

MP1 Distributed Log Query System Report

Design Overview

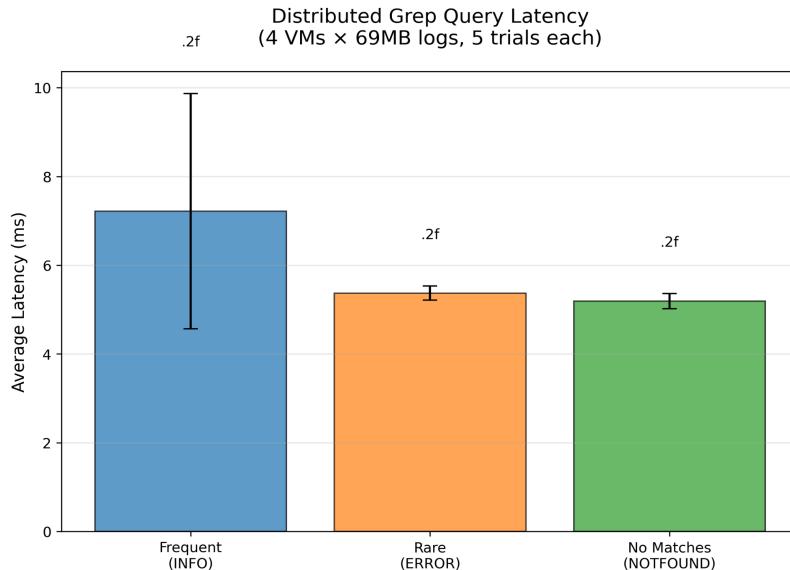
Our distributed log querying system uses a client-server architecture with RPC communication. Each of the 10 VMs runs a Go RPC server on port 12345 to handle grep queries on its local machine.*i.log* (where *i* comes from the hostname). The client executes queries in parallel, sending requests to all VMs via goroutines. Each server returns matching lines with file attribution, and the client aggregates results, showing per-file match counts and all matches. Queries run at the data location (avoiding file transfers), and parallelism provides fault tolerance since failed VMs are simply ignored.

Unit Tests

We implemented distributed unit tests in *test.go* that generate controlled test data across all VMs and verify correctness by comparing RPC query results against ground-truth SSH-based grep execution. Tests cover various scenarios: patterns present on all hosts, some hosts, one host, and frequent occurrences. The system achieves 100% accuracy when all VMs are operational, with graceful degradation when VMs fail (failed queries are excluded from pass/fail counts).

Performance Evaluation

We measured query latency across 4 VMs (9505-9508), each storing approximately 69MB log files with ~2.6M lines. Tests used 5 trials per pattern with a 2500ms timeout.



- Frequent patterns (INFO, ~2.6M matches total) average $7.22\text{ms} \pm 2.65\text{ms}$
- Rare patterns (ERROR, ~200 matches total) average $5.37\text{ms} \pm 0.16\text{ms}$
- No-match queries (NOTFOUND) average $5.19\text{ms} \pm 0.17\text{ms}$

The performance aligns with expectations: frequent patterns require processing all matches across large files, while rare/absent patterns allow grep to terminate early. Low standard deviations (<3ms) indicate consistent parallel execution across trials. This validates our distributed approach - running queries locally avoids the bandwidth cost of transferring 276MB+ of logs to a central location.