CS4210/CS6210:Project 4 - GTStore

Due Sunday April 22 at 11:59 pm

Goals

In this project you will implement a distributed key-value store (GTStore). You should give special attention to properties such as (i) scalability of overall system, (ii) availability of the system, and (iii) system resilience to temporary node failures (reliability).

System components

1. **GTStore**
    a. **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:
        i. Membership management
        ii. Load balancing of clients across replica nodes
        iii. 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)
    b. **Storage 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 can be an array of shopping cart items. E.g. {soap, toothpaste, shoes, soap} and the key can be the client-id. One application process should manipulate a single key-value pair at a time.
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;
    a. init(&env) - initialize the client session with the manager and other control path operations. 'env' stores the session info related to current client session.
    b. put(&env,key, value)
    c. get(&env,key)
    d. finalize(&env) - control path cleanup operations

    You can assume that Max key size = 20 bytes, Max value size = 1 KB and env is a data structure defined in gt_client.h file to store any system related metadata.

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 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;

   a. A data partitioning scheme to partition incoming data across data nodes.
   b. Data insert/lookup algorithm/protocol for the selected partitioning scheme in the context of your system design.
   c. 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.

   a. Clearly describe the replication protocol/set of events during data insert/lookup.
   b. 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 makes the system unavailable during replica failures. Therefore GTStore implements eventual consistency semantics. In this section you will design and implement:

   a. Versioning scheme for the stored objects.
   b. Object puts/gets are executed on majority of data nodes before deemed successful.
   c. Inserted objects get stored on K (replication parameter) nodes within a bounded time.
   d. Read operations are guaranteed to receive the latest version of the requested object.
   e. Detecting conflicts due to concurrent updates on the object replicas. (optional)

4. **Handling Temporary Failures (Extra Credit)**

   Design and implement a technique to handle temporary node failure and rejoin. The proposed system should include,

   a. Online detection of node health. (e.g. detection using heart beats)
   b. Membership management during node leave and join
   c. Preserving the original system properties related to replication and consistency of the data.

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 details

- Preferred program languages are C/C++ and Java.
- 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/remote method invocations. For instance, for C/C++ you may use XML-RPC or Sun RPC, or for Java use RMI. 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.
- You are not allowed to use any other third party libs without explicit permission.
- You will submit a report and the implementation code. Follow the submission guidelines provided.
- The report is as much as important as the working code.

References

- Please refer to the "*Dynamo: Amazon's Highly Available Key-value Store"* paper, in order to understand the design principles that a distributed key-value store should feature. Your implementation, however, need not be the same as the paper.
- XML-RPC: http://xmlrpc-c.sourceforge.net
- RPC:
    - full documentation at Oracle: https://docs.oracle.com/cd/E19683-01/816-1435/index.html;
    - many online tutorials with simple examples on basic steps -- create interface description (.x), compile stubs (rpcgen), call autogenerated functions from your code.