# CS 6210: Advanced Operating Systems

**Project 4: Superstore!**

**Release Date** - Saturday, November 9, 2019

**Due date** - Sunday, December 8, 2019 11:59 PM

- **You may work in pairs on this project.**
- **You may share ideas with other pairs, but you may not share code.**
- **You also may not download code or use any other external libraries without consulting the instructor or TA first.**
- **Copying others' work is NEVER allowed for any reason. Please refer to the Georgia Tech honor code available at [http://www.honor.gatech.edu/](http://www.honor.gatech.edu/)**

## Working in Groups

---

- You are only allowed to work in pairs (2 students in a group).
- If you are working in pairs you should make one **submission** while other student uploads a text file as shown below.
    - Filename - project_3_<gt_handler>.txt
    - Content
        - Your name, GT ID
        - Group partner name, GT ID
- In the **report**, you are required to clarify the task division and each student will give the demo individually.

## Goal

---

In this project you will implement a distributed key-value store (GTStore). You should give special attention to system properties such as: (i) scalability, (ii) availability, (iii) resilience to temporary node failures. Implementation in C/C++.

## Introduction

---

### System Components

1. **GTStore**
    1. **Centralized manager:** One user-space process will handle all the control path decisions of the key-value store. The manager's tasks include, but are not limited to:
        1. Membership management
        2. Load balancing of clients across replica nodes
        3. Index management (e.g., you are allowed (but not required) to allow clients first contact centralized manager on initialization to figure out the data nodes that they should send their requests to)
    2. **Storage/Data nodes:** N user-space processes will act as the storage nodes in the key-value store. They will store in-memory and provide the key-value pairs, that they are responsible for. These processes represent the actual key-value store.
2. **Driver application:** M user-space processes (or threads) that interact with GTStore. In a typical interaction, the application (i) reads the latest version of the object, (ii) mutates it, and (iii) stores it back. For example, the application can mimic a shopping cart service. The stored object (value) can be an array of shopping cart items. E.g. {phone, phone_case, airpods} and the key can be the client-id. One application process should manipulate a single key-value pair at a time, e.g. a single shopping cart.
3. **Client library:** The driver applications should use a simple programming interface, in order to interact with the key-value store.
   The base API includes;
    1. init() - initialize the client session with the manager and other control path operations.
    2. put(key, value)
    3. get(key)
    4. finalize() - control path cleanup operations

   You can assume that  Max key size = 20 bytes, Max value size = 1 KB.

# Project Details

---

## Design Principles

1. **Data Partitioning**
   GTStore aims to be a highly scalable key value store, thus storing all the data in a single node will not work. Ideally, the system should be able to serve an increasing number of client requests in proportion to hardware resources we provision (number of data nodes). Data partitioning schemes divide the responsibility of managing data elements across the data nodes. For this section you will design and implement;
   1. A data partitioning scheme to partition incoming data across data nodes.
   2. Data insert/lookup algorithm/protocol for the selected partitioning scheme in the context of your system design.
   3. Discuss pros and cons of your design.
2. **Data Replication**
   GTStore maintains K number of data replicas for any stored object. Replication increases fault tolerance of the system and under certain consistency schemes increase the system availability. In this section you will design and implement data replication mechanism for GTStore.
   1. Clearly describe the replication protocol/set of events during data insert/lookup.
   2. How does your replication mechanism works alongside with the proposed data partitioning technique? (e.g. node with the primary replica fails).
3. **Data Consistency**
   Data consistency scheme plays a major role in defining the availability of any data store. Strong consistency schemes will not benefit GTStore design as it may make the system unavailable during replica failures. Therefore GTStore implements relaxed consistency semantics. For instance, your consistency protocol may act as follows:
   1. Versioning scheme for the stored objects.
   2. Inserted objects get stored on K (replication parameter) nodes within a bounded time.
   3. Read operations are guaranteed to receive the latest version of the requested object.
4. **Handling Temporary Failures (Extra Credit - 15 points)**
   Design and implement a technique to handle temporary node failure and rejoin. The proposed system should include,
   1. Online detection of node health. (e.g. detection using heart beats)
   2. Membership management during node leave and join

Preserving the original system properties related to replication and consistency of the data.

## Code Structure

---

We've provided a very basic code skeleton and run script so as to give you a usage example. Feel free to make any necessary modifications to the classes and function parameters defined inside the given {.hpp, .cpp}. You can also modify the Makefile. There's only one restriction. Do not add any additional {.hpp, .cpp} files so that we have uniform submissions.

