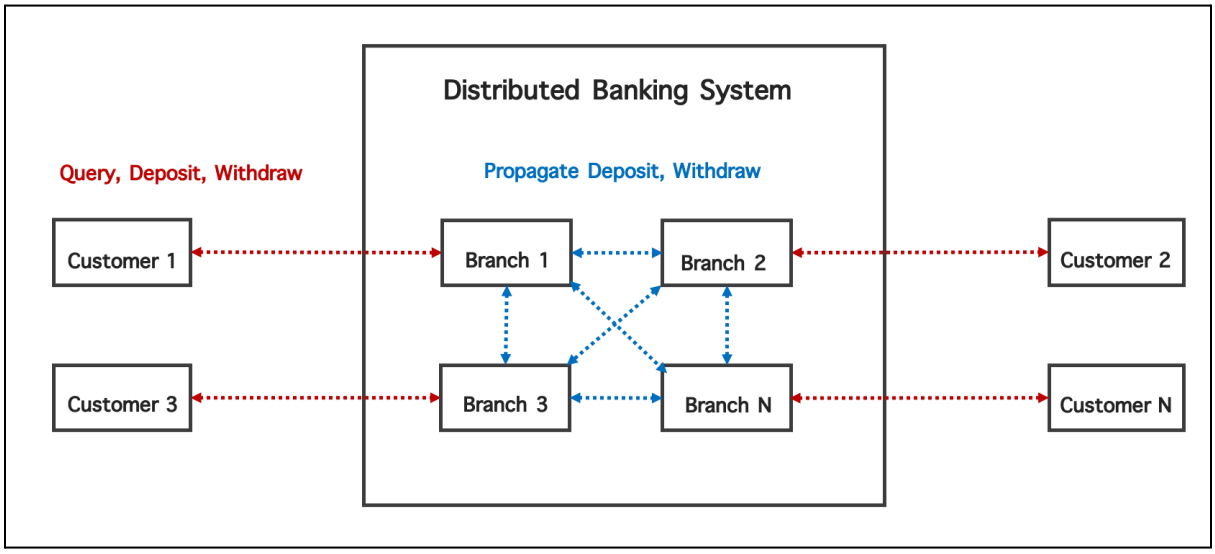


**Diagram A: Topology of the Banking System**

## 1. Description

In the first part of this project, you will implement an interface using gRPC so that the processes can communicate with each other.

### 1.1. gRPC

The communication is implemented with the RPC that we learned in the video lecture "Remote Procedure Call" from *Module 3: Communication and Coordination Part 1*. We will be using the simplest type of RPC in gRPC, where the customer sends a single request and gets back a single response.

1. The client process calls the method on the stub/client object, the server process is notified that the RPC has been invoked with the client's metadata for this call, the method name, and the specified deadline if applicable.
2. The server can then either send back its own initial metadata (which must be sent before any response) straight away, or wait for the client's request message - which happens first is application-specific.

3. Once the server has the client's request message, it does whatever work is necessary to create and populate its response. The response is then returned (if successful) to the client together with status details (status code and optional status message) and optional trailing metadata.

4. If the status is OK, the client then gets the response, which completes the call on the client side.

The gRPC Quick Start Guide explains the basic Python programmer's introduction to working with gRPC using a simple program. We strongly encourage you to refer to this resource.

### 1.2. Customer and Branch Processes

The basic templates for Customer and Branch processes are provided as "Customer.py" and "Branch.py". Learners are expected to complete the function Customer.createStub and Customer.executeEvents in the Customer.py and use it to generate Customer processes. Customers use the Customer.createStub to communicate with the Branch process with the ID identical to the Customer. Customer.executeEvents processes the events from the list of events stored in the Customer class and submits the request to the Branch process.

Additionally, learners are expected to complete the Branch.MsgDelivery of the Branch.py and use it to generate Branch processes. Branch.MsgDelivery processes the requests received from other processes and returns results to the requested process. Learners should complete the described functions and the Customer and Branch to complete this assignment.

### 1.3. Branch to Customer Interface

Branch.MsgDelivery receives the request from the Customer process. The Branch process should decrease or increase its amount of money, Branch.balance, according to the Customer's request. The customer stubs of the Branches are stored as a list in the Branch.stubList of the Branch class.

- ➔ **Branch.Query** interface reads the value from the Branch.balance and returns the result back to the Customer process.
- ➔ **Branch.Withdraw** interface decreases the Branch.balance with the amount specified in the Client's request, propagates the request to its fellow branches, and returns success / fail to the Customer process.
- ➔ **Branch.Deposit** interface increases the Branch.balance with the amount specified in the Client's request, propagates the request to its fellow branches, and returns success / fail to the Customer process.

### 1.4. Branch to Branch Interface

Branch.MsgDelivery receives the request from its fellow branches that propagate the updates. The Branch process should decrease or increase its replica, Branch.balance, according to the branches' request.

- ➔ **Branch.Propogate_Withdraw** interface decreases the Branch.balance with the amount specified in the Branch's request and returns success / fail to the Branch process.
- ➔ **Branch.Propogate_Deposit** interface increases the Branch.balance with the amount specified in the Branch's request and returns success / fail to the Branch process.

### 1.5. Input and Output

The input file contains a list of Customers and Branch processes. For each customer process, it specifies the list of events that it will execute. All the events will be executed in the order specified by their unique identifiers. For each branch process, it specifies its initial account balance, which should be the same across all branches.

