RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Assignment

Course Code	CSE301A
Course Name	Distributed Systems
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	Satyajit Ghana
Reg. No.	17ETCS002159
Semester/Year	06/2020
Course Leader(s)	Chaitra S.

Declaration Sheet

Student Name	Satyajit Ghana		
Reg. No	17ETCS002159		
Programme	B.Tech	Semester/Year	06/2020
Course Code	CSE301A		
Course Title	Distributed Systems		
Course Date		to	
Course Leader	Chaitra S.		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Contents

List Of Figures

1 Question 1

Solution to Part A

1.1 Design of Banking Application

1.1.1 gRPC

Since we need to have multiple clients to be connected to the server and working at real-time data, also being very high performant at the same time, a banking application must be robust, secure and responsive at the same time, we chose gRPC to be the best solution for this.

gRPC is a high performance, open-source universal RPC framework, developed by Google. In gRPC, a client application *can directly call methods on a server application on a different machine as if it was a local object*, making it easier to create distributed applications and services. Google's RPC framework and the protocol itself is built on http2. *It has many advantages such as the ability to enable different languages to interact with each other via gRPC calls.*

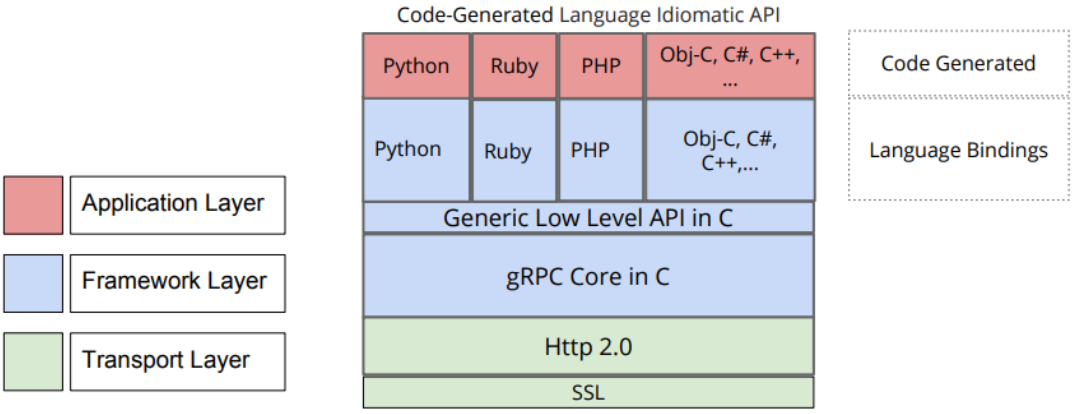

Figure 1 gRPC C Core and Wrapped Language Stack (Google)

gRPC uses Protocol Buffers for serialization/deserialization of objects and data structures sent between clients and servers. Developers need to define their data structures in .proto files and auto generated code provides functions that allow for these structures to be serialized/deserialized.

The gRPC diagram below describes how different clients running of different operating systems and using a different programming language, can connect to the gRPC server and call a remote procedure. This allows for ***heterogeneity*** in the system.

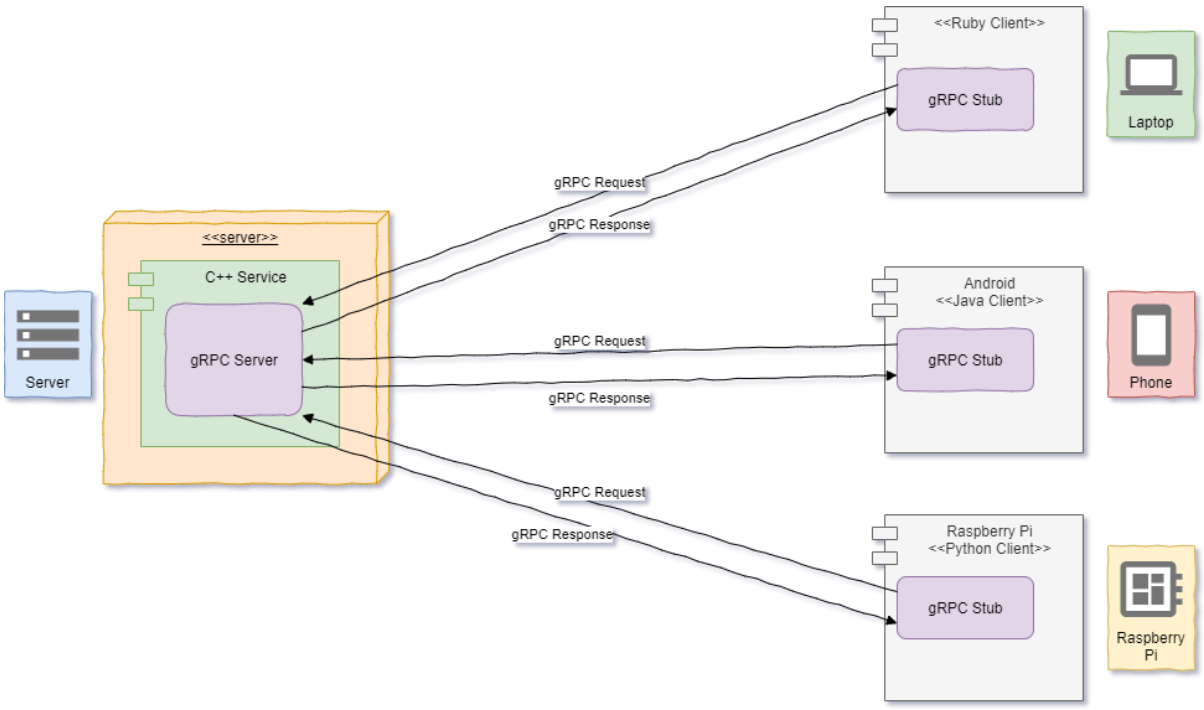

Figure 2 gRPC Deployment Diagram

The Deployment Diagram for our Bank Server and Bank Client is as below,

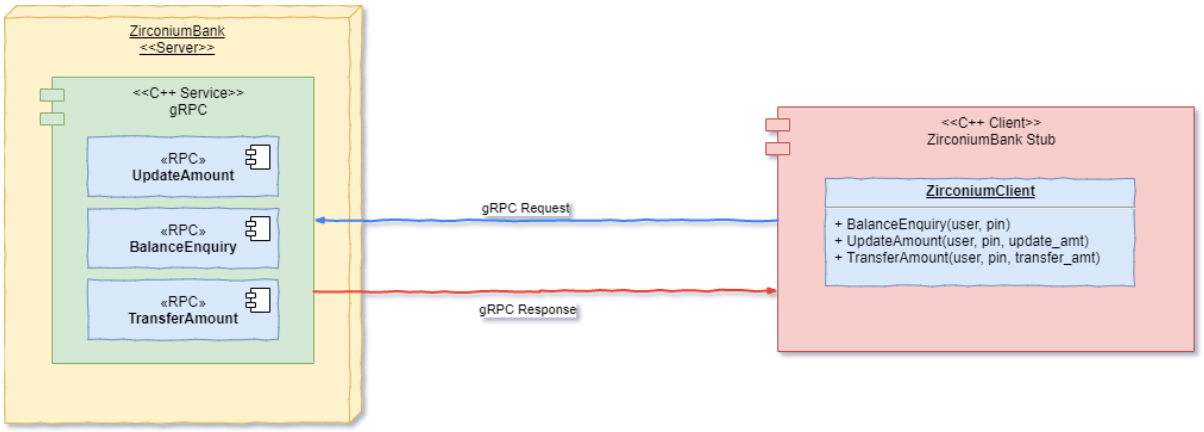

Figure 3 ZrB Deployment Diagram

1.1.2 MongoDB Transactions

The Database chosen for the application was MongoDB, since it supports *distributed transactions*, and handles then very efficiently, also it follows *ACID properties* which is essential for our application.

MongoDB uses multi-granularity locking that allows operations to lock at the global, database or collection level, and allows for individual storage engines to implement their own concurrency control below the collection level (e.g., at the document-level in WiredTiger).

MongoDB uses reader-writer locks that allow concurrent readers shared access to a resource, such as a database or collection. In addition to a shared (S) locking mode for reads and an exclusive (X) locking mode for write operations, intent shared (IS) and intent exclusive (IX) modes indicate an intent to read or write a resource using a finer granularity lock. When locking at a certain granularity, all higher levels are locked using an intent lock.

For example, when locking a collection for writing (using mode X), both the corresponding database lock and the global lock must be locked in intent exclusive (IX) mode. A single database can simultaneously be locked in IS and IX mode, but an exclusive (X) lock cannot coexist with any other modes, and a shared (S) lock can only coexists with intent shared (IS) locks.

Locks are fair, with reads and writes being queued in order. However, to optimize throughput, when one request is granted, all other compatible requests will be granted at the same time, potentially releasing them before a conflicting request. For example, consider a case in which an X lock was just released, and in which the conflict queue contains the following items:

`IS → IS → X → X → S → IS`

(MongoDB Documentation)

Because a single document can contain related data that would otherwise be modeled across separate parent-child tables in a relational schema, MongoDB's atomic single-document operations already provide transaction semantics that meet the data integrity needs of the majority of applications. One or more fields may be written in a single operation, including updates to multiple sub-documents and elements of an array. The guarantees provided by MongoDB ensure complete isolation as a document is updated; any errors cause the operation to roll back so that clients receive a consistent view of the document.

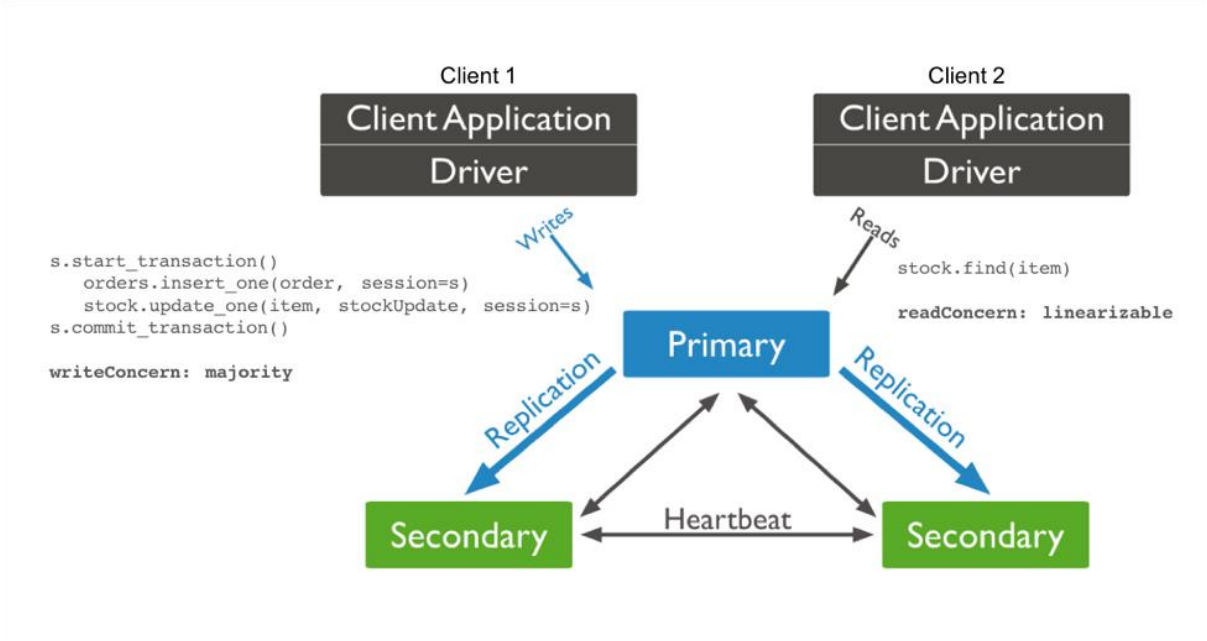

Figure 4 Wired Tiger Transactions

Distributed transactions refer to multi-document transactions on sharded clusters and replica sets. Multi-document transactions (whether on sharded clusters or replica sets) are also known as distributed transactions

1.2 Implementation of Banking Application

The complete code has been attached in Appendix A and Appendix B.

