Persistent Key-Value Store

Architectural & API Reference

Pujith Sai Kumar Korlepara
IIT Bombay (ID: 25M0787)
Email: pujith@cse.iitb.ac.in / pujith22.sde@gmail.com
GitHub: pujith22 LinkedIn: in/pujith22
Portfolio: cse.iitb.ac.in/ pujith

GitHub Repository: https://github.com/pujith22/persistent-key-value-store

1 Purpose

A C++17 HTTP service providing a persistent key-value cache backed by PostgreSQL. It exposes a JSON-centric REST API for CRUD and bulk operations while maintaining latency via an inline in-memory write-through cache. Startup fails fast if persistence cannot be reached, enforcing durability guarantees.

2 High-Level Architecture

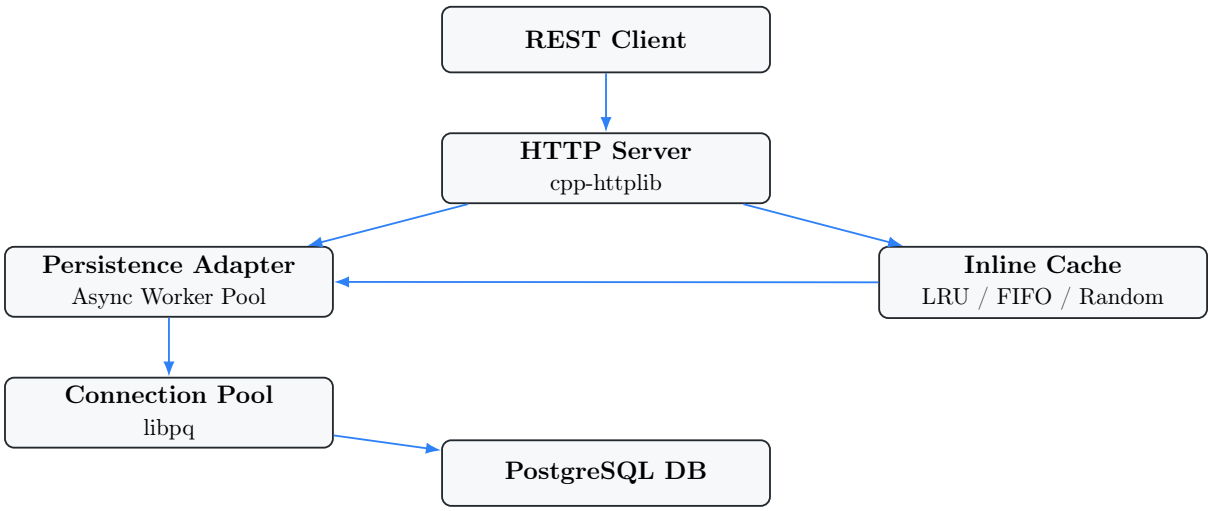

Figure 1: Compact layered architecture: request flow across server, cache, and persistence.

3 Architecture Updates

3.1 Thread Worker Pool & Connection Pooling

To address performance bottlenecks associated with blocking database operations, we have introduced an asynchronous worker pool and connection pooling mechanism in the `PersistenceAdapter`.

- **Connection Pooling**: The adapter maintains a pool of `libpq` connections (default size: 8). This eliminates connection establishment overhead and enables prepared statement

reuse.

- **Async Worker Pool**: A dedicated pool of worker threads handles database operations asynchronously. Tasks are submitted to a queue, picked up by free workers, executed using a pooled connection, and results are returned via `std::future`.

4 Load Test Plan

We have designed a comprehensive load testing strategy with different workloads and testing modes.

4.1 Workloads

- **Workload 1 (Read-Heavy)**: 85% GET, 15% Write. Simulates cache-friendly traffic.
- **Workload 2 (Write-Heavy)**: 50% GET, 50% Write. Simulates high-churn environments.
- **Workload 3 (Balanced)**: Mixed read/write with hot/cold key distribution.
- **Workload 4 (CPU Saturation)**: 100% GET on hot keys (1-100). Stresses CPU/Memory.
- **Workload 5 (Disk Saturation)**: 100% Write (1KB payloads). Stresses Disk I/O.