You need to implement a main program to launch and execute the Customer and Branch processes according to the input file. In this project, we assume that there are no concurrent operations to the bank account. Therefore, the customers in fact execute sequentially one by one, and you can use a sleep command before launching a new Customer process to wait for all the operations performed by the previous customer on one branch to be propagated to the other branches.

```
[ // Start of the Array of Branch and Customer processes
  { // Customer process #1
    "id" : {a unique identifier of a customer},
    "type" : “customer”,
    "events" :  [{"interface":{deposit | withdraw}, "money": {an integer value}, “id“: {unique identifier of
an event} }, {"interface":{query}, “id“: {unique identifier of an event} } ]
  }
  { … } // Customer process #2
  { … } // Customer process #3
  { … } // Customer process #N
  { // 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
] // End of the Array of Branch and Customer processes
```

The output contains the list of successful responses that the Customer processes have received from the Branch processes.

```
[ // Start of the Array of Customer processes
   { // Customer process #1
     “Id” : { a unique identifier of a customer  }
     “recv” {a list of successful returns of the events from the Branch process}
   }
 { … } // Customer process #2
 { … } // Customer process #3
 { … } // Customer process #N
] // End of the Array of Customer processes
```

Below is an example input file and the expected output file. Note that the final query event sleeps for three seconds to guarantee all the updates are propagated among the Branch processes.

**Example of the input file**

```
[
  {
    "id" : 1,
    "type" : "customer",
    "events" :  [{ “id”: 1, "interface":"query" }]
  },
  {
    "id" : 2,
    "type" : "customer",
    "events" :  [{ “id”: 2, "interface":"deposit", "money": 170 },{ “id”: 3, "interface":"query" }]
  },
  {
    "id" : 3,
    "type" : "customer",
    "events" :  [{ “id”: 4, "interface":"withdraw", "money": 70 },{ “id”: 5, "interface":"query" }]
  },
  {
    "id" : 1,
    "type" : "branch",
    "balance" : 400
```

```
  },
  {
    "id" : 2,
    "type" : "branch",
    "balance" : 400
  },
  {
    "id" : 3,
    "type" : "branch",
    "balance" : 400
  }
]
```

**Expected output file**

```
[{"id": 1, "recv": [{"interface": "query", "balance": 400}]}
, {"id": 2, "recv": [{"interface": "deposit", "result": "success"}, {"interface": "query", "balance": 570}]},
 {"id": 3, "recv": [{"interface": "withdraw", "result": "success"}, {"interface": "query", "balance": 500}]}]
```

All the customer and branch processes should be terminated after all the events specified in the input file are executed.

# Formatting Specifications

Naming convention for 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: The 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 gRPC 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 gRPC Project Written Report includes one (1) deliverable:

- **gRPC Project Written Report:** Your written report must be a single PDF with the correct naming convention: *CSE 531_Your Name_gRPC_Written Report*.

## Part 2: Project Code

You are given an unlimited number of attempts to submit your best work. You must submit your gRPC 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: gRPC Project Code**".
2. Click the "**Load Submission…in new window**" button.
3. Once in Gradescope, select the project titled, "**Submission: gRPC 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 gRPC Project Code includes one (1) deliverable:

- **ZIP File:** Your ZIP file must contain your gRPC 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 output file will be checked that operations have been performed successfully. (Review the "Expected output file" in the project directions for reference.)

50 customers and 50 branches and 100 total operations in two (2) rounds:

- **Criteria 1:** In the first round, the returned balance should increase with the customer ID
- **Criteria 2:** In the second round, it should decrease with the customer ID.
- **Criteria 3:** The final balance should be the same as the initial balance, which is 400.

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

- About **20%** of the return values are correct.
- About **40%** of the returned balance values are correct and match the customer IDs.
- About **60%** of the returned balance values are correct and match the customer IDs.
- About **80%** of the returned balance values are correct and match the customer IDs.
- **All** of the returned balance values are correct and they match the customer IDs.

### 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 how many operations are correct. In the first round, the returned balance should increase with the customer ID; in the second round, it should decrease with the customer ID. The final balance should be the same as the initial balance, which is 400. | Provided no response. | About 20% of the return values are correct. | About 40% of the returned balance values are correct and match the customer IDs. | About 60% of the returned balance values are correct and match the customer IDs. | About 80% of the returned balance values are correct and match the customer IDs. | All of the returned balance values are correct and they match the customer IDs. |
| **Component** | **Criteria** | **No Attempt** | **Undeveloped** | **Developing** | **Approaching** | **Proficient** | **Exemplary** |
| **Code** | The code consists of the following components: Branch.Query, Branch.Withdraw, Branch.Deposit, Branch.Propogate_Withdraw, Branch.Propagate_Deposit | Provided no response. | Provided project code that is syntactically or semantically invalid. | Provided project code is functional.<br><br>Project code performs a few of the functions as described in the final report.<br><br>Project code is not engineered or designed.<br><br>No documentation is provided. | Provided project code is functional.<br><br>Project code performs most of the functions as described in the final report.<br><br>Project code is thought out and engineered.<br><br>Project code provides documentation and code comments (where appropriate). | Provided project code is functional.<br><br>Project code performs most of the functions as described in the final report.<br><br>Project code is thought out and engineered.<br><br>Project code provides documentation and code comments (where appropriate). | Provided project code that is functional.<br><br>Project code performs the functions as described in the final report.<br><br>Project code is excellently engineered.<br><br>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.<br><br>The explanation is fully incorrect with no merit in approach or connection with the results. | Provides some results and/or unrelated results.<br><br>The explanation is somewhat incorrect with little merit in approach or connection with the results. | Provides mostly correct results.<br><br>The explanation is reasonable with some ambiguity. | Results are accurate.<br><br>The explanation is logical with no ambiguity. | Results are accurate.<br><br>The explanation is well-developed with strong justifications directly related to the implementation results. |