To start with the server/client we first need to define our service and our protocol buffer messages so the server/client know how to *marshal* and *de-marshal* the data.

```
package zirconium.bank;

// Interface exported by the server
// Here we have 3 remote procedures that can be called
// UpdateAmount, BalanceEnquiry and TransferAmount with their respective arguments
// and return messages
service WebService {
    rpc UpdateAmount(RequestAmount) returns (Balance) {}
    rpc BalanceEnquiry(AuthUser) returns (Balance) {}
    rpc TransferAmount(TransferReq) returns (Balance) {}
}

message TransferReq {
    AuthUser from_user = 1;
    string to_user = 2;
    int32 amount = 3;
```

```
}

message AuthUser {
    string username = 1;
    string pin = 2;
}

message RequestAmount {
    AuthUser user = 1;
    int32 value = 2;
}

message Balance {
    int32 value = 1;
}
```

Now that the protocol buffers are defined, we then need to implement our service at the backend, using any of the supported language of gRPC, We've chosen C++ to make our backend, since it is very robust and high performant, which is very essential for a Bank Server.

All of the Code has been attached at the Appendix, here we'll discuss the important aspects of the server.

All of our code is multithreaded, gRPC takes care of it for us, any client that calls a remote procedure is assigned their own threaded procedure call. Our MongoDB also supports multithreading, so we need to create a multi-threaded connection pool, then we can request for a client from this pool and then start working on the data.

This method will run the service and expose a port so that clients can connect to it,

```cpp
void RunServer(const std::string& server_addr, const std::string& mongo_uri) {
    WebServiceImpl service;

    // initialize the service with mongo db name "zirconium"
    service.initiaize(mongo_uri);

    ServerBuilder builder;

    // Listen on the given address without any authentication mechanism.
    builder.AddListeningPort(server_addr, grpc::InsecureServerCredentials());

    // Register "service" as the instance through which we'll communicate with
    // clients. In this case it corresponds to an *synchronous* service.
    builder.RegisterService(&service);

    // Finally assemble the server.
    std::unique_ptr<Server> server(builder.BuildAndStart());
    LOG_S(INFO) << "Server listening on " << server_addr;
```

```
    // Wait for the server to shutdown. Note that some other thread must be
    // responsible for shutting down the server for this call to ever return.
    server->Wait();
}
```

MongoDB Connection Pool

```
    void initiaize(const std::string& mongo_uri) {
        conn_pool_ = std::make_unique<mongocxx::pool>(mongocxx::uri{mongo_uri});
    };
```

Let's look at one of the RPC method TransferAmount to see how it works

The Basic Transaction here is:

1. START TRANSACTION
2. curr_to_user.amount = Read(to_user)
3. curr_from_user.amount = Read(from_user)
4. check if to_user exists
5. check if from_user has sufficient balance
6. curr_from_user.balance -= amount
7. curr_to_user.balance += amount
8. Write(to_user)
9. Write(from_user)
10. COMMIT TRANSACTION

```
1. /**
2. Transfer Amount: Transfer amount from current user to to_user
3. Tested Works ✅
4. */
This is the TransferAmount RPC method, any client can call this with the appropriate arguments
5. Status TransferAmount(ServerContext* context, const TransferReq* transfer_req, Balance* balance) o
   verride {
6. LOG_S(INFO) << "TransferAmount called by peer: " << context->peer();

7. // get a client from the pool
8. auto client = conn_pool_->acquire();

9. // get the "zirconium" db
10. auto db = (*client)["zirconium"];

Now we bind the data received from the client to local variables
11. // curr_user
12. const auto auth_user = transfer_req->from_user();
13. const auto to_user = transfer_req->to_user();
```

```
14. // guard to check if the to_user is same as curr_user
15. if (to_user == auth_user.username()) {
16. return Status(StatusCode::FAILED_PRECONDITION, "to_user is same as curr_user");
17. }

18. logAuthData_(auth_user.username(), auth_user.pin());

19. try {
20. auto [is_authenticated, user] = getAuthData_(auth_user.username(), auth_user.pin(), (*client));

21. // check if the user is authenticated
22. if (not is_authenticated) {
23. LOG_S(WARNING) << "UNAUTHENTICATED User: " << auth_user.username();
24. return Status(StatusCode::UNAUTHENTICATED, "username or pin wrong!");
25. }

26. // the user is authenticated

This function is our main function, this is the actual transaction that will take place when this RPC
is executed
27. // transaction function
28. auto transfer_user_amount = [&](mongocxx::client_session& session) {
29. // get the transaction options
30. auto txn_opts = zirconium::transaction_mgr::get_transaction_opts();

This function call will start the transaction with the current transaction options
31. // initiate the transaction
32. session.start_transaction(txn_opts);

33. // container for the updated user
34. bsoncxx::stdx::optional<bsoncxx::document::value> updated_from_user;

Transaction begins from here
35. // try the transaction
36. try {
37. // read to_user value from db
38. auto curr_to_user = db["bank"].find_one(session, make_document(kvp("username", to_user)));

39. // this is unnecessary, can be ommited
40. if (not curr_to_user) {
41. // to_user does not exist, abort transaction
42. throw zirconium::exception("to_user does not exist");
43. }

44. // read from_user value from db
45. auto curr_from_user = db["bank"].find_one(session, make_document(kvp("username", auth_user.usernam
    e())));
46. auto curr_from_user_balance = curr_from_user->view()["balance"].get_int32().value;

47. // guard to check if from_user has necessary balance
48. if (curr_from_user_balance - transfer_req->amount() < 0) {
49. // insufficient balance
50. throw zirconium::exception("insufficient balance");
51. }
```

```
52. // decrement the balance of from_user
53. updated_from_user = db["bank"].find_one_and_update(
54. session,
55. curr_from_user->view(),
56. make_document(
57. kvp("$inc", make_document(
58. kvp("balance", -transfer_req->amount()))),
59. mongocxx::options::find_one_and_update().return_document(mongocxx::options::return_document::k_aft
    er));

60. // increment the balance of to_user
61. db["bank"].find_one_and_update(
62. session,
63. curr_to_user->view(),
64. make_document(
65. kvp("$inc", make_document(
66. kvp("balance", transfer_req->amount()))),
67. mongocxx::options::find_one_and_update());

68. // set the new balance as responce
69. balance->set_value(updated_from_user->view()["balance"].get_int32());

70. } catch (const mongocxx::operation_exception& oe) {
71. session.abort_transaction();
72. throw oe;
73. } catch (const zirconium::exception& ex) {
74. session.abort_transaction();
75. throw ex;
76. }

77. // run this if the transaction was successful
78. auto on_success = [&](void) {
79. LOG_S(INFO) << "UPDATED: " << bsoncxx::to_json(*updated_from_user) << "\n";
80. auto update_type = transfer_req->amount() >= 0 ? "credited" : "debited";

81. LOG_S(INFO) << "UpdateAmount [ Account " << updated_from_user-
    >view()["username"].get_utf8().value.to_string() << " " << update_type << " with ₹ " << std::abs(t
    ransfer_req->amount()) << " new balance: " << updated_from_user-
    >view()["balance"].get_int32() << " ]";
82. };

83. // commit the transaction and retry if failure
84. zirconium::transaction_mgr::commit_with_retry(session, on_success);
85. };

86. // create a session and start a transaction
87. auto session = client->start_session();

88. try {
89. zirconium::transaction_mgr::run_transaction_with_retry(transfer_user_amount, session);
90. } catch (const mongocxx::operation_exception& oe) {
91. // some expection occured during commit
```

```
92. LOG_S(ERROR) << "Error during commit: " << oe.what() << ", for session: " << bsoncxx::to_json(sess
    ion.id());
93. }
94. } catch (zirconium::exception& ex) {
95. LOG_S(ERROR) << "Precondition failed: " << ex.what();
96. return Status(StatusCode::FAILED_PRECONDITION, ex.what());

97. } catch (const std::exception& ex) {
98. LOG_S(ERROR) << "Internal Server Error: " << ex.what();
99. return Status(StatusCode::INTERNAL, "Internal Server Error, Cannot Transfer Amount");

100.    } catch (...) {
101.    // something notorious has happened if you reach here
102.    // stop the server
103.    LOG_S(FATAL) << "shutting down";
104.    }

105.    return Status::OK;
106.    }
```

Transactions Manager Code

```
inline auto commit_with_retry(client_session& session, on_success_func succ_fun) {
    while (true) {
        try {
            session.commit_transaction();  // Uses write concern set at transaction start.
            LOG_S(INFO) << "Transaction commited : " << bsoncxx::to_json(session.id());
            succ_fun();
            break;
        } catch (const operation_exception& oe) {
            // Can retry commit
            if (oe.has_error_label("UnknownTransactionCommitResult")) {
                LOG_S(WARNING) << "UnknownTransactionCommitResult, retrying commit operation for sessi
on: " << bsoncxx::to_json(session.id());
                continue;
            } else {
                throw oe;
            }
        }
    }
};

using transaction_func = std::function<void(client_session& session)>;

inline auto run_transaction_with_retry(transaction_func txn_func, client_session& session) {
    while (true) {
        try {
            txn_func(session);  // performs transaction.
            break;
        } catch (const operation_exception& oe) {
```

```
            LOG_S(ERROR) << "Transaction aborted. Caught exception during transaction: " << oe.what()
<< ", for session: " << bsoncxx::to_json(session.id());
            // If transient error, retry the whole transaction.
            if (oe.has_error_label("TransientTransactionError")) {
                LOG_S(WARNING) << "TransientTransactionError, retrying transaction for session: " << b
soncxx::to_json(session.id());
                continue;
            } else {
                throw oe;
            }
        }
    }
};
```

Running the application

```
 /mnt/d/University-Work/University-Work-SEM-06/Assignment/DS/bankingapp  master ⇡2 !16 ?15
./bazel-bin/server/zirconium_server -h
Server for Zirconium Bank
Usage:
  zirconium_server [OPTION...]

      --config arg  config file
  -h, --help        print help
```

Figure 5 server help option

```
 /mnt/d/University-Work/University-Work-SEM-06/Assignment/DS/bankingapp  master ⇡2 !16 ?15
./bazel-bin/client/zirconium_client -h
A client for Zirconium Bank
Usage:
  zirconium_client [OPTION...]

      --config arg       config file
      --username arg     username
      --pin arg          PIN
      --balance_enquiry  Balance Enquiry
  -d, --debug            enable debugging
  -h, --help             print help

 deposit options:
      --deposit [=arg(=0)]  Deposit a given amount

 money transfer options:
      --to_user arg     username of receiver party
      --amt [=arg(=0)]  amount to be transferred

 withdraw options:
      --withdraw [=arg(=0)]  Withdraw a given amount


 /mnt/d/University-Work/University-Work-SEM-06/Assignment/DS/bankingapp  master ⇡2 !16 ?15
```

Figure 6 client help option

1.3 Testing of developed Banking Application

Let's see how the server works with only single operation, then we can move on to more rigorous testing,

Client Logs:

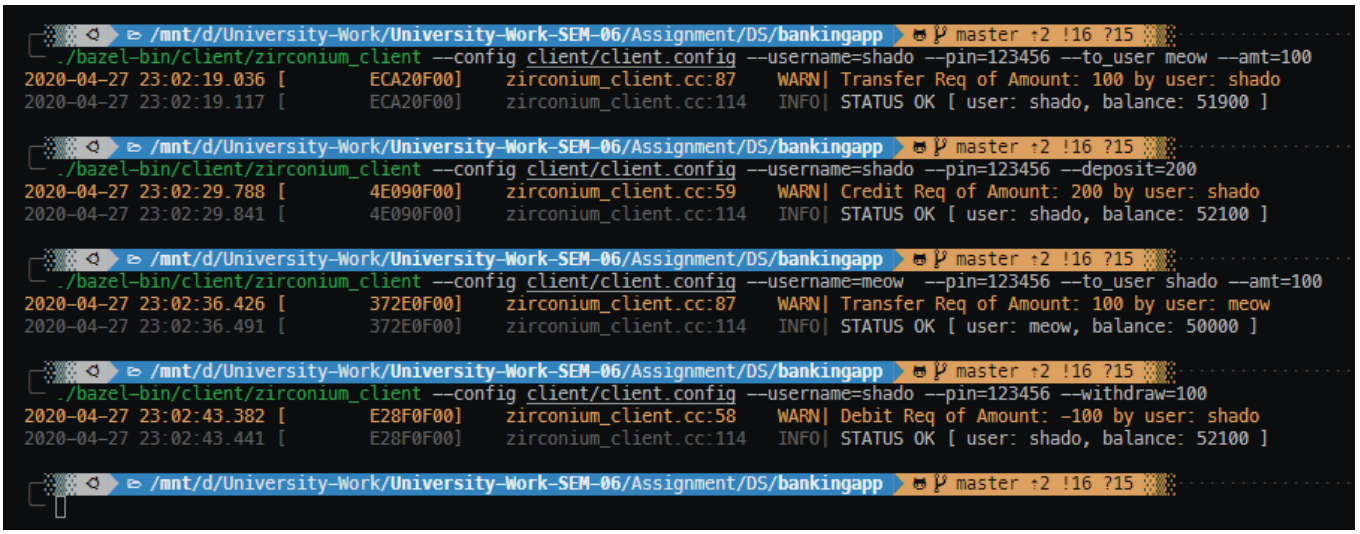

Figure 7 client logs

Server Logs:

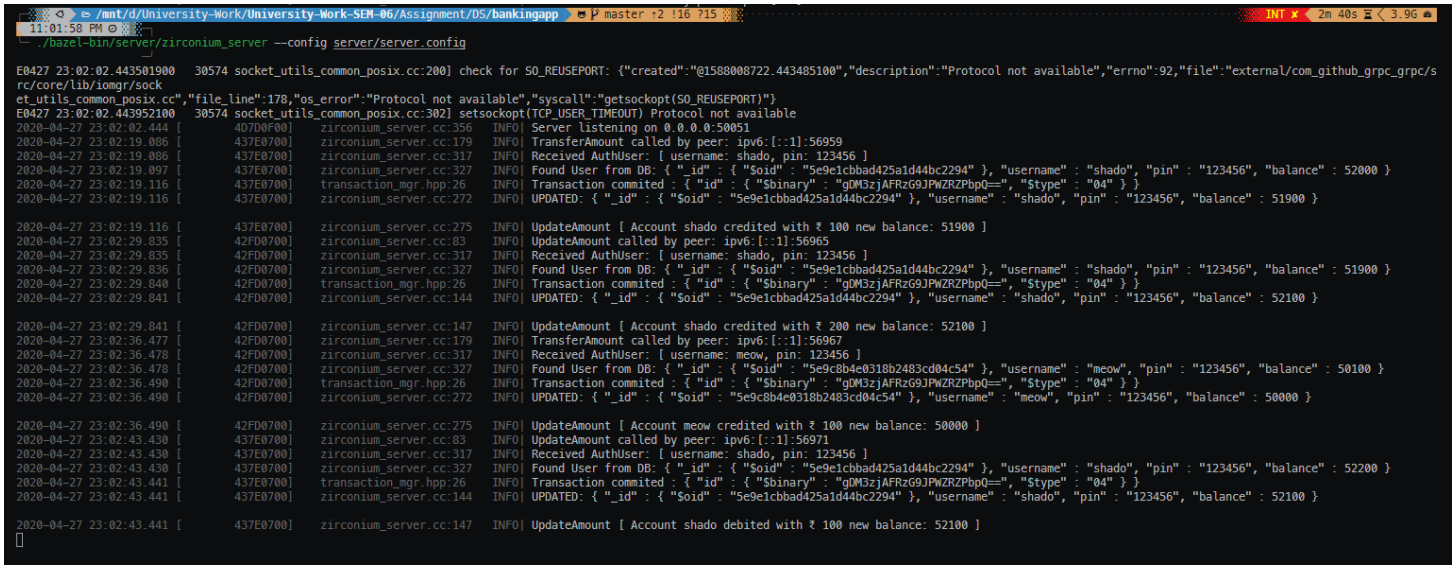

Figure 8 server logs

As we can see the transactions went through successfully, and the balance was updated as intended, also there were no transaction conflicts since we are running the clients serially.

To test the application, we've created two users and done concurrent transactions between them, i.e.

TEST Algorithm

1. Transfer Rs.100 from shadow to meow
2. Deposit Rs. 200 to shadow
3. Transfer Rs.100 from meow to shadow
4. Withdraw Rs. 100 from shadow

At the end of one such instance of this algorithm shado's account will get gain of Rs. 100, so let's call this Rs 100 shado gain test

All the above operations must run parallelly, so we made use of Linux's Background processes and GNU Parallel,

Here we do Run the Test Algorithm, everything parallelly, a total of 10 Times, At the end we expect that shadow's account must have a gain of Rs.1000 while meow's account's balance should remain same.

Initially we start with these values,

Figure 9 DB before transaction

Now we run the transaction test, 10 times

```
seq 10 | parallel -j 10 --workdir $PWD ./test/test_transactions.sh {}
```

Client Logs:

```
/mnt/d/University-Work/University-Work-SEM-06/Assignment/DS/bankingapp  master ↑2 !15 ?15
└─ seq 10 | parallel -j 10 --workdir $PWD ./test/test_transactions.sh {}
Academic tradition requires you to cite works you base your article on.
When using programs that use GNU Parallel to process data for publication
please cite:

  O. Tange (2011): GNU Parallel - The Command-Line Power Tool,
  ;login: The USENIX Magazine, February 2011:42-47.

This helps funding further development; AND IT WON'T COST YOU A CENT.
If you pay 10000 EUR you should feel free to use GNU Parallel without citing.

To silence this citation notice: run 'parallel --citation'.

2020-04-27 22:58:10.806 [        1F9D0F00]    zirconium_client.cc:87    WARN| Transfer Req of Amount: 100 by user: shado
2020-04-27 22:58:10.813 [        A16C0F00]    zirconium_client.cc:59    WARN| Credit Req of Amount: 200 by user: shado
2020-04-27 22:58:10.814 [        9CB80F00]    zirconium_client.cc:87    WARN| Transfer Req of Amount: 100 by user: meow
2020-04-27 22:58:10.817 [        11270F00]    zirconium_client.cc:58    WARN| Debit Req of Amount: -100 by user: shado
2020-04-27 22:58:10.888 [        1F9D0F00]    zirconium_client.cc:114   INFO| STATUS OK [ user: shado, balance: 49900 ]
2020-04-27 22:58:10.912 [        11270F00]    zirconium_client.cc:114   INFO| STATUS OK [ user: shado, balance: 50000 ]
2020-04-27 22:58:10.916 [        A16C0F00]    zirconium_client.cc:114   INFO| STATUS OK [ user: shado, balance: 50100 ]
2020-04-27 22:58:10.832 [        E94F0F00]    zirconium_client.cc:87    WARN| Transfer Req of Amount: 100 by user: shado
2020-04-27 22:58:10.841 [        227B0F00]    zirconium_client.cc:59    WARN| Credit Req of Amount: 200 by user: shado
2020-04-27 22:58:10.848 [        594A0F00]    zirconium_client.cc:87    WARN| Transfer Req of Amount: 100 by user: meow
2020-04-27 22:58:10.858 [        6F7D0F00]    zirconium_client.cc:58    WARN| Debit Req of Amount: -100 by user: shado
2020-04-27 22:58:10.918 [        E94F0F00]    zirconium_client.cc:114   INFO| STATUS OK [ user: shado, balance: 49900 ]
2020-04-27 22:58:10.924 [        227B0F00]    zirconium_client.cc:114   INFO| STATUS OK [ user: shado, balance: 50100 ]
2020-04-27 22:58:10.933 [        594A0F00]    zirconium_client.cc:114   INFO| STATUS OK [ user: meow, balance: 50100 ]
2020-04-27 22:58:10.935 [        6F7D0F00]    zirconium_client.cc:114   INFO| STATUS OK [ user: shado, balance: 50100 ]
2020-04-27 22:58:10.885 [        1DDE0F00]    zirconium_client.cc:87    WARN| Transfer Req of Amount: 100 by user: shado
2020-04-27 22:58:10.912 [        69230F00]    zirconium_client.cc:58    WARN| Debit Req of Amount: -100 by user: shado
2020-04-27 22:58:10.915 [        7B640F00]    zirconium_client.cc:59    WARN| Credit Req of Amount: 200 by user: shado
2020-04-27 22:58:10.919 [        4CDF0F00]    zirconium_client.cc:87    WARN| Transfer Req of Amount: 100 by user: meow
2020-04-27 22:58:10.963 [        1DDE0F00]    zirconium_client.cc:114   INFO| STATUS OK [ user: shado, balance: 50000 ]
2020-04-27 22:58:10.900 [        CDBB0F00]    zirconium_client.cc:87    WARN| Transfer Req of Amount: 100 by user: shado
2020-04-27 22:58:10.912 [        E15D0F00]    zirconium_client.cc:58    WARN| Debit Req of Amount: -100 by user: shado
2020-04-27 22:58:10.920 [        53FC0F00]    zirconium_client.cc:87    WARN| Transfer Req of Amount: 100 by user: meow
2020-04-27 22:58:10.926 [        2A190F00]    zirconium_client.cc:59    WARN| Credit Req of Amount: 200 by user: shado
2020-04-27 22:58:10.939 [        82AF0F00]    zirconium_client.cc:87    WARN| Transfer Req of Amount: 100 by user: shado
2020-04-27 22:58:10.963 [        75220F00]    zirconium_client.cc:59    WARN| Credit Req of Amount: 200 by user: shado
2020-04-27 22:58:10.971 [        4CC60F00]    zirconium_client.cc:58    WARN| Debit Req of Amount: -100 by user: shado
2020-04-27 22:58:10.971 [        18AE0F00]    zirconium_client.cc:87    WARN| Transfer Req of Amount: 100 by user: meow
2020-04-27 22:58:10.990 [        BA560F00]    zirconium_client.cc:59    WARN| Credit Req of Amount: 200 by user: shado
```