4.2 Testing Modes

- **Closed-Loop**: Fixed concurrency (threads). Measures throughput/latency.
- **Open-Loop**: Fixed arrival rate (Poisson). Measures stability/tail latency.
- **Saturation**: Pushes specific resources (CPU, Disk) to limits.

5 Postman API Collection

An interactive Postman collection is published for quick exploration and manual testing of the API surface. You can import it directly via the shared workspace URL below or by copying the JSON specification.

Shared Collection URL: Postman Workspace Link

The local development server (see `main_server.cpp`) binds to host `localhost` on port `2222`. The collection above targets that base URL.

Minimal Postman Collection JSON

```
{
  "info": {
    "name": "Persistent Key Value Store API",
    "schema": "https://schema.getpostman.com/json/collection/v2
       .1.0/collection.json",
    "description": "CRUD, bulk query, transactional update and
       observability endpoints"
  },
  "item": [
    {"name": "Catalog", "request": {"method": "GET", "url": "http
       ://localhost:2222/"}},
    {"name": "Home", "request": {"method": "GET", "url": "http://
       localhost:2222/home"}},
    {"name": "Get Key (cache/persist)", "request": {"method": "GET"
       , "url": "http://localhost:2222/get_key/42"}},
    {"name": "Bulk Query", "request": {"method": "PATCH", "header":
        [{"key": "Content-Type", "value": "application/json"}], "
       body": {"mode": "raw", "raw": "{\n  \"data\": [1,2,3,4]\n}"}
       , "url": "http://localhost:2222/bulk_query"}},
    {"name": "Insert", "request": {"method": "POST", "url": "http
       ://localhost:2222/insert/42/alpha"}},
    {"name": "Bulk Update Txn", "request": {"method": "POST", "
       header": [{"key": "Content-Type", "value": "application/json
       "}], "body": {"mode": "raw", "raw": "{\n  \"ops\": [\n    {
       \"type\": \"insert\", \"key\": 7, \"value\": \"seven\" },\n
          { \"type\": \"get\", \"key\": 7 },\n    { \"type\": \"
       update\", \"key\": 7, \"value\": \"SEVEN\" }\n  ]\n}"}, "url
       ": "http://localhost:2222/bulk_update"}},
    {"name": "Delete Key", "request": {"method": "DELETE", "url": "
       http://localhost:2222/delete_key/42"}},
    {"name": "Update Key", "request": {"method": "PUT", "url": "
       http://localhost:2222/update_key/42/beta"}},
    {"name": "Health", "request": {"method": "GET", "url": "http://
       localhost:2222/health"}},
    {"name": "Metrics", "request": {"method": "GET", "url": "http
       ://localhost:2222/metrics"}},
    {"name": "Stop (dev only)", "request": {"method": "GET", "url":
        "http://localhost:2222/stop"}}
  ]
}
```

6 Endpoint Catalog

The service exposes the following HTTP endpoints.

Method	Path	Description
GET	/	Machine-readable service catalog
GET	/home	Formatted documentation for available routes
GET	/get_key/:key_id	Return the value for the provided numeric key, caching it if not present in cache
PATCH	/bulk_query	Retrieve multiple keys in one request; missing keys noted in response; always returns success with errors appended
POST	/insert/:key/:value	Insert a key/value pair; conflicts return 409 with existing value; write-through to cache and persistence
POST	/bulk_update	Transactional pipeline for create/get/insert/update; rolls back on failure and returns failure response
DELETE	/delete_key/:key	Remove the provided key from both cache and persistence layer
PUT	/update_key/:key/:value	Update an existing key with a new value in both cache and persistence layer
GET	/health	Report service health and uptime
GET	/metrics	Expose cache metrics including hit/miss counts
GET	/stop	Gracefully stop the server (testing/debug only; not for production)

