Speed Testing Report

TCP String Lookup Server

Author: Dean Robin Otsyeno
Date: 2026-02-11

Environment: Ubuntu Linux, Python 3.x

---

1. Introduction

This document presents the performance benchmark results for the **TCP String Lookup Server** implementation, as required by the Introductory Task specification.

The objective of the benchmark is to evaluate **per-query execution time** across multiple search algorithms under different operating modes, specifically:

- `reread_on_query = True` (disk-based lookups)
- `reread_on_query = False` (cache-enabled lookups)

The benchmark focuses on **execution time per query**, not throughput (QPS), in accordance with the specification requirements.

All benchmark results presented in this report are reproducible and backed by raw CSV data generated by the benchmark harness included in the repository.

---

2. Specification Requirements

The specification defines the following performance thresholds:

Mode	Requirement
`reread_on_query = True`	Average execution time ≤ **40 ms**
`reread_on_query = False`	Average execution time ≤ **0.5 ms**

These thresholds are evaluated against datasets of up to **250,000 lines**, as explicitly required.

---

3. Algorithm Classification

To correctly interpret the results, the implemented algorithms are grouped based on their **design characteristics**.

3.1 Disk-Based Algorithms

(Operate directly on the data file for each query)

These algorithms are valid when `reread_on_query = True`:

- `linear_scan` — sequential scan of the file

- `mmap_scan` — memory-mapped file scan
- `grep_fx` — external GNU grep invocation

These algorithms **do not build or retain in-memory caches** and are therefore not architecturally suited for sub-millisecond execution.

3.2 Cache-Based Algorithms

(Operate on preloaded in-memory structures)

These algorithms are valid when `reread_on_query = False`:

- `set_cache` — hash-based O(1) membership testing
- `sorted_bisect` — binary search over sorted in-memory data

Only cache-based algorithms are expected to meet the **0.5 ms** requirement.

---

4. Benchmark Methodology

4.1 Dataset Sizes

Benchmarks were executed against datasets of the following sizes:

- 10,000 lines
- 100,000 lines
- **250,000 lines** (primary reference for compliance)

4.2 Query Load

- **10,000 queries** per algorithm per configuration
- Approximately 50% hit ratio
- Queries are generated deterministically to ensure reproducibility

4.3 Execution Environment

- Python 3.10+
- Linux environment
- Single-process benchmark execution
- No artificial delays or throttling

---

5. Benchmark Command Used

The benchmarks were executed using the following command:

```
python3 -m benchmarks.benchmark_search --mode algo --sizes 10000,100000,250000
--queries 10000 --verbose
```

This command benchmarks **all supported algorithms**, for both `reread_on_query=True` and `reread_on_query=False`, and records per-query execution time metrics.

---

6. Results Summary (250,000 Lines)

6.1 `reread_on_query = True`

Requirement: ≤ 40 ms

Algorithm	Avg (ms)	p95 (ms)	Result
`linear_scan`	~26.6	~37.8	✅ PASS
`mmap_scan`	~3.1	~4.6	✅ PASS
`grep_fx`	~4.9	~6.7	✅ PASS

Observation:
All disk-based algorithms comfortably meet the 40 ms requirement.

