CS 677 Distributed Operating Systems

Spring 2015

Programming Assignment 3: Internet of Things - Fault tolerance, Replication, and Consistency

Due: 5pm, Wed April 29, 2015

- You may work in groups of two for this lab assignment.
- This project has two purposes: first to familiarize you with concepts in fault tolerance, replication and consistency.
- You can be creative with this project. You are free to use any programming languages (C, C++, Java, python, etc) and any abstractions such as sockets, RPCs, RMIs, threads, events, etc. that might be needed. You can also build on the code that you wrote for the previous lab if it is convenient. You have considerable flexibility to make appropriate design decisions and implement them in your program.

A: The problem:

- The goal of this project is to implement fault tolerance and replication in the system and also consistency. Assume that the IOT gateway implements a multi-tier architecture like before. You goal is to replicate the gateway so that there are two replicas (each front-tier replica communicates with its own backend database tier replica). Like before, the system has a number of sensors or smart devices which are NOT replicated. A sensor or a smart device can communicate with either gateway replica. At start-up time, design a technique to associate each sensor or device with either replica such that the number of devices / sensors communicating with either replica is roughly equal. That is, you should not hard code the address of the gateway replica in the sensor / device but choose it dynamically at startup time to balance the load.

  It is assumed that any replica can serve any sensor / device. the gateway replicas implement a consistency technique to ensure that their states (e.g., database states) are syncronized. You may choose any connsistency mechanism for this purpose but be sure to clearly describe the algorithm used in your design document and also discuss the consistency semantics provided by your chosen approach.

  Next, implement a cache in the front-end tier to enhance performance of the gateway. The cache will store all recently accessed data items / query results from the database in the in-memory cache. When the front-end tier needs to make a request to the database tier, it should first look in the in-memory cache to see if the results are already cached (and if so, use it). In the event of a cache miss, the front tier should make a request to the database tier like before. Assume that the cache can store up to maximum *N* item (N should be configurable). Implement a simple cache replcaement startegy such as LRU or least frequently used policy to evict cached items when a new item needs to inserted in the cache and the cache is full.

  Since the cache holds copies of certain data from the database, you should extends your cache

consistency technique to handle replicated data items in the cache as well as that in the database replicas.

Since the gateway is replicated, you should also make your gateway fault tolerant. It is sufficient to handle crash faults (Byzantine faults need not be handled). Also for simplicity, assume that the both tiers of a gateway replicas fail at once and in this case, the other gateway needs to take over the functions of the failed replica. A gateway node needs to dynamically determine the failure of the other replica (this can be done by any method that you choose such as exhanging "I am alive" heartbeat messages). Upon detecting a failure, the remaining gateway replica implements a failure recovery algorithm thay involves taking over the responsibility of servicing all sensors and devices that were communicating with the failed replica. Your failure recovery method needs to inform the sensors / devices of the failure and have them reconfigure themselves to communicate with the new replica for subsequent requests. While the failure recovery "algorithm" can be straightforward, clearly document how failures are detected and all the steps your replica peforms to take over the functions of the failed gateway. Also explain if failiures can lead to any data loss in your system (which will depend on the choice of your consistency mechanisms that syncronize state between the replicas) and the impact of any such data loss. Implement your replication, caching, cache consistency and fault tolerance techniques in your code. This lab does not need vector, logial clocks or leader election aspects of lab 2 and it is fine to simply assume that clocks are syncronized and simple timestamptsfor determining event ordering.

Design for Paxos: The final part of this lab requires you to provide a design if you were asked to implement Paxos in this sytem (you only need to writeup a high-level design / algorithm in your Design doc and do not need to implement the algorithm). Assume that there are *k* gateway replicas and that each request is sent to all of them and the replicas run Paxos to reach agreement on the answer before providing a reply to a request. How might such a system work? Explain clearly how the Paxos algorithm can be used by your gateway nodes and you would have implemented it in your current design. Do not blindly cut and paste thr algorithm from the class slides or from the Internet - you are expected to gain some faimiliarity with it and come up with a design that uses Paxos. Provide a writeup of your design with the main design document (no implementation is necessary to get credit for this part).

Extra Credit: This part is optional. For extra credit, implement your Paxos design in the gateway nodes and conduct simple experiment to demonstrate it works (e.g., the system functions even when nodes fail or one of the node produces an incorrent answer).

Requirements:

1. You need to implement all the mandatory parts: replication, caching, consistency and fault tolereance and provide a design of Paxos.
2. The extra credit part of the lab is optional.

- **Other requirements:**

  No GUIs are required. Simple command line interfaces and textual output of scores and medal tallies are fine.

  You are free to develop your solution on any platform, but please ensure that your programs compile and run on the edlab machines (See note below).

B. Evaluation and Measurement

Deploy two gateway replicas and a few sensors

1. Start your system with different numbers of sensors and show how your system "balances" the load from these sensors across the two replicas.
2. Conduct a series of simple experiments to demonstrate the working of your system with and without caching. MEasure the response times of requests with and withotu caching and for cache hits and cache misses. Be sure to show the impact of any cache inconsistency if you have used a technique that does not provide strict consistency guarantees.
3. Conduct simple experiments to inject a gateway failure and demonstrate how your system recovers from the failure. Repeat your experiment by injecting failures at different points and show that your system can recover in each case. Meaure the time needed to detect a failure, the time to recover from a failure once it has been detected and also show whether the failure can cause a sensor / device push/pull messahe to get lost and the impact of any such loss.

   Make necessary timeline plots or figures to support your conclusions.

- ## C. What you will submit

- When you have finished implementing the complete assignment as described above, you will submit your solution in the form of a zip file that you will upload into moodle.
- Each program must work correctly and be **documented**. The zip file you upload to moodle should contain:

1. An electronic copy of the output generated by running your program. Print informative messages when a client or server receives and sends key messages and the scores/medal tallies.
2. A seperate document of approximately two pages describing the overall program design, a description of "how it works", and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made). You also need to describe clearly how we can run your program - if we can't run it, we can't verify that it works.
3. A program listing containing in-line documentation.
4. A seperate description of the tests you ran on your program to convince yourself that it is indeed correct. Also describe any cases for which your program is known not to work correctly.
5. Performance results.

- ## D. Grading policy for all programming assignments

1. Program Listing
   works correctly ---------- 50%
   in-line documentation -------- 15%
2. Design Document
   quality of design and creativity ---------- 15%
   understandability of doc ------- 10%
3. Thoroughness of test cases ---------- 10%
4. Grades for late programs will be lowered 12 points per day late.

- ## Note about edlab machines

- We expect that most of you will work on this lab on your own machine or a machine to which you have access. However we will grade your submission by running it on the EdLab machines, so please keep the following instructions in mind.
- You will soon be given accounts on the EdLab. Read more about edlab and how to access it here
- Although it is not required that you develop your code on the edlab machines, we will run and test your solutions on the edlab machines. Testing your code on the edlab machines is a good way to ensure that we can run and grade your code. Remember, if we can't run it, we can't grade it.
- There are no visiting hours for the edlab. You should all have remote access to the edlab machines. Please make sure you are able to log into and access your edlab accounts.
- IMPORTANT - No submissions are to be made on edlab. Submit your solutions only via moodle.

Stumped?

1. Stumped on how to proceed? Better yet, ask the TA or the instructor by posting a question on the Piazza 677 questions. General clarifications are best posted on Piazza. Questions of a personal nature regarding this lab should be asked in person or via email.