COSC 3351: Advanced Data Structures

Spring 2020

Course Project (Individual Effort)

Distributed Data Structures (DDS)

Assigned: 04/02/2020

What is a distributed system?

With the ever-growing technological expansion of the world, distributed systems are becoming more and more widespread. They are a vast and complex field of study in computer science.

Various definitions of distributed systems have been given in the literature, none of them satisfactory, and none of them in agreement with any of the others. For our purposes it is sufficient to give a loose characterization:

A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.

This definition refers to two characteristic features of distributed systems. The first one is that a distributed system is a collection of computing elements each being able to behave independently of each other. A computing element, which we will generally refer to as a **node**, can be either a hardware device or a software process. A second element is that users (be they people or applications) believe they are dealing with a single system. This means that one way or another the autonomous nodes need to collaborate. How to establish this collaboration lies at the heart of developing distributed systems. Note that we are not making any assumptions concerning the type of nodes. In principle, even within a single system, they could range from high-performance mainframe computers to small devices in sensor networks. Therefore, in the project we will mimic the behavior of a distributed system on a small scale, where several nodes will be connected using a client-server system model.

What is a Distributed Data Structure (DDS)?

Distributed data structures (DDS): a DDS has a strictly defined consistency model: all operations on its elements are atomic, in that any operation completes entirely, or not at all. DDS's have one-copy equivalence, so although data elements in a DDS are replicated across multiple servers, clients see a single, logical data item. At all times, all replicas are coherent, and thus all clients see the same image of a DDS through its interface.

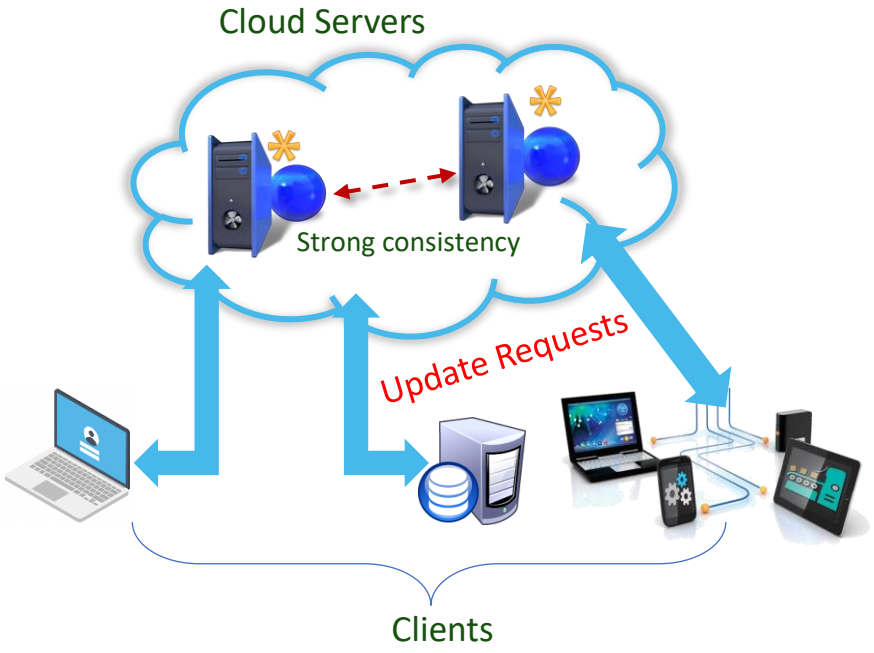
What is this project all about?

In this project, you will integrate several concepts that has been covered in the COSC 2351 and COSC 3351 course series. You will have hands-on experience in using one of the generic data structures that we designed in this course, use this data structure in a DDS application, apply data

structures permanent persistence through disk storage and use sockets programming and multi-threading to simulate a client-server communication model.