Figure 10 client logs

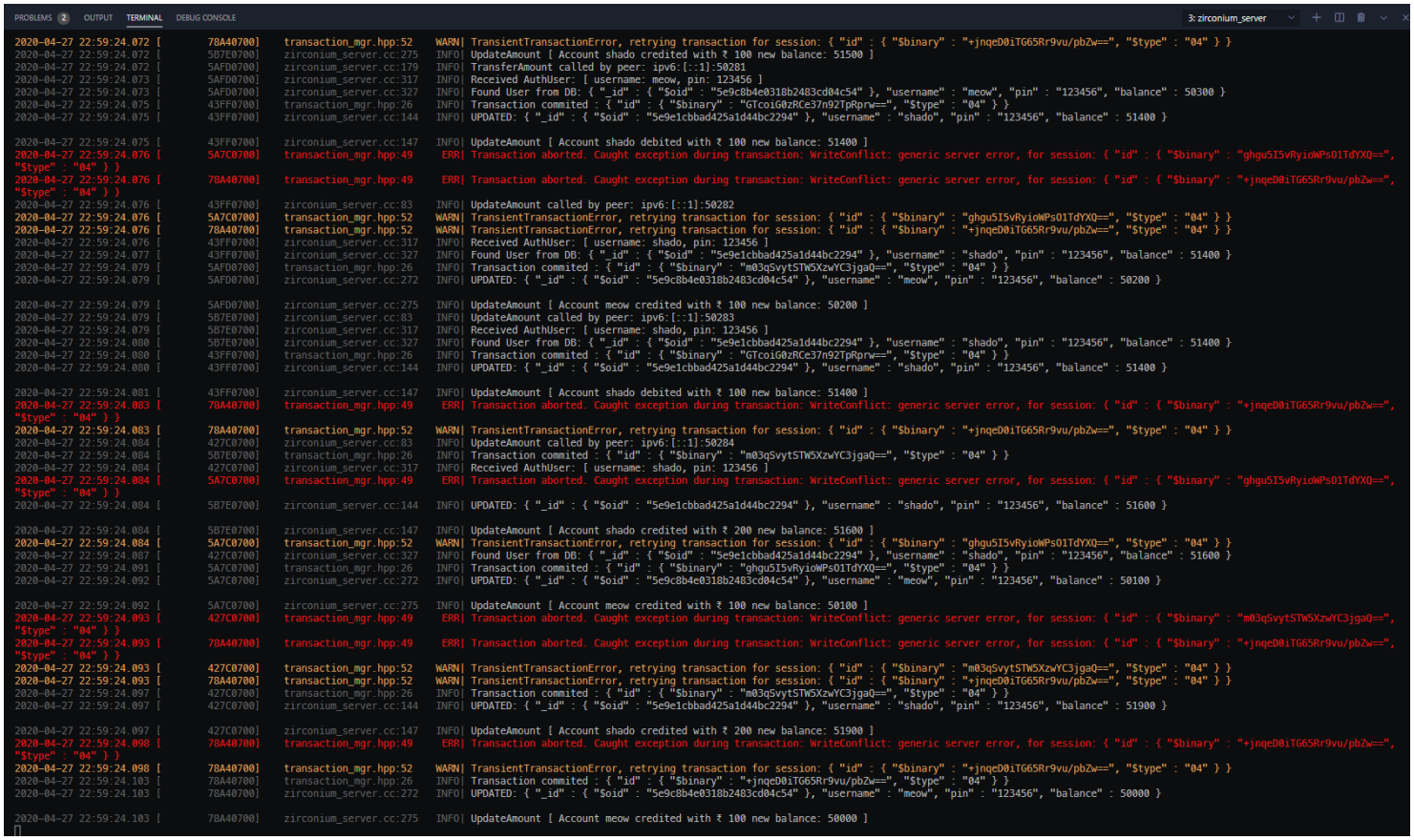

Figure 11 server logs

Server Logs:

What's really interesting to observe is that, since we are performing concurrent transactions on a distributed database, there are many conflicts, here our server takes really good care of them, by detecting such conflicts and quickly retrying the transaction again.

DB after running the test

Figure 12 DB after transaction

As expected, we have shado's account with gain of Rs. 1000, this makes sure that our transactions are working as expected, now let's try to run our test operation 100 times, with 10 parallel threads, before this we reset the accounts to have Rs 50,000 each.

Figure 13 DB after transaction

As expected now we have Rs 10,000 gain in shado's account as we ran the Rs 100 gain test a 100 times.

2 Question 2

Solution to Part B

2.1 RPC, the need for it and two sample programs

The main usage scenarios of RPC are

- Efficiently connecting polyglot services in microservices style architecture
- Connecting mobile devices, browser clients to backend services
- Generating efficient client libraries

Remote Procedure Calls (RPC)

1. Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.
2. Stubs – client-side proxy for the actual procedure on the server.
3. The client-side stub locates the server and marshalls the parameters.
4. The server-side stub receives this message, unpacks the marshalled parameters, and peforms the procedure on the server

The main goal of RPC is to hide the existence of the network from a program. As a result, RPC doesn't quite fit into the OSI model:

The message-passing nature of network communication is hidden from the user. The user doesn't first open a connection, read and write data, and then close the connection. Indeed, a client often doesn not even know they are using the network!

RPC often omits many of the protocol layers to improve performance. Even a small performance improvement is important because a program may invoke RPCs often. For example, on (diskless) Sun workstations, every file access is made via an RPC.

2.1.1 Example 1

Here we've made a simple Streaming NodeJS gRPC Client and Server, where the Server ingests weather data from a stored file and streams the data to a client when the procedure is called.

server.js

```
var PROTO_PATH = 'weather.proto';
```

```
var fs = require('fs');
var _ = require('lodash');
var grpc = require('grpc');
var protoLoader = require('@grpc/proto-loader');
var packageDefinition = protoLoader.loadSync(
    PROTO_PATH,
    {
        keepCase: true,
        longs: String,
        enums: String,
        defaults: true,
        oneofs: true
    });

var weather_proto = grpc.loadPackageDefinition(packageDefinition).weather;

let rawdata = fs.readFileSync('city_temps.json');
let city_data = JSON.parse(rawdata);

/**
 * Implements the GetWeather Service
 */
function getWeather(call) {
    var cityName = call.request.city_name;
    console.log('GetWeather Called by: ' + call.metadata._internal_repr['user-agent']);

    _.each(city_data, function (data) {
        data_res = {
            timestamp: data.timestamp,
            city: cityName,
            temp: data.temp,
            weather_status: "cloudy",
        }
        call.write(data_res);
    });

    call.end();
}

/**
 * Starts an RPC server that receives requests for the Greeter service at the
 * sample server port
 */
function main() {
    var server = new grpc.Server();
    server.addService(weather_proto.Weather.service, { getWeather: getWeather });
    server.bind('0.0.0.0:50051', grpc.ServerCredentials.createInsecure());
    console.log("Server Started on 0.0.0.0:50051");
    server.start();
}
```

```
main();
```

client.js

```
var PROTO_PATH = 'weather.proto';

var grpc = require('grpc');
var protoLoader = require('@grpc/proto-loader');
var packageDefinition = protoLoader.loadSync(
    PROTO_PATH,
    {
        keepCase: true,
        longs: String,
        enums: String,
        defaults: true,
        oneofs: true
    });
var weather_proto = grpc.loadPackageDefinition(packageDefinition).weather;

var client = new weather_proto.Weather('localhost:50051',
    grpc.credentials.createInsecure());

function runGetWeather(callback) {
    console.log('STREAMING WEATHER DATA FOR Bangalore');

    var city = {
        city_name: "bangalore",
    }

    var call = client.getWeather(city);

    call.on('data', function (weather_data) {
        console.log(weather_data);
    });

    call.on('end', callback);
}

function main() {

    runGetWeather(function () {
        console.log("END OF STREAM");
    });
}

main();
```

weather.proto

```
syntax = "proto3";

package weather;
```

```
service Weather {
  rpc GetWeather (CityRequest) returns (stream WeatherReply) {}
}

// The request message containing the city name
message CityRequest {
  string city_name = 1;
}

// The stream of reply containing the weather data
message WeatherReply {
  string timestamp = 1;
  string city = 2;
  string temp = 3;
  string weather_status = 4;
}
```

Running the Server

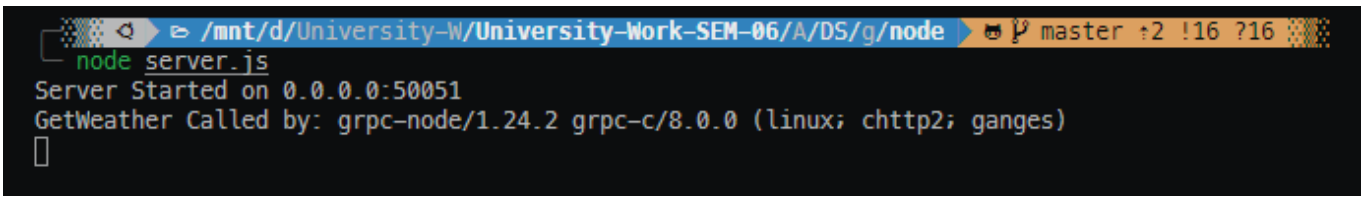

Figure 14 Node gRPC Server

Running the Client

```
/mnt/d/University-W/University-Work-SEM-06/A/DS/g/node  master ↑2 !16 ?16
└─ node client.js
STREAMING WEATHER DATA FOR Bangalore
{
  timestamp: '2020-04-27T23:02:31.125Z',
  city: 'bangalore',
  temp: '27.8',
  weather_status: 'cloudy'
}
{
  timestamp: '2020-04-27T23:02:32.125Z',
  city: 'bangalore',
  temp: '28.8',
  weather_status: 'cloudy'
}
{
  timestamp: '2020-04-27T23:02:33.125Z',
  city: 'bangalore',
  temp: '27.6',
  weather_status: 'cloudy'
}
{
  timestamp: '2020-04-27T23:02:34.125Z',
  city: 'bangalore',
  temp: '27.5',
  weather_status: 'cloudy'
}
{
  timestamp: '2020-04-27T23:02:35.125Z',
  city: 'bangalore',
  temp: '27.9',
  weather_status: 'cloudy'
}
{
  timestamp: '2020-04-27T23:02:36.125Z',
  city: 'bangalore',
  temp: '28',
  weather_status: 'cloudy'
}
{
  timestamp: '2020-04-27T23:02:37.125Z',
  city: 'bangalore',
  temp: '28.2',
  weather_status: 'cloudy'
}
{
  timestamp: '2020-04-27T23:02:38.125Z',
  city: 'bangalore',
  temp: '28',
  weather_status: 'cloudy'
}
{
  timestamp: '2020-04-27T23:02:39.125Z',
  city: 'bangalore',
  temp: '27.9',
  weather_status: 'cloudy'
}
END OF STREAM
/mnt/d/University-W/University-Work-SEM-06/A/DS/g/node  master ↑2 !16 ?16
```

Figure 15 Node gRPC client

The Client here receives weather updates from the server every minute, with the temperature and current weather status.