7 Screenshots

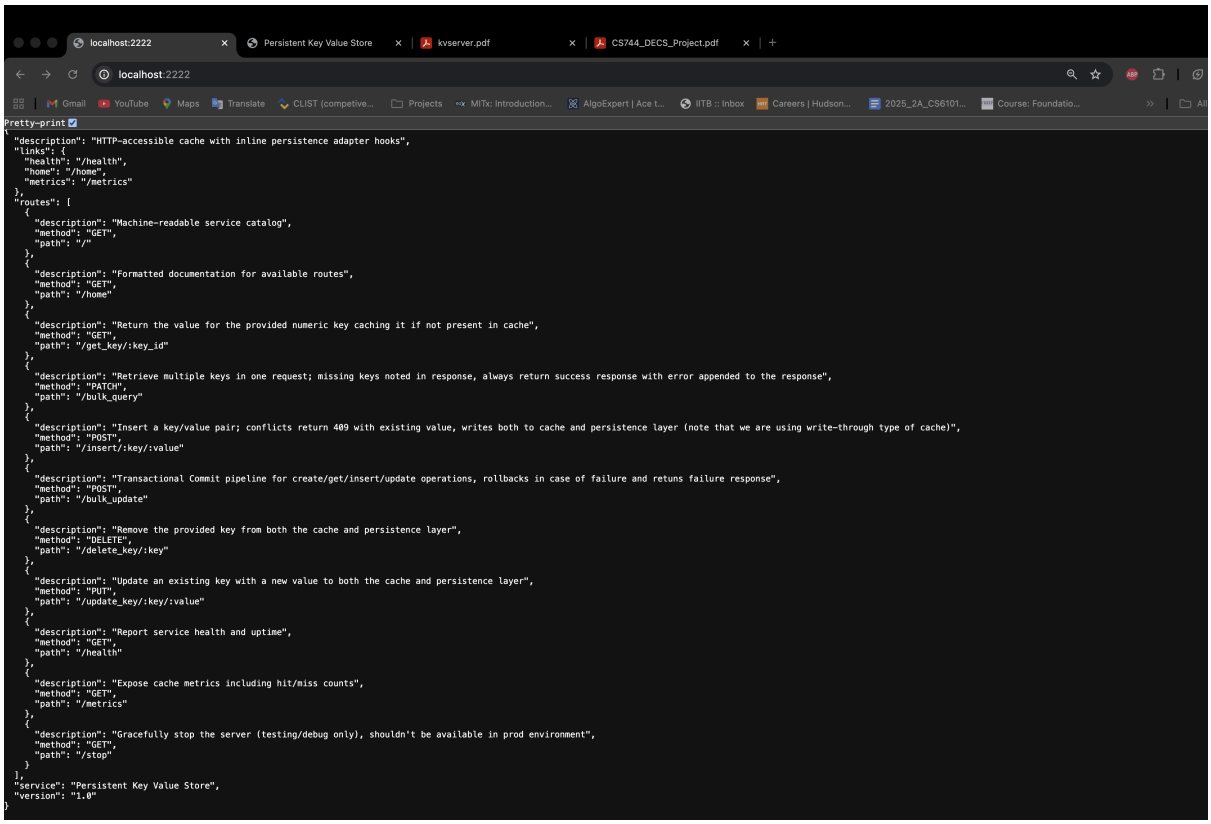

Figure 2: Root endpoint (service catalog)

