INF3200: Distributed Systems Fundamentals
Mandatory Assignment 1

September 3, 2019

Introduction

This project will provide you with hands-on experience on designing and implementing a distributed system. You will implement a key-value store, and investigate the characteristics of your chosen design.

Distributed Data stores

A *distributed* data store is a network of computers providing a service for storing and retrieving data. Compared to a single machine system, distributing the data across multiple machines may improve performance, increase scalability and/or provide better fault-tolerance. These benefits come with the cost of higher complexity and a bigger developing effort. A *key-value* store is a type of distributed data store designed for storing and retrieving records identified by a unique key.

Processes across multiple machines provide a single service, passing messages between one another. The processes communicate according to the rules of the chosen architecture. Structured and unstructured *peer-to-peer* architectures are both suitable architectures for building a distributed data store. A distributed hash table is the most common way of organizing the processes in peer-to-peer systems.

The data is distributed between the nodes through a partitioning scheme. The partitioning scheme dictates which node is responsible for storing a given key-value pair <k,v>. A simple solution would be to assign <k,v> to a random node, with the cost of lookup being O(N) given N nodes. A more refined solution is to assign nodes with the responsibility of a key-space partition (e.g. assign node1 with keys in the range a-p node2 with q-z). Other solutions include consistent hashing (used in the Chord [1] protocol). For more information on these types of systems, you can read about DHTs in the textbook or look at Windows Azure Storage [2], Dynamo [3] and Cassandra [4].

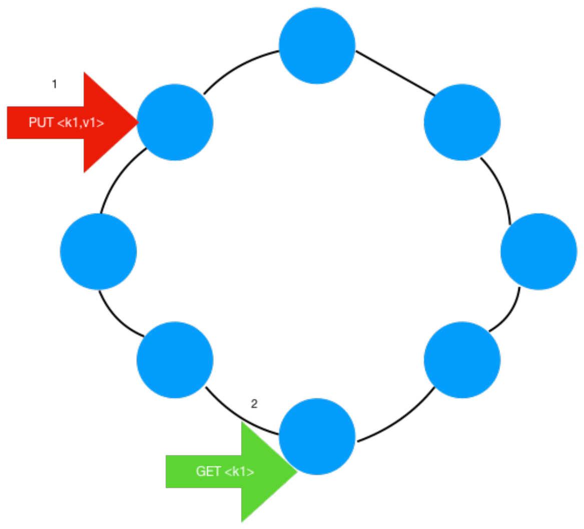

Figure 1: Expected system's behavior, values inserted at any node are available by querying at any of the nodes in the system.

Client

A client application issues PUT and GET requests to the storage service. The frontend component of the storage node is responsible for forwarding any requests to the other nodes that may hold the data.

We provide sample code to get you started. The client application runs a series of PUT and GET requests to simulate client behavior.

Storage nodes

Upon receiving a request from a client, the storage node is responsible for completing the requested operation. This likely involves contacting other nodes to handle the request.

API

Please extend the API of your network with the following calls:

- **PUT /storage/<key>**: Store the value (message body) at the specific key (last part of the URI). PUT requests issued with existing keys should overwrite the stored data.

- **GET /storage/<key>**: Retrieve the value at the specific key (last part of the URI). The response body should then contain the value for that key.
- **GET /neighbours**: Return a JSON object with neighbouring nodes i.e. predecessor and successor nodes in a chord [1] system.

Key Requirements

1. Distribute data storage load between storage nodes according to the Chord [1] consistent hashing algorithm.
2. Support running the service with a minimum 16 nodes. We require you to run approximately 16 nodes at the demo session. Note that we do not require you to support nodes joining and leaving when serving storage requests.
3. Any (random) node in the network should be able to serve incoming requests. I.e. you need to forward GET and PUT requests to the node responsible for storing the data according to the hashing algorithm.
4. Store the data in-memory (please avoid storing any data on disk)
5. The report must contain a graph showing the throughput as transactions per second (y-axis) of the system with 1, 2, 4, 8 and 16 nodes (x-axis) for both PUT and GET.
6. The report should be approximately 6-10 pages long.

Hand-in

The delivery must include:

1. **Source code** (programming language of your choice[1]) with instructions on how to run.
2. **Report**

For the report, we expect you to follow structure similar to the following

- Introduction - describe your task.
- Design - description of your architecture, communication requirements, lookup algorithms.
- Experiments - describe your experiments with different metrics and what they attempt to evaluate. Explain choice of metrics.

[1]and supported by uvcluster.cs.uit.no

- Discussion - discuss positive and negative sides of your solution. Have you critically evaluated your solution?
- Conclusion - conclude your findings.
- Do not forget to

  Make your figures clear and understandable.

  Cite references, insert captions and axes descriptions.

Other things

- There will be a demo presentation during the colloquium following the deadline. Present your work and demonstrate it.
- You share the cluster with multiple students, so please try to keep resource consumption to a minimum. You should add a *reasonable* time-to-live for each process so that it terminates after a given amount of time.
- Start early, fail early. (Make it better early.)
- Deadline **October 1, 2019**.

References

[1] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," IEEE/ACM Transactions on Networking (TON), vol. 11, no. 1, 2003, pp. 17–32.

[2] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci et al., "Windows azure storage: a highly available cloud storage service with strong consistency," in Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM, 2011, pp. 143–157.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in ACM SIGOPS operating systems review, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[4] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," ACM SIGOPS Operating Systems Review, vol. 44, no. 2, 2010, pp. 35–40.