2.1.2 Example 2

server.js

```
var PROTO_PATH = 'marks.proto';

var fs = require('fs');
var _ = require('lodash');
var grpc = require('grpc');
var protoLoader = require('@grpc/proto-loader');
var packageDefinition = protoLoader.loadSync(
    PROTO_PATH,
    {
        keepCase: true,
        longs: String,
        enums: String,
        defaults: true,
        oneofs: true
    });

var marks_proto = grpc.loadPackageDefinition(packageDefinition).marks;

let rawdata = fs.readFileSync('marks.json');
let marks_data = JSON.parse(rawdata);

/**
 * Implements the GetMarks Service
 */
function getMarks(call) {
    var start_time = process.hrtime();
    console.log('GetMarks Called by: ' + call.metadata._internal_repr['user-agent']);
    call.on('data', function (student) {
        var data = {
            student_name: student.name,
            marks: marks_data[student.name]
        }
        call.write(data);
    });
    call.on('end', function () {
        console.log({ elapsed_time: process.hrtime(start_time) });
        call.end();
    });
}

/**
 * Starts an RPC server that receives requests for the Greeter service at the
 * sample server port
 */
function main() {
    var server = new grpc.Server();
    server.addService(marks_proto.MarksPortal.service, { getMarks: getMarks });
    server.bind('0.0.0.0:50051', grpc.ServerCredentials.createInsecure());
```

```
    console.log("Server Started on 0.0.0.0:50051");
    server.start();
}

main();
```

client.js

```
var PROTO_PATH = 'marks.proto';

var _ = require('lodash');
var grpc = require('grpc');
var protoLoader = require('@grpc/proto-loader');
var packageDefinition = protoLoader.loadSync(
    PROTO_PATH,
    {
        keepCase: true,
        longs: String,
        enums: String,
        defaults: true,
        oneofs: true
    });
var marks_proto = grpc.loadPackageDefinition(packageDefinition).marks;

var student = ['satyajit', 'shubham', 'samhitha', 'shikhar', 'shobhan'];

var client = new marks_proto.MarksPortal('localhost:50051',
    grpc.credentials.createInsecure());

function runGetMarks(callback) {
    console.log('START STREAM');
    var call = client.getMarks();

    call.on('data', function (marks_data) {
        console.log(marks_data);
    });

    call.on('end', callback);

    _.each(student, function (std) {
        var my_student = { name: std };
        call.write(my_student);
    });

    call.end();
}

function main() {

    runGetMarks(function () {
        console.log("END OF STREAM");
    });
```

```
}

main();
```

marks.proto

```
syntax = "proto3";

package marks;

service MarksPortal {
  rpc GetMarks (stream StudentRequest) returns (stream MarksReply) {}
}

// The request message containing the student  name
message StudentRequest {
  string name = 1;
}

// The reply containing the marks
message MarksReply {
  string student_name = 1;
  Marks marks = 2;
}

message Marks {
  string DS = 2;
  string DBMS = 3;
  string Compilers = 4;
  string Graphics = 5;
}
```

Start Server

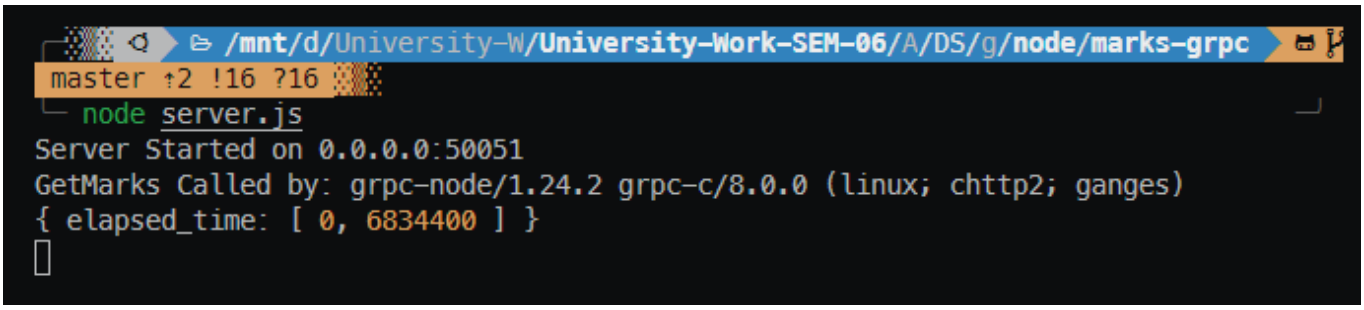

Figure 16 Node Marks Server

Start Client

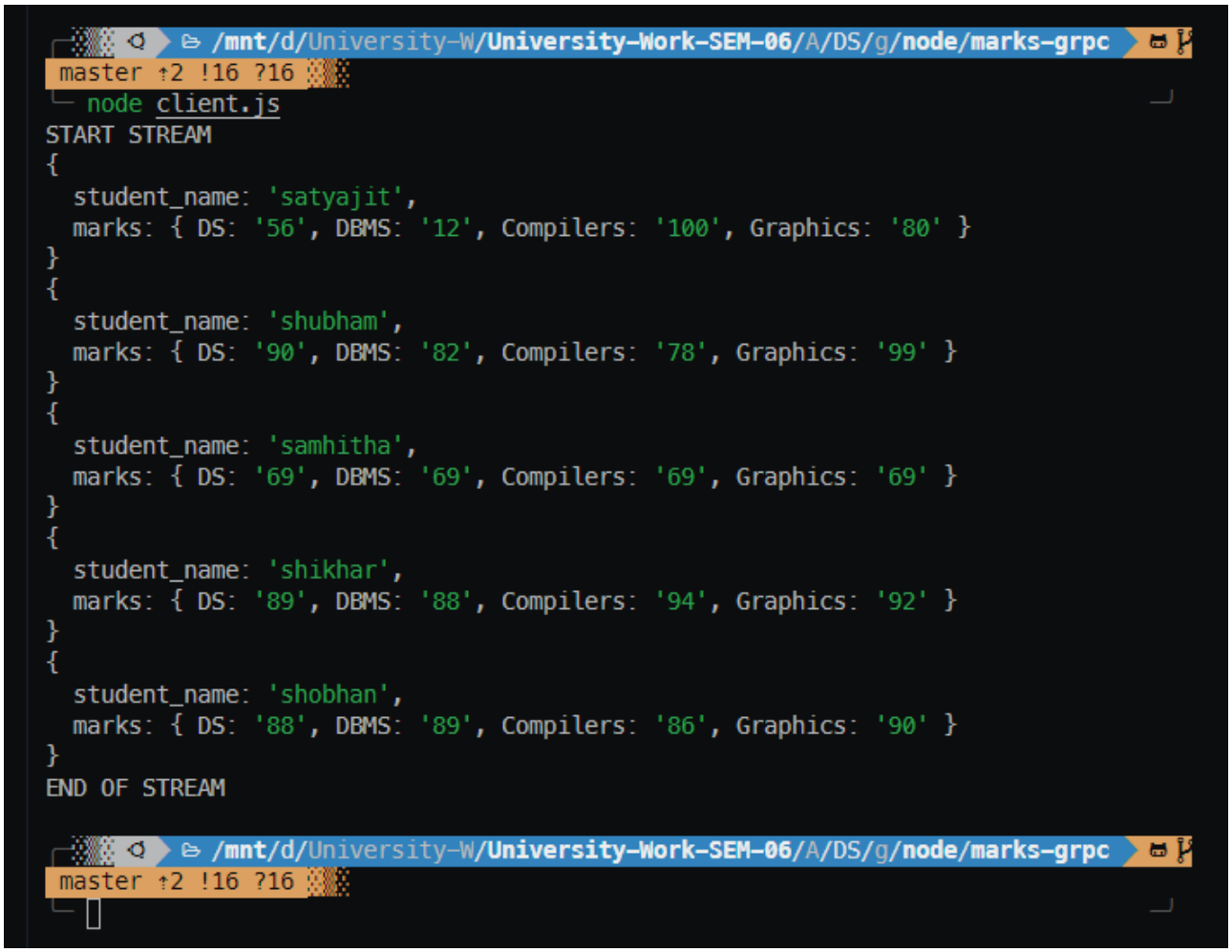

Figure 17 Node Marks Client

Unlike normal procedure calls, many things can go wrong with RPC. Normally, a client will send a request, the server will execute the request and then return a response to the client. What are appropriate semantics for server or network failures? Possibilities:

- Just hang forever waiting for the reply that will never come. This models regular procedure call. If a normal procedure goes into an infinite loop, the caller never finds out. Of course, few users will like such semantics.
- Time out and raise an exception or report failure to the client. Of course, finding an appropriate timer value is difficult. If the remote procedure takes a long time to execute, a timer might time-out too quickly.
- Time out and retransmit the request.

2.2 RMI, the need for it and two sample programs

Remote Method Invocation (RMI)

- RPC + Object Orientation
- Allows objects living in one process to invoke methods of an object living in another process

RMI Programming

- Proxy
  - Behaves like remote object to clients (invoker)
  - Marshals arguments, forwards message to remote object, unmarshals results, returns results to client
- Skeleton
  - Server side stub;
  - Unmarshals arguments, invokes method, marshals results and sends to sending proxy's method
- Dispatcher
  - Receives the request message from communication module, passes on the message to the appropriate method in the skeleton

The goals for supporting distributed objects in the Java programming language are:

- Support seamless remote invocation on objects in different virtual machines
- Support callbacks from servers to applets
- Integrate the distributed object model into the Java programming language in a natural way while retaining most of the Java programming language's object semantics
- Make differences between the distributed object model and local Java platform's object model apparent
- Make writing reliable distributed applications as simple as possible
- Preserve the type-safety provided by the Java platform's runtime environment
- Support various reference semantics for remote objects; for example live (nonpersistent) references, persistent references, and lazy activation

- Maintain the safe environment of the Java platform provided by security managers and class loaders Underlying all these goals is a general requirement that the RMI model be both simple (easy to use) and natural (fits well in the language). **(Java Docs)**

Advantages of RMI:

- Simple and clean to implement that leads to more robust, maintainable and flexible applications
- Distributed systems creations are allowed while decoupling the client and server objects simultaneously
- It is possible to create zero-install client for the users.
- No client installation is needed except java capable browsers
- At the time of changing the database, only the server objects are to be recompiled but not the server interface and the client remain the same.

Disadvantages of RMI:

- Less efficient than Socket objects.
- Assuming the default threading will allow ignoring the coding, being the servers are thread- safe and robust.
- Cannot use the code out of the scope of java.
- Security issues need to be monitored more closely.

2.2.1 Example 1

Calculator.java

```
import java.rmi.*;

public interface Calculator extends Remote {
    public String add(double x, double y) throws RemoteException;

    public String sub(double x, double y) throws RemoteException;

    public String mult(double x, double y) throws RemoteException;

    public String div(double x, double y) throws RemoteException;
}
```

CalculatorRemote.java

```
import java.rmi.*;
import java.rmi.server.*;
```

```java
public class CalculatorRemote extends UnicastRemoteObject implements Calculator {
    CalculatorRemote() throws RemoteException {
        super();
    }

    public String add(double x, double y) {
        String result;
        result = "Sum : " + (x + y);
        return result;
    }

    public String sub(double x, double y) {
        String result;
        result = "Difference(first value - second value) : " + (x - y);
        return result;
    }

    public String mult(double x, double y) {
        String result;
        result = "Product : " + (x * y);
        return result;
    }

    public String div(double x, double y) {
        String result;
        if (y == 0)
            result = "Division: Math Error! You cannot divide by 0.";
        else
            result = "Quotient : " + (x / y) + "\nRemainder : " + (x % y);
        return result;
    }

}
```

MyClient.java

```java
import java.rmi.*;
import java.util.Scanner; //for taking input from the user

public class MyClient {
    public static void main(String args[]) {

        try {
            Calculator stub = (Calculator) Naming.lookup("rmi://localhost:5001/calculator");

            System.out.println("\nSimple Calculator!");
            System.out.println("==========================");

            Scanner reader = new Scanner(System.in);
            char a;

            do {
                System.out.print("\n\nEnter two numbers: ");
```

```
                // nextDouble() reads the next double from the keyboard
                double a1 = reader.nextDouble();
                double a2 = reader.nextDouble();

                System.out.print("Enter an operator (+, -, *, /): ");
                char op = reader.next().charAt(0);
                switch (op) {
                    case '+':
                        System.out.println(stub.add(a1, a2));
                        break;
                    case '-':
                        System.out.println(stub.sub(a1, a2));
                        break;
                    case '*':
                        System.out.println(stub.mult(a1, a2));
                        break;
                    case '/':
                        System.out.println(stub.div(a1, a2));
                        break;

                    default:
                        System.out.printf("Error: Invalid operator!");
                        return;
                }

                System.out.print("Enter Y/y to continue and anything else to exit: ");
                a = reader.next().charAt(0);
            } while (a == 'Y' || a == 'y');

        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

MyServer.java