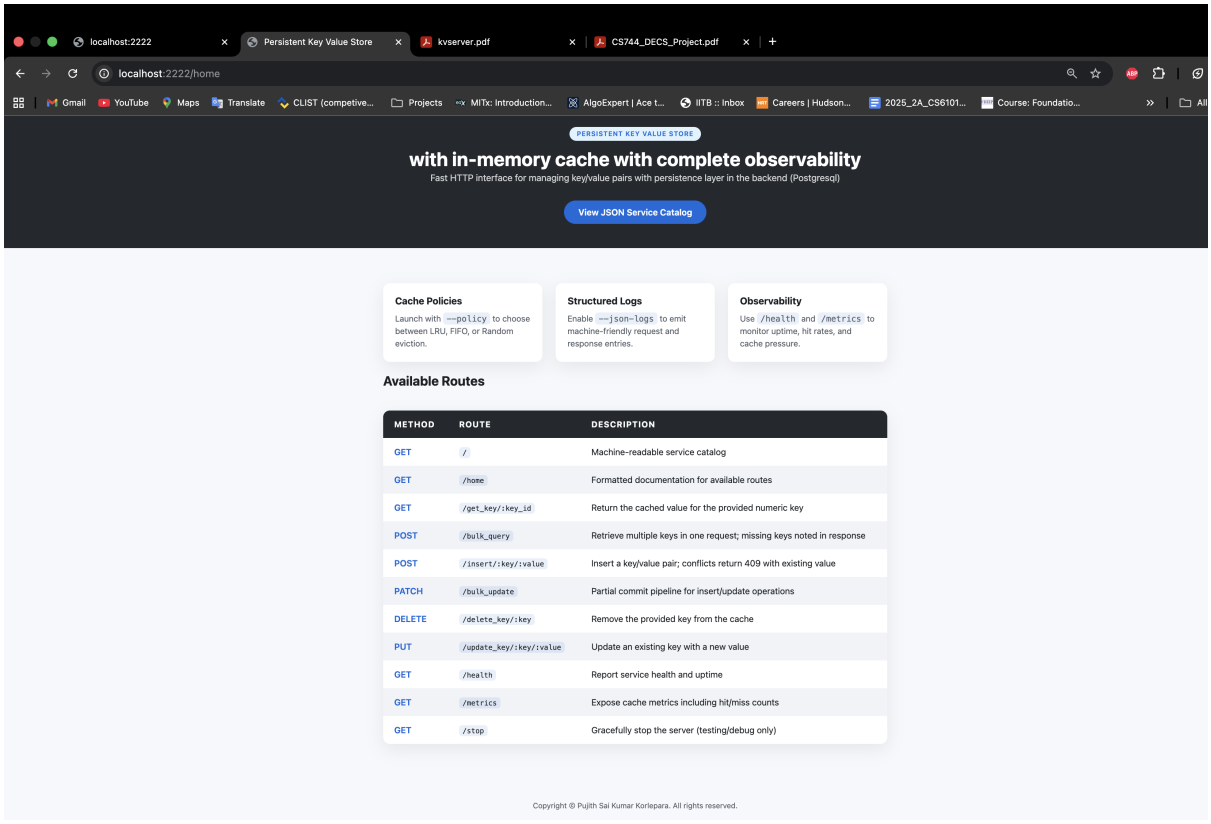

Figure 3: Home HTML endpoint

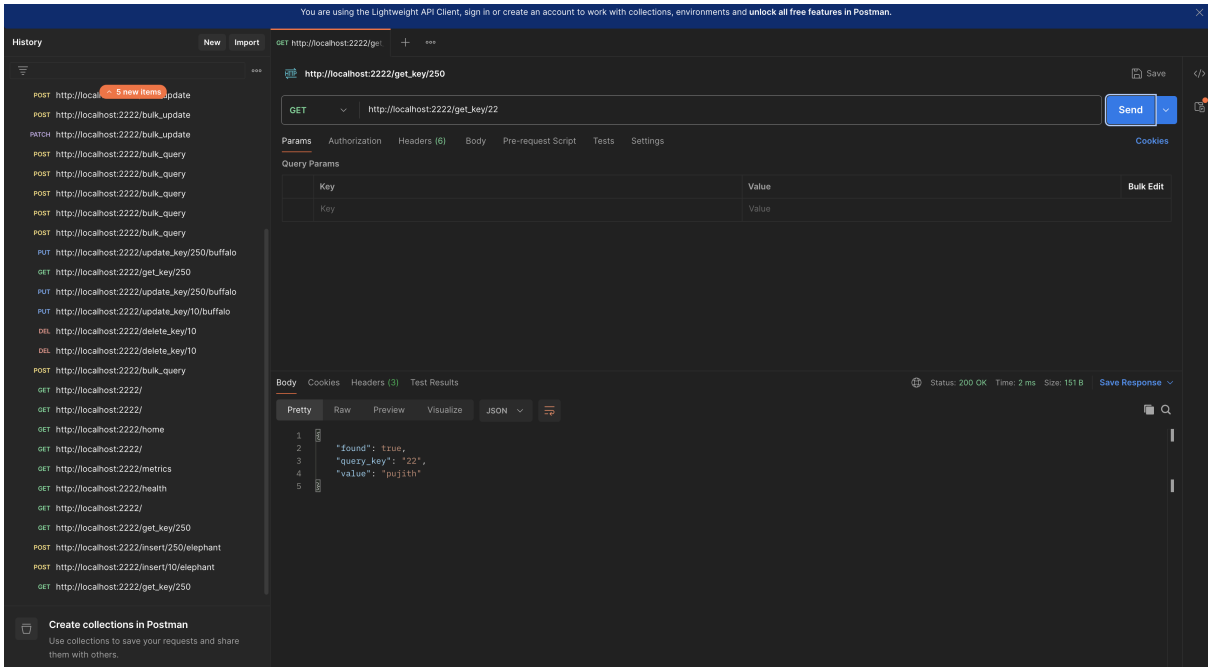