In this project, the servers in the client-server model oversee a data structure that is being replicated across all servers. We will assume a strong-consistency model across all servers (i.e. the data structure replicas are always consistent and coherent). Clients will connect to the servers to issue update requests to the data structures (i.e. delete, add or insert). Servers will cross communicate to push updates to one another and maintain consistency.

Figure 1 demonstrates the client-server system-model.



Using the client-server multithreaded sockets programming code provided for you, you need to implement the following functionalities in your DDS project:

1- Servers can communicate to each other by maintaining a list of all the available servers on the network. The list should contain information about the server IP addresses and open port numbers.

2- Servers maintain local copies of the data structures in memory and on disk. The disk copies are not necessarily consistent at all times with the local memory copies. A "Commit to disk" instruction issued by a client machine to the server, would force the server to serialize the data structures from memory to disk after ensuring that it has the latest version of data structures updates (by consulting all other servers for the latest data structure update).

3- Servers maintain along with the data structure, a log of when was the last update issued to the data structure. The log should include the following information: who issued the update,

what was the update, and the timestamp of when this update was enforced on the data structure.

4- Client machines can connect to any of the available servers and issue the following commands:

   a. Retrieve (view) the data structure.
   b. Update the data structure (delete, add, or insert a data item).
   c. Commit to disk
   d. Rollback to an older version of the data structure (i.e. load an older version from disk into memory to override the existing version)

   Servers receiving the above commands, need to ensure that the client is receiving (viewing) the latest updated data structure across all servers, that any update issued to any of the servers is being pushed to all other servers, and finally before committing to disk, all servers commit a consistent version of the data structure.

More specifics:

To simplify implementation and testing of the code we will assume the following:

1- Our DDS system consists of two servers and 4 client machines.

2- Clients can simultaneously establish connections to the different available servers, but only one client will issue an update or commit request to a server at a time. This assumption is made to avoid the need to synchronize the execution of the different threads while they are accessing the common data structure.

3- We will assume that only one data structure (a linked list of Integers) is maintained in memory by both servers. You can use the generic linked list implementation we did in COSC 2351.

4- Rollback can only go back one version from the latest one in memory.

What do you need to do:

Before you start implementation, you need to design your system first. The design will include the consistency protocol that you will use to ensure all servers are maintaining the latest updated copy of the data structure at all times, which means any update request issued to any server should be pushed to the other server. You should define a protocol for each of the **four** possible commands issued by any client (View, update, commit and rollback). To describe your protocol details, you will use the standard Sequence Diagram notations. The design will also include a Class diagram, showing all classes and interfaces that will used in the implementation of the system along with

the interaction between the classes. There should be two class diagrams, one for the server side and another for the client side.

See link below for videos explaining both Class and Sequence Diagrams with examples:

Class Diagrams: https://www.youtube.com/watch?v=UI6lqHOVHic

Sequence Diagrams: https://www.youtube.com/watch?v=pCK6prSq8aw

Once you have your protocols designed and classes and objects defined, you can start implementing your code.

Best programming practice is to incrementally build and test one function at a time. Start by establishing and testing connections between client-server and server-server. Next design the commands that the client will use to interact with the server. For example, your update command can look something like:

- Add: 100
- Delete: 20
- Insert: 10, 300 (i.e. insert in position 10 value 300)

Implement your consistency protocol that pushes updates to all servers. Implement your serialization code (read and write from disk storage with version control).

Finally, when all the main functionalities are created, you will need to identify how to demo the code to the instructor. While grading your code I want you to demonstrate the following:

- Connections established successfully
- Client requests handled correctly
- The data structure at all servers is in a consistent state. (Put some thought in how you will demo this part).
- Rollback loads the last committed version to disk into memory

Deliverables

1- **System Design - Due: 04/16/2020**

- Your class and sequence diagrams. Submit those to blackboard and you will present your deign and protocol in class as well.

2- **Final project delivery and demo - Due: 04/30/2020**

- Present your work in class. You will share your monitor during the Zoom session and demonstrate all the functionalities in your code. Upload your code to BB.