```
import java.rmi.*;
import java.rmi.registry.*;

public class MyServer {

    public static void main(String args[]) {

        try {
            Calculator stub = new CalculatorRemote();
            Naming.rebind("rmi://localhost:5001/calculator", stub);
        } catch (Exception e) {
            System.out.println(e);
        }

    } // main
} // MyServer
```

Execution

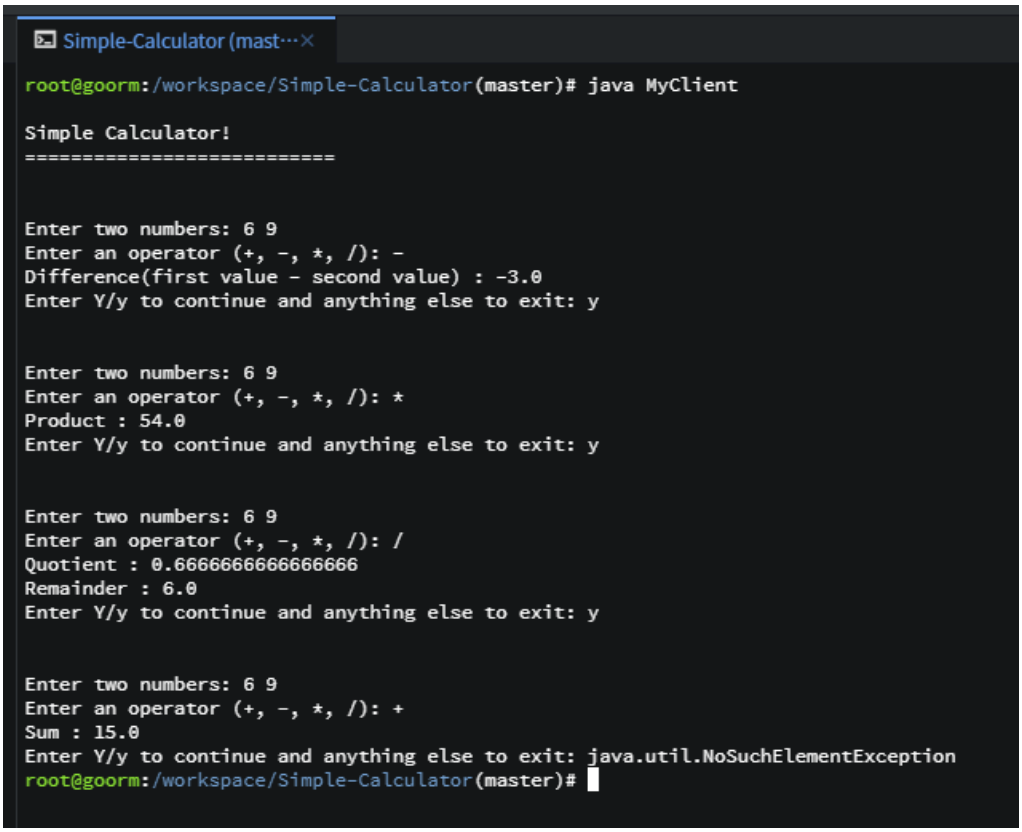

Figure 18 Simple Calculator RMI

2.2.2 Example 2

RMIInterface.java

```
package com.example.rmiinterface;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RMIInterface extends Remote {
    public String helloTo(String name) throws RemoteException;
}
```

ClientOperation.java

```
package com.mkyong.rmiclient;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

import javax.swing.JOptionPane;
```

```
import com.example.rmiinterface.RMIInterface;

public class ClientOperation {
    private static RMIInterface look_up;

    public static void main(String[] args) throws MalformedURLException, RemoteException, NotBoundExce
ption {

        look_up = (RMIInterface) Naming.lookup("//localhost/MyServer");
        String txt = JOptionPane.showInputDialog("What is your name?");

        String response = look_up.helloTo(txt);
        JOptionPane.showMessageDialog(null, response);
    }
}
```

ServerOperation.java

```
package com.example.rmiserver;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

import com.mkyong.rmiinterface.RMIInterface;

public class ServerOperation extends UnicastRemoteObject implements RMIInterface{
    private static final long serialVersionUID = 1L;

    protected ServerOperation() throws RemoteException {
        super();
    }

    @Override
    public String helloTo(String name) throws RemoteException{
        System.err.println(name + " is trying to contact!");
        return "Server says hello to " + name;
    }

    public static void main(String[] args){
        try {
            Naming.rebind("//localhost/MyServer", new ServerOperation());
            System.err.println("Server ready");

        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
          e.printStackTrace();
        }
    }
}
```

Execution

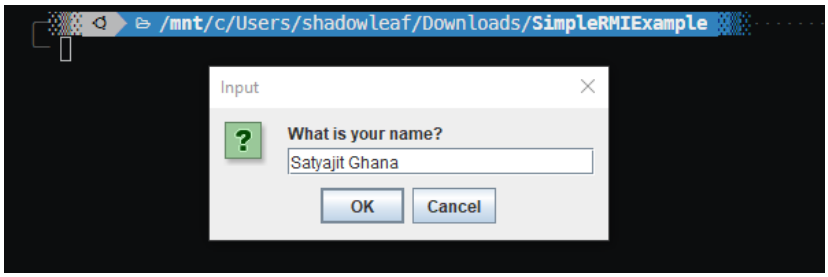

Figure 19 RMI Client

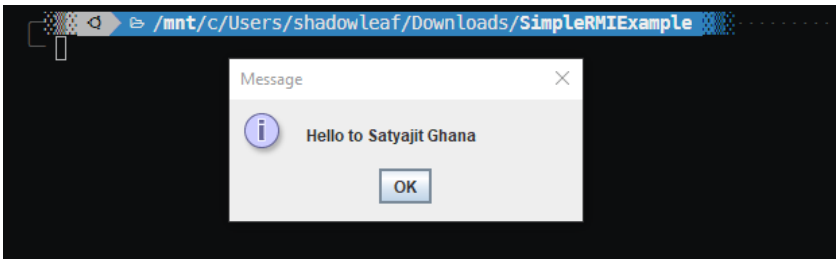

Figure 20 RMI Server

2.3 Similarities and Differences between RPC and RMI

RMI:

- The remote objects are accessed by the references.
- Implements object to object implementation among different java objects to implement distributed communication model.
- RMI passes the objects as parameters to remote methods.
- RMI invokes the remote methods from the objects.

RPC:

- The process is through methods / functions.
- Proxy server is involved in processing the procedure calls.
- Calls a procedure remotely like invoking the methods.
- The remoteness is not exactly transparent to the client.

Difference between RPC and RMI.

- RPC can be used to invoke functions through a proxy functions.
- RMI can used to invoke methods of an object.
- RPC supports procedural programming paradigms thus is C based, while RMI supports object-oriented programming paradigms and is java based.
- RPC protocol generates more overheads than RMI.
- The parameters passed in RPC must be "in-out" which means that the value passed to the procedure and the output value must have the same datatypes. In contrast, there is no compulsion of passing "in-out" parameters in RMI.
- In RPC, references could not be probable because the two processes have the distinct address space, but it is possible in case of RMI.
- The parameters passed to remote procedures in RPC are the ordinary data structures. On the contrary, RMI transits objects as a parameter to the remote method.

The similarities in RPC and RMI

- Both support programming with interfaces
- Both typically constructed on top of request-reply protocols and can offer a range of call semantics such as at-least-once and at-most-once
- Both offer a similar level of transparency –that is, local and remote calls employ the same syntax
- Both RPC and RMI follow the call/return style, where communication is initiated by the client, and the server reactively responds.
- Both RPC and RMI provide communication via local stubs, which support an interface for method calling.
- In RPC and RMI calling and returning is implemented via message passing.
- Both use separate mechanisms for dynamic binding (e.g., object registry, JINI).

Bibliography

1. Refer to this for complete code
   `https://github.com/satyajitghana/ProjektZirconium`
2. https://platformlab.stanford.edu/Seminar%20Talks/gRPC.pdf
3. https://docs.mongodb.com/manual/faq/concurrency/
4. https://cs.gmu.edu/~setia/cs571-F02/slides/lec7a.pdf
5. https://web.cs.wpi.edu/~cs4514/b98/week8-rpc/week8-rpc.html
6. https://mkyong.com/java/java-rmi-hello-world-example/
7. http://docs.oracle.com/javase/1.4.2/docs/guide/rmi/spec/rmi-intro3.html
8. https://www.careerride.com/RMI-vs-RPC.aspx
9. https://techdifferences.com/difference-between-rpc-and-rmi.html

Appendix A

Server Code

zirconium_server.cc

```
// to use c++ style logging
// specifically define this before including loguru.hpp
#define LOGURU_WITH_STREAMS 1

#include <grpcpp/grpcpp.h>

#include <bsoncxx/builder/stream/document.hpp>
#include <bsoncxx/json.hpp>
#include <iostream>
#include <map>
#include <memory>
#include <mongocxx/client.hpp>
#include <mongocxx/exception/operation_exception.hpp>
#include <mongocxx/instance.hpp>
#include <mongocxx/pool.hpp>
#include <mongocxx/stdx.hpp>
#include <mongocxx/uri.hpp>
#include <stdexcept>
#include <string>
#include <thread>

#include "protos/zirconiumbank.grpc.pb.h"
#include "server/transaction_mgr.hpp"
#include "utils/cxxopts.hpp"
#include "utils/loguru.hpp"
#include "utils/zirconium_utils.hpp"

using bsoncxx::builder::basic::kvp;
using bsoncxx::builder::basic::make_array;
using bsoncxx::builder::basic::make_document;
using bsoncxx::builder::stream::close_array;
using bsoncxx::builder::stream::close_document;
using bsoncxx::builder::stream::document;
using bsoncxx::builder::stream::finalize;
using bsoncxx::builder::stream::open_array;
using bsoncxx::builder::stream::open_document;

// gRPC namespaces
using grpc::Server;
using grpc::ServerBuilder;
using grpc::ServerContext;
using grpc::Status;
using grpc::StatusCode;

// zirconium grpc-proto namespaces
using zirconium::bank::AuthUser;
using zirconium::bank::Balance;
using zirconium::bank::RequestAmount;
```

```
using zirconium::bank::TransferReq;
using zirconium::bank::WebService;

class WebServiceImpl final : public WebService::Service {
    /**
     * BalanceEnquiry: Gets the balance of a AuthUser
    */
    Status BalanceEnquiry(ServerContext* context, const AuthUser* auth_user, Balance* balance) override {
        // get a client from the pool
        auto client = conn_pool_->acquire();

        // log the auth data from the request
        logAuthData_(auth_user->username(), auth_user->pin());

        // check if the user is authenticated and also fetch the document
        auto [is_authenticated, user] = getAuthData_(auth_user->username(), auth_user->pin(), *(client));

        if (not is_authenticated) {
            LOG_S(WARNING) << "UNAUTHENTICATED User: " << auth_user->username();
            return Status(StatusCode::UNAUTHENTICATED, "username or pin wrong!");
        }

        // the user is authenticated

        balance->set_value(user->view()["balance"].get_int32());

        return Status::OK;
    }

    /**
     * UpdateAmount : updates the balance of the user with the given amount
     * Tested Works ✅
    */
    Status UpdateAmount(ServerContext* context, const RequestAmount* update_req, Balance* balance) override
{
        LOG_S(INFO) << "UpdateAmount called by peer: " << context->peer();