Figure 4: Cache hit on get_key

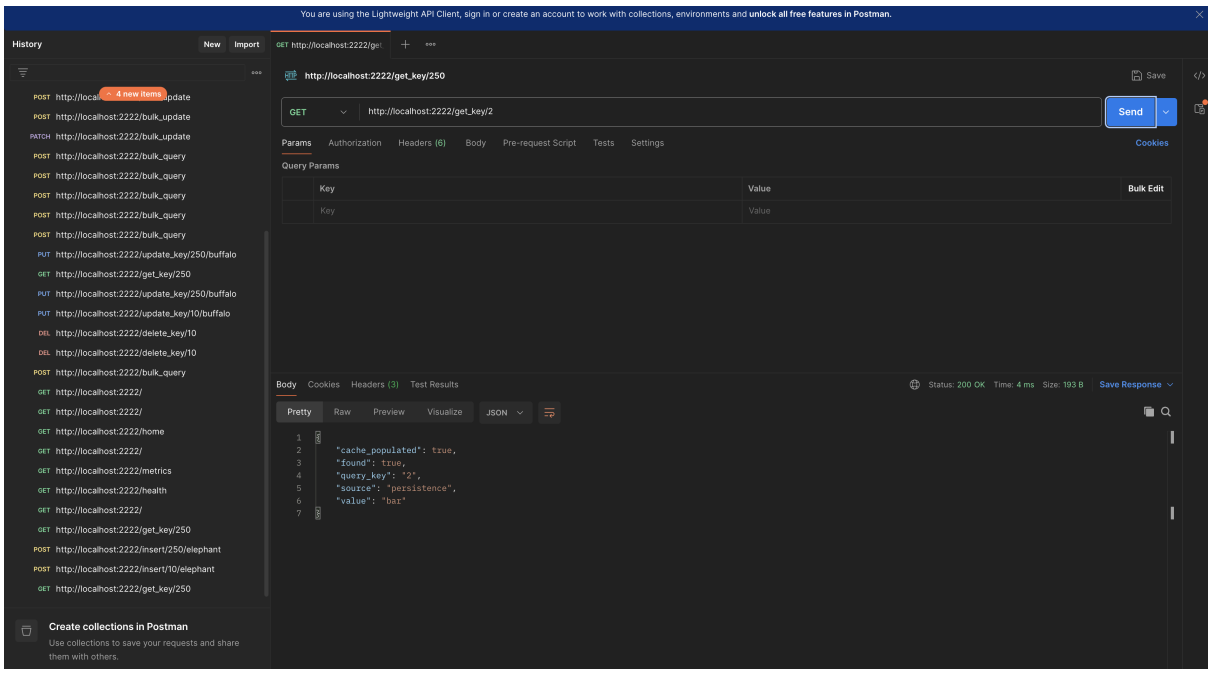

Figure 5: Hydrating uncached key from persistence

