CS 425 MP 1

Design

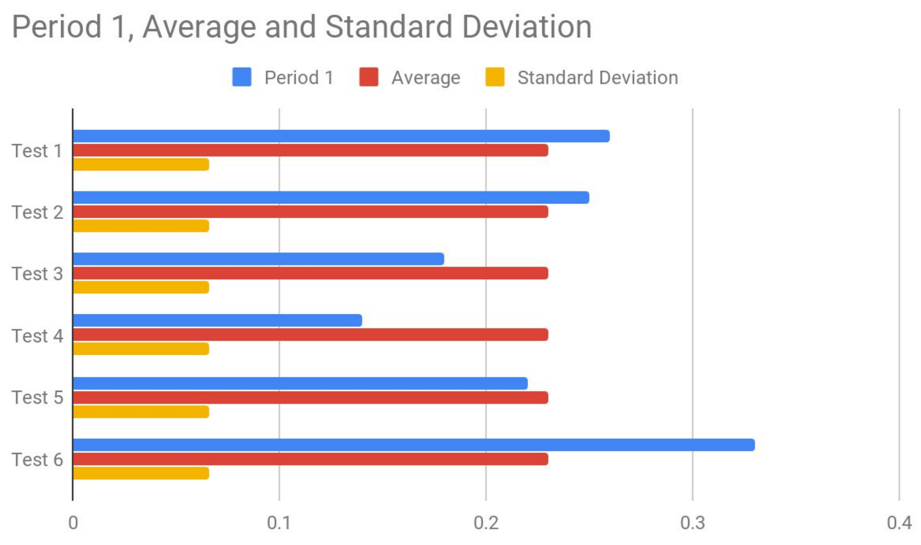
Our distributed log querier uses a parallelized client-server model to provide reliable distributed logging. Specifically, for each logging machine, an always-on background server is made available on a known IP address and port. This endpoint exposes a Go-RPC request endpoint that can be queried by the logging client. When a logging request is made, the client creates a shared message buffer and then spawns a single goroutine for every server specified in the configuration file.

Each goroutine calls a RPC subroutine, `queryServer()`, which calls the `Loggly.Retrieve()` subroutine exposed on each server. Internally, the subroutine finds and opens up the requested TXT file and filters lines that match the provided regular expression argument of the RPC subroutine. This subroutine returns a status indicating if an error occurred and a pointer to a filled struct which contains a list of every single line that matched the regular expression. With no errors the goroutine will empty the matched lines into a shared message buffer, which is then printed out by the main client thread, otherwise errors are printed to serverlog.log and the goroutine exits normally. This ensures a single failure will not break any other running servers.

Unit Tests

Using the built-in unit testing Go offers we test each package in our program by calling the main functionality of each package. This allows us to test that each part of the program correctly returns when given proper input, and fails when invalid input is given.

Experimental Data



Here is a graph of the runtimes of 6 passes of our program, querying all 10 VMs. The max was .33s, and the fastest pass was .14s. To test the speed we used the regex provided on Piazza in post @144.