        // get a client from the pool
        auto client = conn_pool_->acquire();

        // get the "zirconium" db
        auto db = (*client)["zirconium"];

        const auto auth_user = update_req->user();
        logAuthData_(auth_user.username(), auth_user.pin());

        try {
            auto [is_authenticated, user] = getAuthData_(auth_user.username(), auth_user.pin(), (*client));

            if (not is_authenticated) {
                LOG_S(WARNING) << "UNAUTHENTICATED User: " << auth_user.username();
                return Status(StatusCode::UNAUTHENTICATED, "username or pin wrong!");
            }

            // the user is authenticated

            auto update_user_amount = [&](mongocxx::client_session& session) {
```

```
                // get the transaction options
                auto txn_opts = zirconium::transaction_mgr::get_transaction_opts();

                // initiate the transaction
                session.start_transaction(txn_opts);

                // container for the updated user
                bsoncxx::stdx::optional<bsoncxx::document::value> updated_user;

                // try the transaction
                try {
                    // read fuser value from db
                    auto curr_user = db["bank"].find_one(session, make_document(kvp("username", auth_user.u
sername())));
                    auto curr_user_balance = curr_user->view()["balance"].get_int32().value;

                    // guard to check if from_user has necessary balance
                    if (curr_user_balance + update_req->value() < 0) {
                        // insufficient balance
                        throw zirconium::exception("insufficient balance");
                    }

                    // increment/decrement the balance of user
                    updated_user = db["bank"].find_one_and_update(
                        session,
                        curr_user->view(),
                        make_document(
                            kvp("$inc", make_document(kvp("balance", update_req->value())))),
                        mongocxx::options::find_one_and_update().return_document(mongocxx::options::return_
document::k_after));

                    // set the new balance as responce
                    balance->set_value(updated_user->view()["balance"].get_int32());

                } catch (const mongocxx::operation_exception& oe) {
                    session.abort_transaction();
                    throw oe;
                }

                // run this if the transaction was successful
                auto on_success = [&](void) {
                    LOG_S(INFO) << "UPDATED: " << bsoncxx::to_json(*updated_user) << "\n";
                    auto update_type = update_req->value() >= 0 ? "credited" : "debited";

                    LOG_S(INFO) << "UpdateAmount [ Account " << updated_user-
>view()["username"].get_utf8().value.to_string() << " " << update_type << " with ₹ " << std::abs(update_req
->value()) << " new balance: " << updated_user->view()["balance"].get_int32() << " ]";
                };

                // commit the transaction and retry if failure
                zirconium::transaction_mgr::commit_with_retry(session, on_success);
            };

            // create a session and start a transaction
            auto session = (*client).start_session();

            try {
```

```
                zirconium::transaction_mgr::run_transaction_with_retry(update_user_amount, session);
            } catch (const mongocxx::operation_exception& oe) {
                LOG_S(ERROR) << "Error during commit: " << oe.what() << ", for session: " << bsoncxx::to_js
on(session.id());
            }

        } catch (zirconium::exception& ex) {
            LOG_S(ERROR) << "Precondition failed: " << ex.what();
            return Status(StatusCode::FAILED_PRECONDITION, ex.what());
        } catch (std::exception& ex) {
            LOG_S(ERROR) << "Internal Server Error: " << ex.what();
            return Status(StatusCode::INTERNAL, "Internal Server Error, Cannot Update Amount");
        }

        return Status::OK;
    }

    /**
     * Transfer Amount: Transfer amount from current user to to_user
     * Tested Works ☑
    */
    Status TransferAmount(ServerContext* context, const TransferReq* transfer_req, Balance* balance) overri
de {
        LOG_S(INFO) << "TransferAmount called by peer: " << context->peer();

        // get a client from the pool
        auto client = conn_pool_->acquire();

        // get the "zirconium" db
        auto db = (*client)["zirconium"];

        // curr_user
        const auto auth_user = transfer_req->from_user();
        const auto to_user = transfer_req->to_user();

        // guard to check if the to_user is same as curr_user
        if (to_user == auth_user.username()) {
            return Status(StatusCode::FAILED_PRECONDITION, "to_user is same as curr_user");
        }

        logAuthData_(auth_user.username(), auth_user.pin());

        try {
            auto [is_authenticated, user] = getAuthData_(auth_user.username(), auth_user.pin(), (*client));

            // check if the user is authenticated
            if (not is_authenticated) {
                LOG_S(WARNING) << "UNAUTHENTICATED User: " << auth_user.username();
                return Status(StatusCode::UNAUTHENTICATED, "username or pin wrong!");
            }

            // the user is authenticated

            // transaction function
            auto transfer_user_amount = [&](mongocxx::client_session& session) {
                // get the transaction options
                auto txn_opts = zirconium::transaction_mgr::get_transaction_opts();
```

```
                // initiate the transaction
                session.start_transaction(txn_opts);

                // container for the updated user
                bsoncxx::stdx::optional<bsoncxx::document::value> updated_from_user;

                // try the transaction
                try {
                    // read to_user value from db
                    auto curr_to_user = db["bank"].find_one(session, make_document(kvp("username", to_user)
));

                    // this is unnecessary, can be ommited
                    if (not curr_to_user) {
                        // to_user does not exist, abort transaction
                        throw zirconium::exception("to_user does not exist");
                    }

                    // read from_user value from db
                    auto curr_from_user = db["bank"].find_one(session, make_document(kvp("username", auth_u
ser.username())));
                    auto curr_from_user_balance = curr_from_user->view()["balance"].get_int32().value;

                    // guard to check if from_user has necessary balance
                    if (curr_from_user_balance - transfer_req->amount() < 0) {
                        // insufficient balance
                        throw zirconium::exception("insufficient balance");
                    }

                    // decrement the balance of from_user
                    updated_from_user = db["bank"].find_one_and_update(
                        session,
                        curr_from_user->view(),
                        make_document(
                            kvp("$inc", make_document(
                                        kvp("balance", -transfer_req->amount())))),
                        mongocxx::options::find_one_and_update().return_document(mongocxx::options::return_
document::k_after));

                    // increment the balance of to_user
                    db["bank"].find_one_and_update(
                        session,
                        curr_to_user->view(),
                        make_document(
                            kvp("$inc", make_document(
                                        kvp("balance", transfer_req->amount())))),
                        mongocxx::options::find_one_and_update());

                    // set the new balance as responce
                    balance->set_value(updated_from_user->view()["balance"].get_int32());

                } catch (const mongocxx::operation_exception& oe) {
                    session.abort_transaction();
                    throw oe;
                } catch (const zirconium::exception& ex) {
                    session.abort_transaction();
```

```
                    throw ex;
                }

                // run this if the transaction was successful
                auto on_success = [&](void) {
                    LOG_S(INFO) << "UPDATED: " << bsoncxx::to_json(*updated_from_user) << "\n";
                    auto update_type = transfer_req->amount() >= 0 ? "credited" : "debited";

                    LOG_S(INFO) << "UpdateAmount [ Account " << updated_from_user-
>view()["username"].get_utf8().value.to_string() << " " << update_type << " with ₹ " << std::abs(transfer_r
eq->amount()) << " new balance: " << updated_from_user->view()["balance"].get_int32() << " ]";
                };

                // commit the transaction and retry if failure
                zirconium::transaction_mgr::commit_with_retry(session, on_success);
            };

            // create a session and start a transaction
            auto session = client->start_session();

            try {
                zirconium::transaction_mgr::run_transaction_with_retry(transfer_user_amount, session);
            } catch (const mongocxx::operation_exception& oe) {
                // some expection occured during commit
                LOG_S(ERROR) << "Error during commit: " << oe.what() << ", for session: " << bsoncxx::to_js
on(session.id());
            }
        } catch (zirconium::exception& ex) {
            LOG_S(ERROR) << "Precondition failed: " << ex.what();
            return Status(StatusCode::FAILED_PRECONDITION, ex.what());

        } catch (const std::exception& ex) {
            LOG_S(ERROR) << "Internal Server Error: " << ex.what();
            return Status(StatusCode::INTERNAL, "Internal Server Error, Cannot Transfer Amount");

        } catch (...) {
            // something notorious has happened if you reach here
            // stop the server
            LOG_S(FATAL) << "shutting down";
        }

        return Status::OK;
    }

   public:
    void initiaize(const std::string& mongo_uri) {
        conn_pool_ = std::make_unique<mongocxx::pool>(mongocxx::uri{mongo_uri});
    };

   private:
    std::unique_ptr<mongocxx::pool> conn_pool_;

    void logAuthData_(std::string user, std::string pin) {
        LOG_S(INFO) << "Received AuthUser: [ username: " << user << ", pin: " << pin << " ]";
    }
```

```
    std::tuple<bool, bsoncxx::stdx::optional<bsoncxx::document::value>> getAuthData_(std::string username,
std::string pin, mongocxx::client& client) {
        auto db = client["zirconium"];

        bool is_authenticated = false;
        bsoncxx::stdx::optional<bsoncxx::document::value> user = db["bank"].find_one(document{} << "usernam
e" << username << finalize);

        if (user) {
            LOG_S(INFO) << "Found User from DB: " << bsoncxx::to_json(*user);
            if (pin == user->view()["pin"].get_utf8().value.to_string()) {
                is_authenticated = true;
            }
        } else {
            LOG_S(ERROR) << username << " not found !";
        }

        return std::make_tuple(is_authenticated, user);
    }
};

void RunServer(const std::string& server_addr, const std::string& mongo_uri) {
    WebServiceImpl service;

    // initialize the service with mongo db name "zirconium"
    service.initiaize(mongo_uri);

    ServerBuilder builder;

    // Listen on the given address without any authentication mechanism.
    builder.AddListeningPort(server_addr, grpc::InsecureServerCredentials());

    // Register "service" as the instance through which we'll communicate with
    // clients. In this case it corresponds to an *synchronous* service.
    builder.RegisterService(&service);

    // Finally assemble the server.
    std::unique_ptr<Server> server(builder.BuildAndStart());
    LOG_S(INFO) << "Server listening on " << server_addr;

    // Wait for the server to shutdown. Note that some other thread must be
    // responsible for shutting down the server for this call to ever return.
    server->Wait();
}

int main(int argc, char* argv[]) {
    // turn off the printing thread info in the output
    // loguru::g_preamble_thread = false;

    // turn off the uptime in the output
    loguru::g_preamble_uptime = false;

    cxxopts::Options options("zirconium_server", "Server for Zirconium Bank");

    options
        .positional_help("[optional args]")
        .show_positional_help();
```

```
    options
        .add_options()("config", "config file", cxxopts::value<std::string>())("h,help", "print help");

    auto result = options.parse(argc, argv);

    if (result.count("help")) {
        std::cout << options.help() << "\n";
        exit(0);
    }

    // check for the config file
    if (!result.count("config")) {
        LOG_S(ERROR) << "config file not provided";
    }

    auto config = zirconium::parse_config(result["config"].as<std::string>());

    // check if the config was provided correctly
    if ((config.find("server_addr") == config.end()) or (config.find("mongo_addr") == config.end())) {
        throw std::runtime_error("config file incorrect, server_addr or mongo_addr not found");
    }

    // instantiate the mongo connection
    mongocxx::instance instance{};  // This should be done only once.
    std::string mongo_addr(config["mongo_addr"]);

    // create a pool
    // mongocxx::pool pool{uri};

    std::string server_addr(config["server_addr"]);