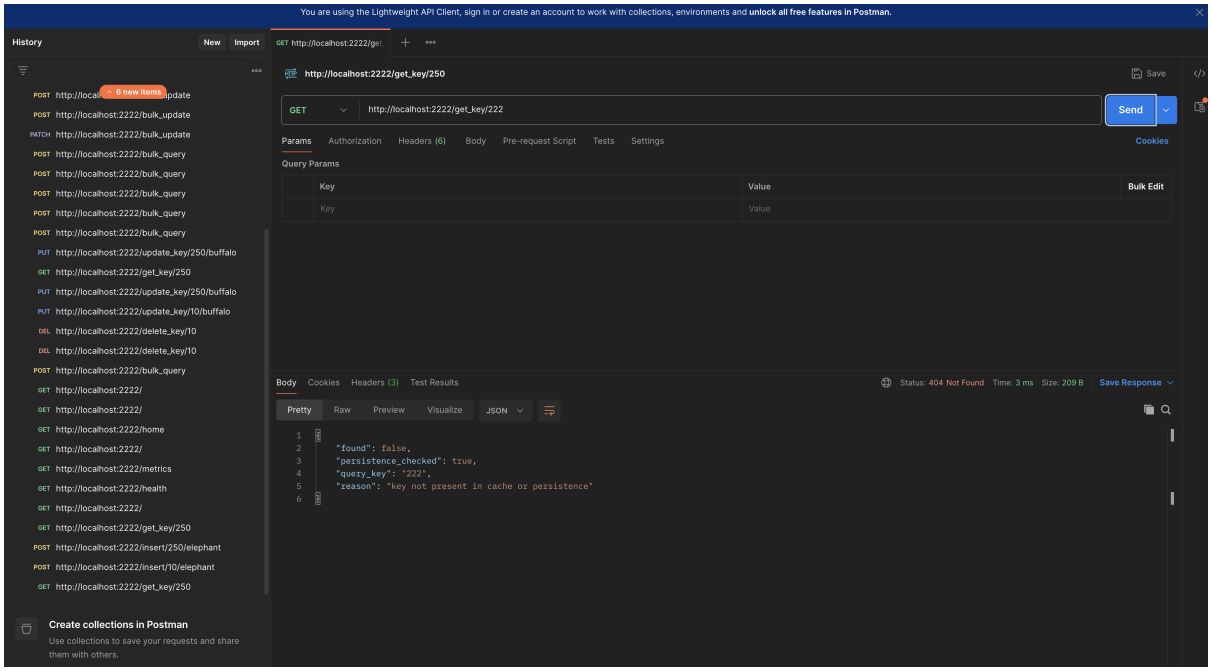

Figure 6: Key not found scenario

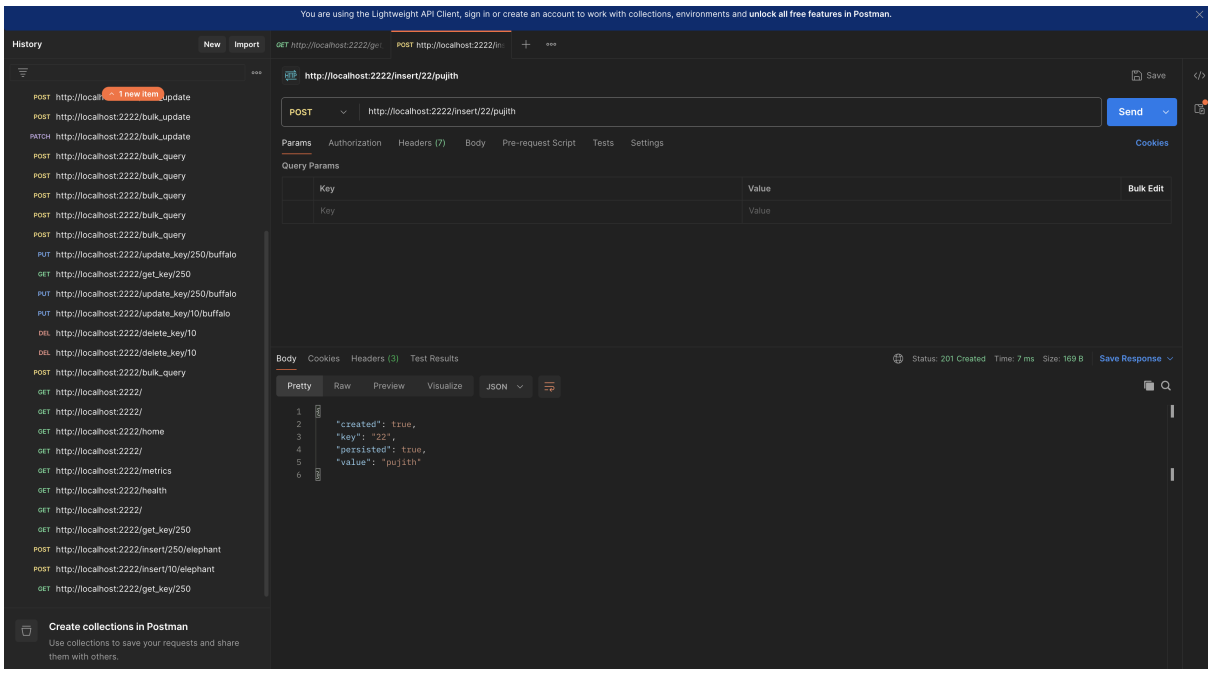