- `gtstore.hpp`
- `manager.cpp`
- `storage.cpp`
- `client.cpp`
- `test_app.cpp`
- `Makefile`
- `run.sh`
- `RPC related meta-files if any`

**Implementation details**

- Code implementation in C/C++.
- You will model each node in the distributed system using a separate process running on a single machine.
- Avoid using shared memory or file system to communicate between nodes. It has to be a network call, so your GTStore can work when deployed across many machines.
- You may use sockets for communication between nodes.
- You are also allowed to use any support for Remote Procedure Calls (RPC). Look at the "References" Section for choice of framework. Note that if you go this route, you will likely benefit only from the stubs for the message buffer management (i.e., sending/receiving the key value pairs), but you will still need to orchestrate the distributed interactions among the client(s) and server(s) by yourself.
- Demonstrate the interaction of clients and GTStore inside test_app.cpp and run.sh
- You are not allowed to use any other third party libs without explicit permission.

## Evaluation - Scoring Matrix

---

| Code & Test Cases | Report | Demo | Piazza Discussion | Total |
|---|---|---|---|---|
| 50 | 20 | 25 | 5 | 100 |

## Submission Instructions

---

**Please see the submission practice document in the project directory.**

**<u>The tar file should follow the following structure</u>**
**Tar Name** project_<i>_<gt_handler>.tar.gz

project_4_gburdell3.tar.gz
- code/
- project_<i>_report_<gt_handler>.pdf

## Report & Deliverables

---

Your report is as important as your code implementation and should include an elaborate description of the design choices and functionality. Please include screenshots of the relevant log messages for every functionality. Additionally, address the questions described in Sections 2 and 3 formatted in the following way:

- **System Components**
  - GTStore Centralized Manager.
  - GTStore Storage Node.
  - Client API calls. (one-by-one)
- **Design Principles**
  - Data Partitioning
  - Data Replication
  - Data Consistency
  - Handling Temporary Failures (if implemented)
- **Client Driver Application**

Describe anything inside `test_app.cpp` and `run.sh`.

- **Design Tradeoffs**

We often have to tradeoff one aspect of the system, in favor of another. Describe one interesting property of your designed data store (e.g., lightning fast responses, linear scalability, etc.) and identify how that design decision impact the performance of the rest of the system/ particular aspect of the system (e.g., lowered throughput, increase in storage space/replicas, etc.).

- **Implementation Issues**

Describe any restrictions or faults your current implementation is having, if any.

**Please include all results in the report to be considered for evaluation. Each feature implementation needs to be supported with test case/graphs to demonstrate their working to receive full credit.**

### Deliverables

Follow these strict guidelines for the project submission:

One team member submits `team.txt` including the names of the team members.
The other one compresses and submits the following inside

Do not include any object files or executables. Your code should compile and run on the provided class servers by executing the `run.sh` file. If you're not able to run it on the class servers, please inform the TAs.

## Resources

---

Refer to the following papers, which you can find in `canvas/files/papers/`:

- **Dynamo: Amazon's Highly Available Key-value Store.** Go through the described design principles that a distributed key-value store should feature. Your implementation, however, need not be the same as the paper.
- **RPC options**

  Choose one of the following. Spend time to familiarize with their usage.
  - http://xmlrpc-c.sourceforge.net/
  - https://docs.oracle.com/cd/E19683-01/816-1435/index.html
  - https://grpc.io/docs/tutorials/basic/c.html

## Deadlines

---

- **Release Date** : **Saturday, November 9, 2019**
- **Due date - Sunday, December 8, 2019 11:59 PM - Final report and code**

## Communication Policy

---

Our effort is to ensure we can address the problems of all students in a prompt manner and students help each other (while honoring the Georgia Tech honor code as mentioned above). Additionally, we would like to make sure everyone is aware of the solutions to general doubts

and problems faced as you work on your projects. Thus, to have transparent communication we are following the below given policy -

1. All the questions related to project/class must be asked on the Piazza.
2. Emails to the TAs should be done in cases such as submission issues or unless asked due to an issue affecting an individual only.

**If an email is sent -**

1. If an email is sent - make sure the email is sent to both the TA's
    a. Ranjan Sarpangala Venkatesh | ranjansv@gatech.edu
    b. Harshit Daga | harshitdaga@gatech.edu
2. To ensure emails are answered in a timely manner we have our filters set for the subject pattern [CS-6210]. Please make sure to add [CS-6210] before your subject.
   For example [CS-6210] Is George P. Burdell real?

## Late Submission

---

1. A flat penalty of 30% for late submission will give you an extension by 3 days.
2. To avail late submission - email must be sent to the TAs within 12 hours of the deadline.
3. TAs will let you know about the late submission process.