    // run the server with the pool
    RunServer(server_addr, mongo_addr);

    return 0;
}
```

transaction_mgr.hpp

```
#pragma once

// to use c++ style logging
// specifically define this before including loguru.hpp
#define LOGURU_WITH_STREAMS 1

#include <bsoncxx/json.hpp>
#include <iostream>
#include <mongocxx/client.hpp>
#include <mongocxx/exception/operation_exception.hpp>

#include "utils/loguru.hpp"

namespace zirconium {

namespace transaction_mgr {

using namespace mongocxx;
```

```
using on_success_func = std::function<void(void)>;

inline auto commit_with_retry(client_session& session, on_success_func succ_fun) {
    while (true) {
        try {
            session.commit_transaction();  // Uses write concern set at transaction start.
            LOG_S(INFO) << "Transaction commited : " << bsoncxx::to_json(session.id());
            succ_fun();
            break;
        } catch (const operation_exception& oe) {
            // Can retry commit
            if (oe.has_error_label("UnknownTransactionCommitResult")) {
                LOG_S(WARNING) << "UnknownTransactionCommitResult, retrying commit operation for session: "
 << bsoncxx::to_json(session.id());
                continue;
            } else {
                throw oe;
            }
        }
    }
};

using transaction_func = std::function<void(client_session& session)>;

inline auto run_transaction_with_retry(transaction_func txn_func, client_session& session) {
    while (true) {
        try {
            txn_func(session);  // performs transaction.
            break;
        } catch (const operation_exception& oe) {
            LOG_S(ERROR) << "Transaction aborted. Caught exception during transaction: " << oe.what() << ",
 for session: " << bsoncxx::to_json(session.id());
            // If transient error, retry the whole transaction.
            if (oe.has_error_label("TransientTransactionError")) {
                LOG_S(WARNING) << "TransientTransactionError, retrying transaction for session: " << bsoncx
x::to_json(session.id());
                continue;
            } else {
                throw oe;
            }
        }
    }
};

inline auto get_transaction_opts() {
    mongocxx::options::transaction txn_opts;

    // read and write concerns and read preference
    mongocxx::write_concern wc_majority{};
    wc_majority.acknowledge_level(mongocxx::write_concern::level::k_majority);

    mongocxx::read_concern rc_local{};
    rc_local.acknowledge_level(mongocxx::read_concern::level::k_local);

    mongocxx::read_preference rp_primary{};
    rp_primary.mode(mongocxx::read_preference::read_mode::k_primary);
```

```
    txn_opts.write_concern(wc_majority);
    txn_opts.read_concern(rc_local);
    txn_opts.read_preference(rp_primary);

    return txn_opts;
};

}  // namespace transaction_mgr

class exception : public std::exception {
   public:
    exception(const std::string& message) : msg_(message) {}

    virtual const char* what() const throw() { return msg_.c_str(); }

   private:
    std::string msg_;
};

}  // namespace zirconium
```

zirconiumbank.proto

```
syntax = "proto3";

// ZirconiumBank (ZrB)

/* The User can requestt for the following from the server
 * 1. Update Amount
 * 2. Transfer Amount
 * 3. Balance Enquiry
 * */

/* The Request should always contain
 * 1. Username
 * 2. 6-digit PIN
 * */

 // The C++ namesapce will be zirconium::bank
package zirconium.bank;

// Interface exported by the server
service WebService {
    rpc UpdateAmount(RequestAmount) returns (Balance) {}
    rpc BalanceEnquiry(AuthUser) returns (Balance) {}
    rpc TransferAmount(TransferReq) returns (Balance) {}
}

message TransferReq {
    AuthUser from_user = 1;
    string to_user = 2;
    int32 amount = 3;
}

message AuthUser {
    string username = 1;
    string pin = 2;
```

```
}

message RequestAmount {
    AuthUser user = 1;
    int32 value = 2;
}

message Balance {
    int32 value = 1;
}
```

Appendix B

Client Code

zirconium_client.cc

```
// to use c++ style logging
// specifically define this before including loguru.hpp
#define LOGURU_WITH_STREAMS 1

#include <grpcpp/grpcpp.h>

#include <iostream>
#include <memory>
#include <sstream>
#include <string>

#include "protos/zirconiumbank.grpc.pb.h"
#include "utils/cxxopts.hpp"
#include "utils/loguru.hpp"
#include "utils/zirconium_utils.hpp"

// gRPC namespaces
using grpc::Channel;
using grpc::ClientContext;
using grpc::Status;

// zirconium grpc-proto namespaces
using zirconium::bank::AuthUser;
using zirconium::bank::Balance;
using zirconium::bank::RequestAmount;
using zirconium::bank::TransferReq;
using zirconium::bank::WebService;

class ZirconiumClient {
   public:
    ZirconiumClient(std::shared_ptr<Channel> channel) : stub_(WebService::NewStub(channel)) {}

    std::string BalanceEnquiry(const std::string& user, const std::string& pin) {
        // Data that we will send to the server
        AuthUser authuser_req;
        authuser_req.set_username(user);
        authuser_req.set_pin(pin);

        // The Response we will get
        Balance balance_res;

        // Context for the client. It could be used to convey extra information to
        // the server and/or tweak certain RPC behaviors.
        ClientContext context;

        // The actual RPC
        Status status = stub_->BalanceEnquiry(&context, authuser_req, &balance_res);
```

```
        return getResponse_(status, user, balance_res);
    }

    std::string UpdateAmount(const std::string& user, const std::string& pin, int update_amt) {
        // the despoit amount request
        RequestAmount update_amt_req;

        LOG_IF_S(WARNING, update_amt == 0) << "update amount = 0";

        LOG_IF_S(WARNING, update_amt < 0) << "Debit Req of Amount: " << update_amt << " by user: " << user;
        LOG_IF_S(WARNING, update_amt > 0) << "Credit Req of Amount: " << update_amt << " by user: " << user
;


        // create the request payload
        update_amt_req.mutable_user()->set_username(user);
        update_amt_req.mutable_user()->set_pin(pin);
        update_amt_req.set_value(update_amt);

        // The Response we will get
        Balance balance_res;

        ClientContext context;

        // The DepositAmount RPC
        Status status = stub_->UpdateAmount(&context, update_amt_req, &balance_res);

        return getResponse_(status, user, balance_res);
    }

    std::string TransferAmount(const std::string& user, const std::string& pin, const std::string& to_user,
 int trans_amt) {
        // the transfer amount request
        TransferReq trans_amt_req;

        if (trans_amt < 0) {
            LOG_S(ERROR) << "Transfer Amount cannot be negative, wtf you trying to do?";
            exit(0);
        }

        LOG_IF_S(WARNING, trans_amt == 0) << "transfer amount = 0";
        LOG_IF_S(WARNING, trans_amt > 0) << "Transfer Req of Amount: " << trans_amt << " by user: " << user
;


        trans_amt_req.mutable_from_user()->set_username(user);
        trans_amt_req.mutable_from_user()->set_pin(pin);
        trans_amt_req.set_to_user(to_user);
        trans_amt_req.set_amount(trans_amt);

        // The Response we get
        Balance balance_res;

        ClientContext context;

        // The Transfer Amount RPC
        Status status = stub_->TransferAmount(&context, trans_amt_req, &balance_res);
```

```
        return getResponse_(status, user, balance_res);
    }

   private:
    std::unique_ptr<WebService::Stub> stub_;

    ClientContext ctx;

    std::string getResponse_(Status status, std::string user, Balance balance_res) {
        if (status.ok()) {
            std::stringstream response;
            response << "STATUS OK [ user: " << user << ", balance: " << std::to_string(balance_res.value()
) << " ]";
            LOG_S(INFO) << response.str();

            return response.str();
        } else {
            std::stringstream response;
            response << "RPC FAILED [ " << status.error_code() << ": " << status.error_message();

            LOG_S(ERROR) << response.str();

            return response.str();
        }
    }
};

int main(int argc, char* argv[]) {
    // turn off the printing thread info in the output
    // loguru::g_preamble_thread = false;

    // turn off the uptime in the output
    loguru::g_preamble_uptime = false;

    try {
        cxxopts::Options options("zirconium_client", "A client for Zirconium Bank");

        options
            .positional_help("[optional args]")
            .show_positional_help();

        options.allow_unrecognised_options()
            .add_options()("config", "config file", cxxopts::value<std::string>())("username", "username",
cxxopts::value<std::string>())("pin", "PIN", cxxopts::value<std::string>())("balance_enquiry", "Balance Enq
uiry", cxxopts::value<bool>()-
>implicit_value("false"))("d,debug", "enable debugging")("h,help", "print help");

        options.add_options("withdraw", {{"withdraw", "Withdraw a given amount", cxxopts::value<int>()-
>implicit_value("0")}});
        options.add_options("deposit", {{"deposit", "Deposit a given amount", cxxopts::value<int>()-
>implicit_value("0")}});
        options.add_options("money transfer", {{"to_user", "username of receiver party", cxxopts::value<std
::string>()},
                                               {"amt", "amount to be transferred", cxxopts::value<int>()-
>implicit_value("0")}});

        auto result = options.parse(argc, argv);
```

```
        if (result.count("help")) {
            std::cout << options.help() << "\n";
            exit(0);
        }

        // check for the config file
        if (!result.count("config")) {
            LOG_S(ERROR) << "config file not provided";
        }

        auto config = zirconium::parse_config(result["config"].as<std::string>());

        // check if the config was provided correctly
        if (config.find("server_addr") == config.end()) {
            throw std::runtime_error("config file incorrect, server_addr not found");
        }

        std::string server_addr(config["server_addr"]);

        // Instantiate the client. It requires a channel, out of which the actual RPCs
        // are created. This channel models a connection to an endpoint (in this case,
        // localhost at port 50051). We indicate that the channel isn't authenticated
        // (use of InsecureChannelCredentials()).
        ZirconiumClient zirconium_bank(grpc::CreateChannel(server_addr, grpc::InsecureChannelCredentials())
);

        // check if the username or the pin is not given as argument
        if (!(result.count("username") and result.count("pin"))) {
            LOG_S(ERROR) << "username or pin not given";
            exit(1);
        }

        // get the given values from the arguments
        std::string username = result["username"].as<std::string>();
        std::string pin = result["pin"].as<std::string>();

        // check if the strings are empty
        if (username.empty() or pin.empty()) {
            LOG_S(ERROR) << "username or pin empty"
                         << "\n";
            exit(1);
        }

        // the username and pin is provided properly

        // deposit action argument passed
        if (result.count("deposit")) {
            // the deposit amount
            int dep_amt = result["deposit"].as<int>();
            std::string reply = zirconium_bank.UpdateAmount(username, pin, dep_amt);
        }

        // withdraw action argument passed
        if (result.count("withdraw")) {
            // the withdraw amount
            int with_amt = result["withdraw"].as<int>();
```

```
            std::string reply = zirconium_bank.UpdateAmount(username, pin, -with_amt);
        }

        // money transfer argument passed
        if (result.count("to_user")) {
            int trans_amt = result["amt"].as<int>();
            std::string to_user = result["to_user"].as<std::string>();

            if ((to_user == username) or to_user.empty()) {
                LOG_S(ERROR) << "to_user cannot be emoty nor the same user as current user";
                exit(1);
            }

            std::string reply = zirconium_bank.TransferAmount(username, pin, to_user, trans_amt);
        }

        // balance enquiry argument passed
        if (result["balance_enquiry"].as<bool>()) {
            std::string reply = zirconium_bank.BalanceEnquiry(username, pin);
        }

    } catch (const cxxopts::OptionException& e) {
        LOG_S(ERROR) << "error parsing options: " << e.what();
        exit(1);
    }

    return 0;
}
```