Figure 7: Successful clean insert

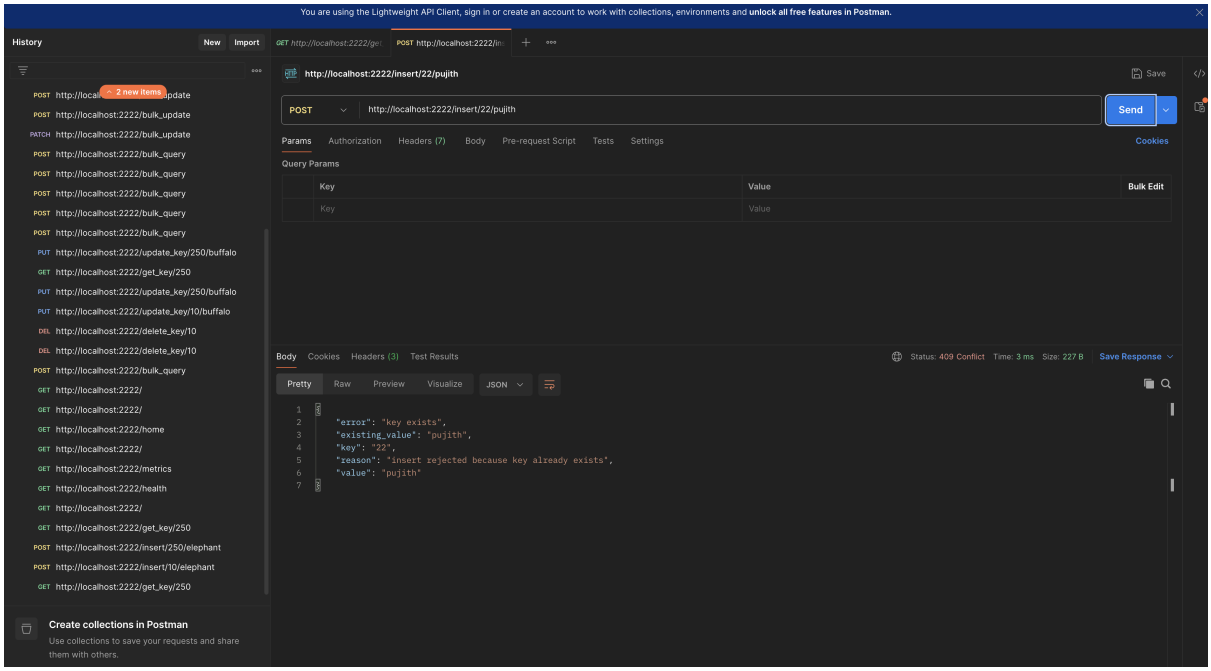

Figure 8: Insert conflict (409)

