Master of Software Systems Program

CICS 525 | Real-Time and Distributed Systems | Summer 2011

Assignment #3

Amazon AWS

For this assignment, we will use Amazon's cloud computing offering (AWS). In particular, to obtain distributed computing nodes, we will use EC2 (the Elastic Compute Cloud). EC2 allows one to create and start remote nodes on demand. These nodes are, typically, virtual machines with some computing and storage capacity. EC2 nodes or instances can be controlled using a web service interface.

EC2 offers many different instance types based on computing power, memory and storage capacity, but for this assignment we will restrict our use to the standard small instance with 1.7 GB memory, 1 EC2 compute unit, 160GB local instance storage and 32-bit microarchitecture support. We will also use Amazon Linux AMI as the machine image for the EC2 instances.

Students will be provided with access credentials for AWS via email or a similar mechanism.

Amazon's cloud computing services offer many different features, which make this assignment trivial to implement, but we will focus on EC2 alone as the purpose of the assignment is to improve one's understanding of how robust and scalable distributed applications can be built.

Distributed Shared Folders (SyncDir)

Using Amazon's EC2 you will implement a mechanism, called SyncDir, for synchronizing a specific folder (directory) on an end-user's system with a cloud-based service (think Dropbox). An end-user will select a folder on one of her machines and that folder should be replicated on one or more EC2 instances. This folder should also be synchronized across other machines that the end-user may choose to run the SyncDir application on.

This simple service raises several issues that would need to be handled and constitute different components of this assignment. (The contribution of each aspect to the overall grade is also indicated.)

0. **Basic synchronization (25%):** This is the first step of the project. Users can designate a folder to keep synchronized and this folder is synchronized across different machines that the user may choose. You should implement an authentication scheme but a simple authentication scheme will be sufficient.
1. **Server-side failures (35%):** If a folder is replicated on only one EC2 instance then the failure of that instance will render the folder inaccessible from a new machine or data will be stale across machines. To this end, one should use some form of replication on the server-side to protect against failures. The system architecture should be capable of tolerating two server-side instance failures. The system should also take suitable actions when a failed instance restarts. (You will be

asked to test and demonstrate this capability by forcibly shutting down server instances to simulate failures.)

2. **Simultaneous editing (20%):** If two users share the same folder (or if the same user has the folder synchronized across multiple machines) then users may edit a file concurrently. You will have to make a suitable design choice as regards resolving this situation. One approach may be to have a unique lock per file and only a user that holds the lock may perform writes. Another alternative is to support different versions and inform the users about conflicts and then allow them to resolve it directly. (Other options are also possible. You will have to describe the design choice and provide use-cases and justifications for the choice.)
3. **Bandwidth efficiency (15%):** Users may synchronize data over low data rate links (e.g., Edge or 3G network links). To make the most efficient use of resources, the SyncDir service should detect changes and only communicate updates (instead of complete files) whenever possible.

You do not need to implement GUIs at the client or server side although one would expect that a production system would have a client-side GUI for users to manage data synchronization and sharing. A server-side GUI may allow a system administrator to view the status of the system and also examine performance statistics. You may, however, choose to implement GUIs and other features for extra credit. A feature that Dropbox supports and can be implemented would be to allow users to share sub-folders.

You should use Java for your server-side and client-side implementations.

Submission

You must submit your assignment using `handin`. The name of the course is `cics525` and the assignment name is `assignment3`. You must submit the following files:

1. All source code files;
2. A makefile with compilation instructions: Makefile;
3. A detailed report that must contain (at the least):
   a. Names and student numbers;
   b. Description of the service and the client- and server-side implementations;
   c. Rationale for the design choices;
   d. Known bugs;
   e. Possible extensions.
4. The report need not contain performance evaluations as regards scaling behaviour (number of users, files, etc.) although such evaluations or qualitative studies would merit additional credit.

   Compilation instructions

   Execution instructions

   Known bugs

Demo

You will demonstrate your work to either the TA or the instructor and a signup notice will be sent closer to the due date. The demo itself is worth 5% of the grade but you will not be assigned a grade without a demo.

Comments

The purpose of assignments such as this is to provide a small and yet interesting distributed application to develop. Some aspects of the assignment may not be fully specified. That is, in some cases, a deliberate choice so that you can decide on the architecture. You are welcome to discuss your ideas and doubts with the instructor or the TA. You are also encouraged to think of suitable pricing models for the service that you will develop.

References

- Amazon EC2 documentation: http://aws.amazon.com/documentation/ec2/
- Amazon Linux AMI: http://aws.amazon.com/amazon-linux-ami/