Problems 1

Simple Analysis of Horizontal Scaling

This Assignment treats some properties of a simple horizontally-scaled key-value store as discussed in the first couple of Lectures. Such a system is made up of one or more server nodes. The system processes *client requests*, of the form `value = get(key)` or `put(key, value)`. Processing such a client request requires executing local `get()` or `put()` operations at one or more server nodes.

We shall consider sharded and replicated implementations as well as a hybrid of the two. In all cases, we shall assume

1. All server nodes are identical hardware, possibly running different software.
2. There is plenty of storage at each node, so you don't need to worry about nodes running out of space.
3. A local `get(k)` operation at any node requires 1 millisecond of processing. A local `put(k,v)` operation at any node requires 2 milliseconds. Use the following (simple if somewhat unrealistic) model: A node is a single-core 1 GIPS machine -- one billion instructions per second. A local `get()` takes one million instructions to complete; a local `put()` takes two million instructions. Scheduling and process switching are needed to keep the node busy, but you may assume their cost is negligible. You should assume the scheduler never allows a node to remain idle if there is work available for it to do.
4. The network is infinitely fast -- a node can send a message of arbitrary length at negligible cost. (Clearly, this assumption is not realistic.)
5. The throughput of the system is the expected number of client requests completed per second in steady state, given that a fraction *p* of the requests are `put()` rather than `get()` requests. We shall assume client `get()` and `put()` requests independently are distributed uniformly over the keys. So, for example, if `put()` requests make up 10% of all requests (p = 0.1), then in expectation `put()` requests make up 10% of the requests for each individual key (hence, in a sharded system, for each individual server node).

Under these assumptions, answer the following three questions:

1. A Sharded System

Suppose we have a sharded system made up of N nodes, S_0, S_1, ..., S_{N-1}. As discussed in Lecture, node S_i handles all keys `k` such that `hash(k)%N` = i, and clients are required to send a request for key `k` to the appropriate server node.

(a) What is the maximum throughput (in requests / sec) of a single node system (N = 1) with 10% `put()` requests (p = 0.1)?

(b) What is the maximum throughput (again in requests / sec, summed over all server nodes) of a system comprising ten nodes (N = 10) with 10% `put()` requests (p = 0.1)?

(c) What is the maximum throughput of a single-node system (N = 1) with 20% `put()` requests (p = 0.2)?

(d) What is the general throughput formula as a function of N and p?

2. A Replicated System

Now consider a replicated system consisting of N nodes. As discussed in Lecture, the server nodes are identical: every node S_i handles all keys, and if it receives a `put(k,v)` request from a client it performs the operation locally and also forwards the operation to all of the (N-1) other nodes, and does not return to the client until all N operations are completed. (**Note:** as discussed in Lecture, a naive implementation of this protocol does not guarantee consistency, even in the absence of failures. For this question you should ignore this.) Clients are are assumed to send every request to a server node chosen uniformly at random.

(a) What is the maximum throughput (in requests / sec) of a single node system (N = 1) with 10% `put()` requests (p = 0.1)?

(b) What is the maximum throughput (again in requests / sec, summed over all server instances) of a system comprising ten nodes (N = 10) as a function of *p*, the fraction of `put()` operations in the workload?

> **Hint:** This is a bit tricky. When a node receives a `put()` request, it performs the operation locally and forwards the request to the (N-1) other nodes, which also must execute the operation locally before it can be considered complete. So each `put()` is executed N times rather than just once. This significantly restricts the scalability of the system.

(c) Suppose 10% of the requests in the workload are `put()`s (p = 0.1). Suppose you can choose N arbitrarily (as large as you want!). What is the maximum achievable throughput?

3. A Hybrid System

Suppose a client can detect whether a server node has failed (e.g. by timeout) and resend its request to a different server node. Assume a model in which any node failure is eventually detectable by clients and other nodes. As we discussed in Lecture, the simple sharded system scales very well but cannot survive even a single node failure without data loss. In contrast, the replicated implementation doesn't scale but survives even if all but one of its nodes have failed.

This suggests we might achieve scalability together with some degree of fault tolerance by combining the two techniques. Let's choose a modest goal: we want a system that scales and is "2-resilient;" that is, the system can survive the failure of any two nodes.

(a) Describe how you could accomplish this.

> **Hint:** Use sharding for scalability, then instead of serving each shard with a single node, serve it with a replicated cluster that achieves the desired resiliency.

(b) Suppose 10% of the requests in the workload are `put()` requests. What is the maximum achievable throughput as a function of the total number of instances N?

Submit Your answers

Create a file, in `zip` archive format, named `solutions1.zip` This archive should contain a file, named "README," which may be in `.pdf` or `.txt` format. Your README file should contain answers to Questions 1-3 above.

Remark: The same structure will be used for turning in all CS5300 assignments: you will upload to CMS a `.zip` archive containing a README file with a top-level description of your solution, together with any additional files we want to see. Admittedly, in this case it is pretty trivial. Submit your `solutions1.zip` file using CMS by the specified deadline.