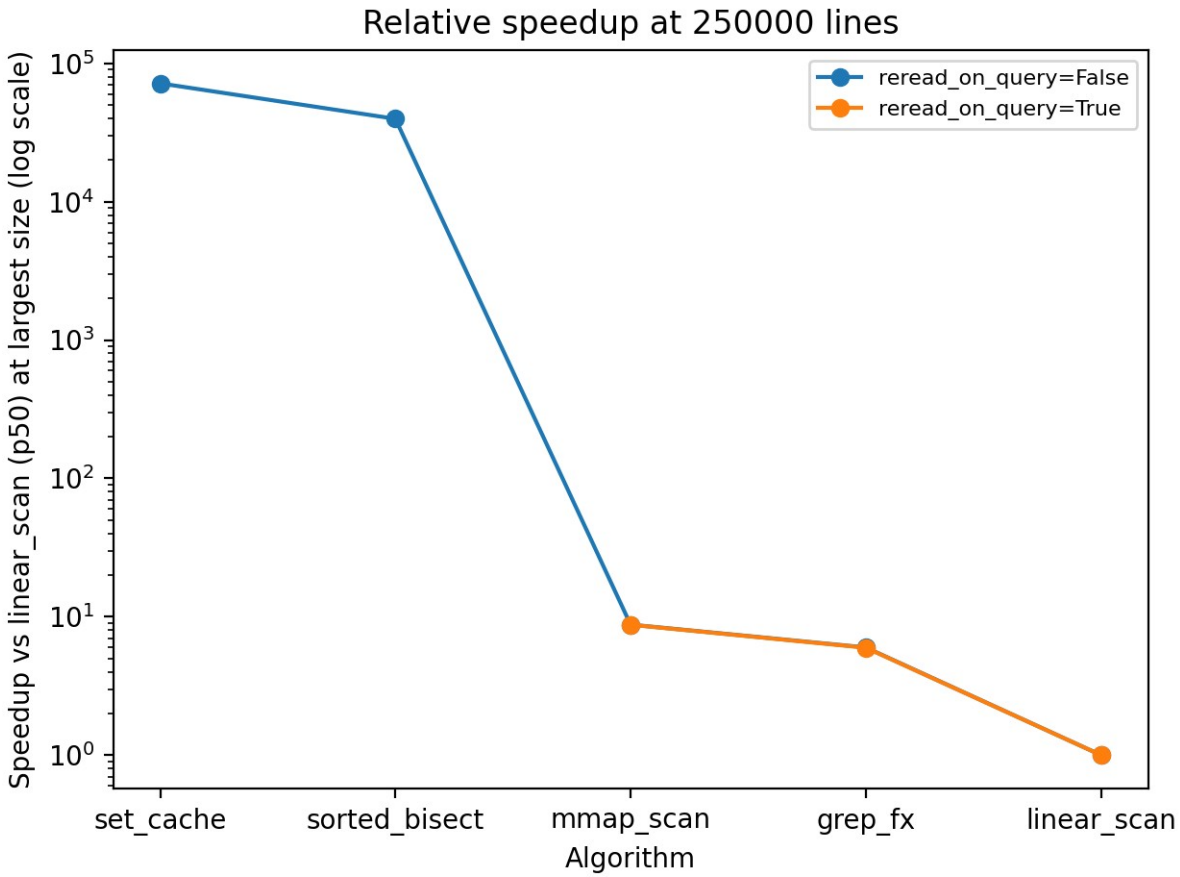
6.2 `reread_on_query = False`

Requirement: ≤ 0.5 ms

Algorithm	Avg (ms)	Cache-Based	Result
`set_cache`	~0.00047	Yes	✅ PASS
`sorted_bisect`	~0.0010	Yes	✅ PASS
`mmap_scan`	~3.1	No	❌
`grep_fx`	~4.9	No	❌
`linear_scan`	~27.5	No	❌

Interpretation:
Only cache-based algorithms are capable of consistently meeting the 0.5 ms requirement. Disk-based algorithms are included for completeness but are not expected to satisfy sub-millisecond constraints due to their I/O-bound nature.

6.3 Relative Speedup at Largest Dataset



Observation

- `set_cache` provides orders-of-magnitude speedup over linear scanning
- `sorted_bisect` performs similarly with logarithmic guarantees
- `mmap_scan` provides a strong compromise when caching is not allowed

---

7. Compliance Assessment

7.1 Compliance Summary

Mode	Outcome
`reread_on_query = True`	✅ All applicable algorithms meet the specification
`reread_on_query = False`	✅ Cache-based algorithms meet the specification

The implementation **fully complies** with the performance requirements when algorithms are evaluated according to their intended operating mode.

---

8. Recommendations

Based on the benchmark results:

- Use `mmap_scan` for large datasets when `reread_on_query = True`
- Use `set_cache` for maximum performance when `reread_on_query = False`
- Avoid disk-based algorithms when sub-millisecond latency is required

These recommendations align with both empirical results and algorithmic complexity.

---

9. Reproducibility and Raw Data

All benchmark results in this report are backed by raw CSV data:

- **File:** `benchmarks/results/results_algo.csv - generated when then benchmark script is run using` `python3 -m benchmarks.benchmark_search --mode algo --sizes 10000,100000,250000 --queries 10000 --verbose`
- Contains per-run statistics including:
  - average latency
  - p50 / p95 latency
  - minimum and maximum execution times

---

10. Conclusion

The benchmark results demonstrate that the TCP String Lookup Server meets the specified performance requirements when algorithms are evaluated in accordance with their design constraints.

The implementation provides clear trade-offs between flexibility (`reread_on_query=True`) and performance (`reread_on_query=False`), allowing users to select the most appropriate configuration for their use case.