MP1 Report

Design

We have both a daemon server process and a client command line tool for each node with a JSON file that contains all the IP and port numbers information for all peers in this distributed log system.

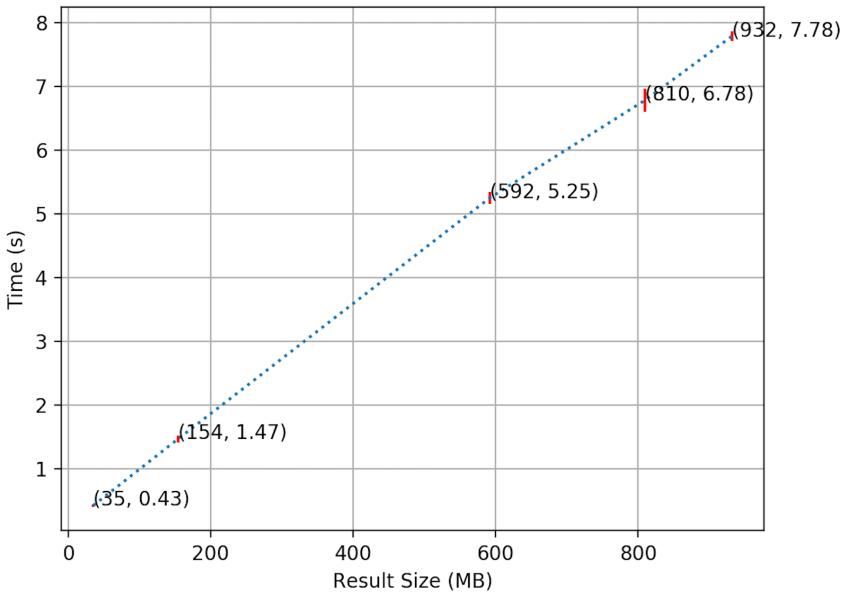
When a grep command is issued on any VM through ./client.bin <grep_pattern>, the client generates 1 goroutine for each node so that we can grep from all the server concurrently, instead of blocking each other. In each goroutine, the client initiates an RPC connection to the corresponding server with a timeout. If the connection is established, the client requests for log files through RPC and write the results to the pipeline. The server accepts grep commands from remote nodes through RPC calls and returns grep results for its local log files. If the server is dead, the goroutine will also write an error message into the pipeline. Then, the client will collect the grep results from the pipeline and output the final results.

UnitTest

For testing, we generate different random logs files with several lines contains known patterns and push the logs to each node. Then, we grep from all the nodes. We compare all the line count with expected one. If the line count for one node equals to the expected one, the test pass for this node. Otherwise, we say this node fails in the test.

Benchmark

For the benchmark, we have 10 machines each store 94-95 MB log files. We test the performance for 5 different grep patterns. The X-axis of the plot is the size of the grep result. The Y-axis is the total latency of the grep. We run 10 tests for each grep pattern. The center of every data point is pinpoint according to file size and the average latency. The red segment represents the standard deviation of the latency of the 10 tests. We saw a linear positive correlation between the latency and result size. We think it's because the bottleneck is the IO performance and the network bandwidth. So when the grep result getting bigger, the latency also raises.



Benchmark Plot