CSE403: Advanced Operating Systems

Jacob Nelson, jjn217

HW1 – Feb 29th, 2018

**Introduction**

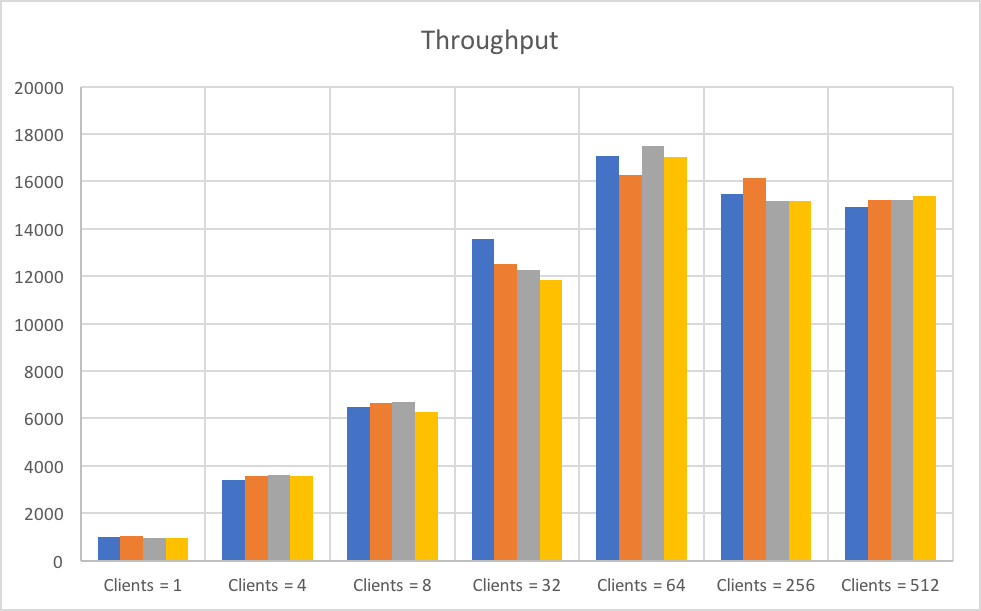
The aim of this project was to develop a distributed hash table (DHT). Each node in a cluster is responsible for some subset of the key value pairs. Together, they contain the entire hash table, but they do not have any direct knowledge other than their own local table, and where to send requests given a key that is not local to that node. My implementation is broken into two parts. A node library, used for creating and maintaining a node in the cluster, and a client library to interact with the existing DHT. The next section details their architecture.

**Architecture**

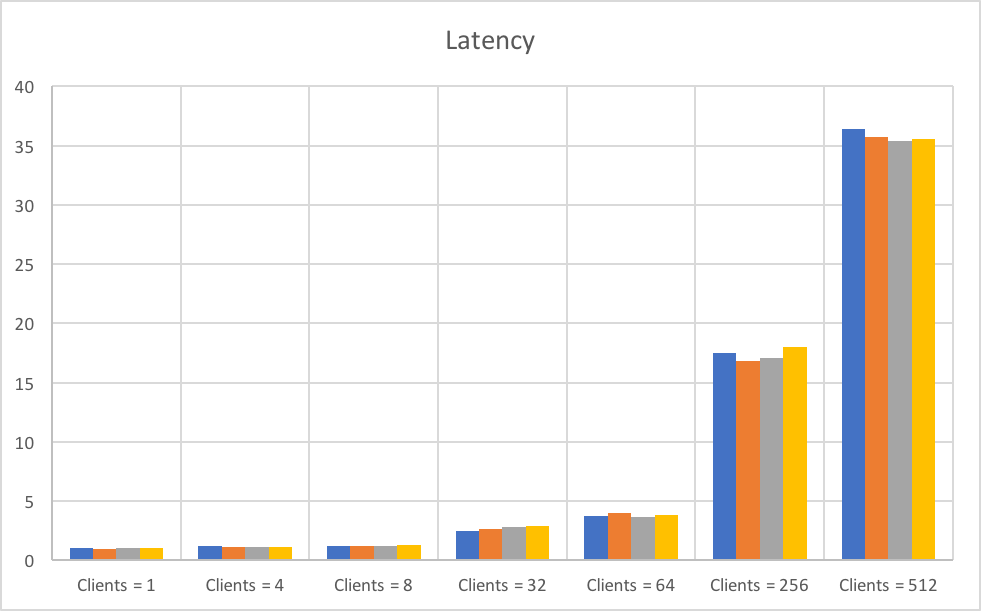
Each DHTNode contains an instance of a HashTable which, as per the specifications, can be added to and read from. Once a key-value pair is added to the table it cannot be edited. A few custom operations I included for easy of testing include a size function and a clear function. They are not safe and will not behave correctly if run simultaneously with put and get operations. The DHTNode also includes a Network object, which handles all communication with the other nodes. It has knowledge of the network mapping and is responsible for sending and receiving messages (from both the client and other nodes). Each node’s Network listens on a public for incoming connections from clients. When one is received, the Network parses the incoming message and passes it to the node for processing. The node then decides if it should be processed locally or on another node. In the local case, the node adds the key-value pair to the local HashTable and sends a response to the client. For a key that exists on a remote node, the network connects to that nodes private IP and passes a request as if it were itself a client. When the remote node is done handling the request the original recipient of the client message relays the message back. In this way, each node is responsible for delegating and responding to the messages it receives from clients.

The second piece of the implementation is the DHTClient, which is packaged as a client library. The client does not know any internal mappings of the DHT and only connects to a single IP address. In fact, in the configuration I tested, the cluster has private addresses for internal communication, which are not accessible to the client. Only put, get and size are available for the client. When initialized, the client opens a TCP connection with the desired node and maintains this connection throughout.

**Testing**

To test my DHT I wrote a simple program to spin up a configurable number of clients and connect to specific nodes in the DHT network. It is possible to provide the desired target node, which was necessary for testing. First, each node is started using a script to build and run a DHT Node. Next, client processes start and connect to the node on the same machine and starts sending requests. The number of clients is variable, as is the number of total requests and the size of the key range. For all tests, each client sends 10,000 requests to the address of the node with which it is co-located. Nodes are AWS instances.

Tests were conducted to measure the throughput and latency with different numbers of clients connecting to the DHT. The single client test is a measure of the network latency involved in the network itself. There is no contention so it is the fastest possible execution on the DHT. Latency is smallest for this configuration, but so is throughput. Increasing the number of clients both increases the latency and throughput. For each experimental setup, four different key ranges are tested. In general, there was very little difference between key ranges, however, at high client numbers some variation is observed.

It terms of latency, the single client experiment had the lowest. At first the latency increases very little. Up to eight clients total (two per node) the application can handle the load. After that there starts to be more strain until it the max throughput is obtained at 64 clients. This is the point at which it cannot handle the number of requests being sent and is saturated.

One can imagine that while the number of requests being processed at one time is doubled, the latency for each of those requests also doubles. This results in a stagnant throughput.

\*Code is available at [www.github.com/jacnel/go-dht](http://www.github.com/jacnel/go-dht)