**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 the multiple branches in the bank. We assume that there are no concurrent updates on the same resources (money) in the bank, and no customer accesses multiple branches. Each branch maintains a replica of the money that needs to be consistent with the replicas in other branches. customer only communicates with only a specific branch that has the same unique ID with 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**

Students 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
* 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](https://ets.engineering.asu.edu/vmware/)).

**Helpful Resources**

* [gRPC Quick Start Guide](https://grpc.io/docs/quickstart/python/)

**Directions**

**Part 1: Written Report**

Your written report must be a single PDF with the correct naming convention: Your Name\_gRPCF\_Written Report.

Using the provided Student Template\_Your Name\_gRPCF\_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?

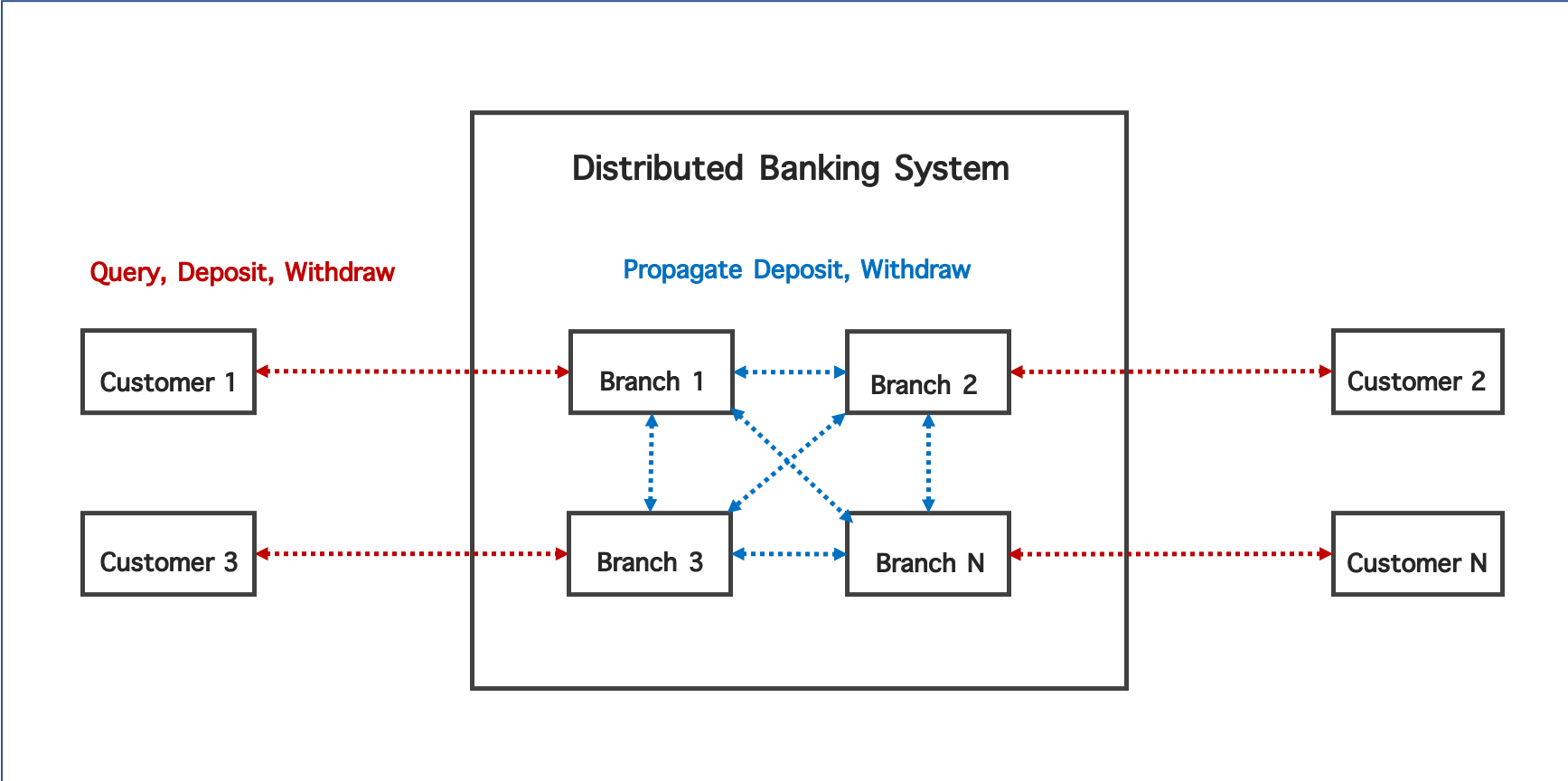
*\*Students may add subheadings on the template to purposefully call attention to specific, organized details.*

**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. 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**

### **Description**

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

### **gRPC**

The communication is implemented with the RPC that we have learned in our previous lecture (Unit 3 Module 2 Lecture 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.

[gRPC Quick Start Guide](https://grpc.io/docs/quickstart/python/) explains the basic Python programmer’s introduction to working with gRPC using a simple program. We strongly encourage you to refer to this resource.

### **Customer and Branch Processes**

The basic templates for Customer and Branch processes are provided as “Customer.py” and “Branch.py”. Students 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, students 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. Students should complete the described functions and the Customer and Branch to complete this assignment.

### **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** interfaceincreases 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 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.

**Input and Output Formats**

The input file contains a list of Customers and Branch processes. The number of customers should be identical to the number of branch processes. The list of events that will be stored in Customer.events is described within the Customer’s description. Every Customer.event contains a Query event to check the consistency of the replica. The Branch.balance is described within the Branch's description. The initial value of the Branch.balance is identical among all the descriptions of the Branch processes.

|  |
| --- |
| [ // Start of the Array of Branch and Customer 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} } ]  }  { … } // Customer process #2   { … } // Customer process #3  { … } // Customer process #N  { // 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  ] // End of the Array of Branch and Customer processes |

The output contains the list of successful responses from the Branch process. All the requested events should return successfully back to the Customer, and the “money” value from the final query request of each Customer should be identical to pass the assignment.

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

|  |  |
| --- | --- |
| **Expected output file:** | |
| **Expected output file:**  {'id': 1, 'recv': [{'interface': 'query', 'result': 'success', 'money': 500}]}  {'id': 2, 'recv': [{'interface': 'deposit', 'result': 'success'}, {'interface': 'query', 'result': 'success', 'money': 500}]}  {'id': 3, 'recv': [{'interface': 'withdraw', 'result': 'success'}, {'interface': 'query', 'result': 'success', 'money': 500}]} | |

**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 gRPC Project includes **two (2)** deliverables:

1. **gRPC Project Written Report:** Your written report must be a single PDF with the correct naming convention: Your Name\_gRPC\_Written Report.
2. **gRPC 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 gRPC 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** |
| **gRPC 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. |
| **gRPC 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. |
| **gRPC 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: Implemented the processes but with inaccurate communication using gprc.] | Provided a reasonable explanation and implementation. Steps may be illogical or missing.    [Example for students: Communication is established but interface(Query, Deposit, and Withdraw) isn’t established between customer and branch process] | Provided a logical explanation and implementation. Steps are logical, but may be disjointed or unrelated to one another.  [Example for students: Incorrect implementation of the interface(Propogate\_Deposit and Propogate\_Withdraw) between branch processes.] | Provided a sound explanation and implementation. All steps are accurate, related, and working correctly. |
| **gRPC 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. |