Fall 2017, Inf-3200 Mandatory Project 2: Distributed Key-Value Store

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 implementation. You will extend the functionality of your existing system with the ability to store and retrieve data.

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].

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.

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 1200 words long. Try to emulate the structure of the Dynamo paper [3].

Hand in

In your report, briefly present the details of your approach to the assignment. Then discuss the details that are interesting to you. We want to see that you can assess the strengths and weaknesses of your implementation. Think outside of the context of the rocks cluster. How would it behave under different workloads such as more users or a higher number of requests per second?

The delivery must include:

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

Other Practical Details

- You are free to use any language supported by the cluster.
- Groups of two is preferred (contact us for other arrangements).
- There will be a demo presentation in connection with the hand in. This will occur during the colloquium following the deadline. We expect you to briefly present your work and demonstrate it.
- Deadline is **Monday, November 6.**

References

[1] Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.* 31, 4 (August 2001), 149-160. DOI=10.1145/964723.383071 http://doi.acm.org/10.1145/964723.383071

[2] Windows Azure Storage: a highly available cloud storage service with strong consistency. In*Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (SOSP '11). ACM, New York, NY, USA, 143-157. DOI=10.1145/2043556.2043571 http://doi.acm.org/10.1145/2043556.2043571

[3] Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles* (SOSP '07). ACM, New York, NY, USA, 205-220. DOI=10.1145/1294261.1294281 http://doi.acm.org/10.1145/1294261.1294281

[4] ”Cassandra: a decentralized structured storage system." *ACM SIGOPS Operating Systems Review* 44.2 (2010): 35-40. http://dl.acm.org/citation.cfm?id=1773922