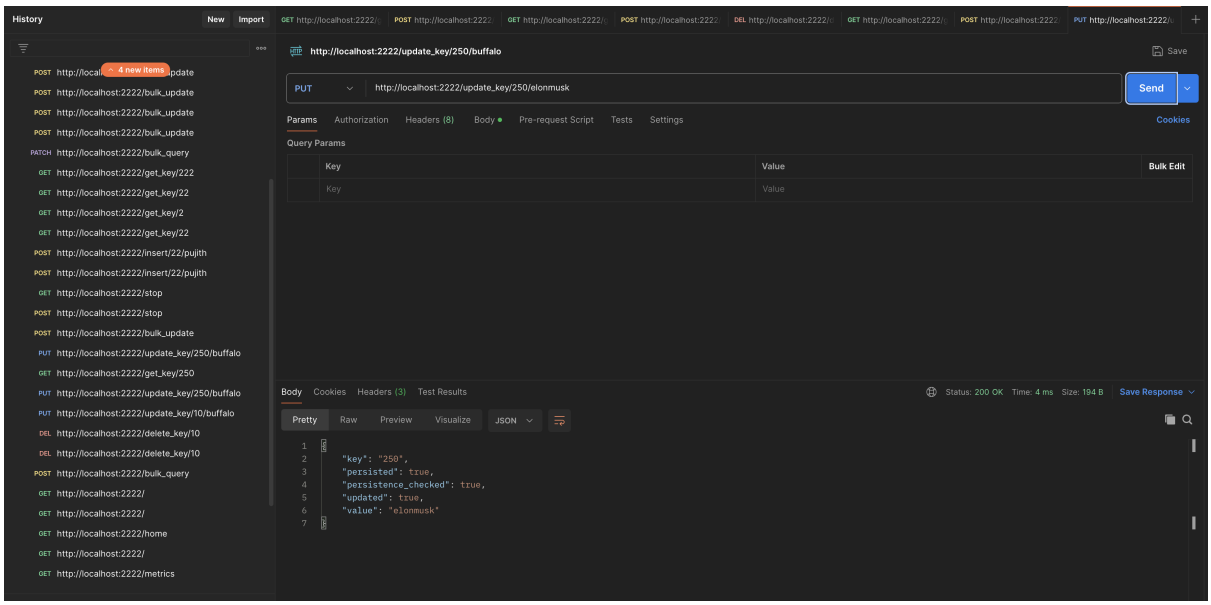

Figure 9: Successful update operation

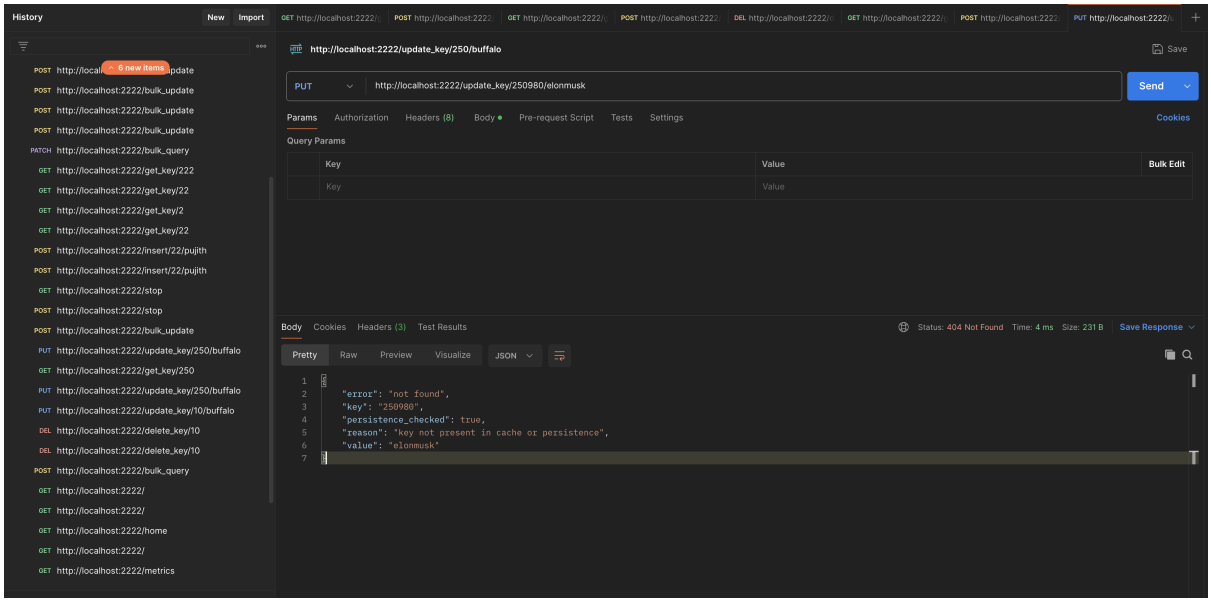

Figure 10: Update when key absent (404)

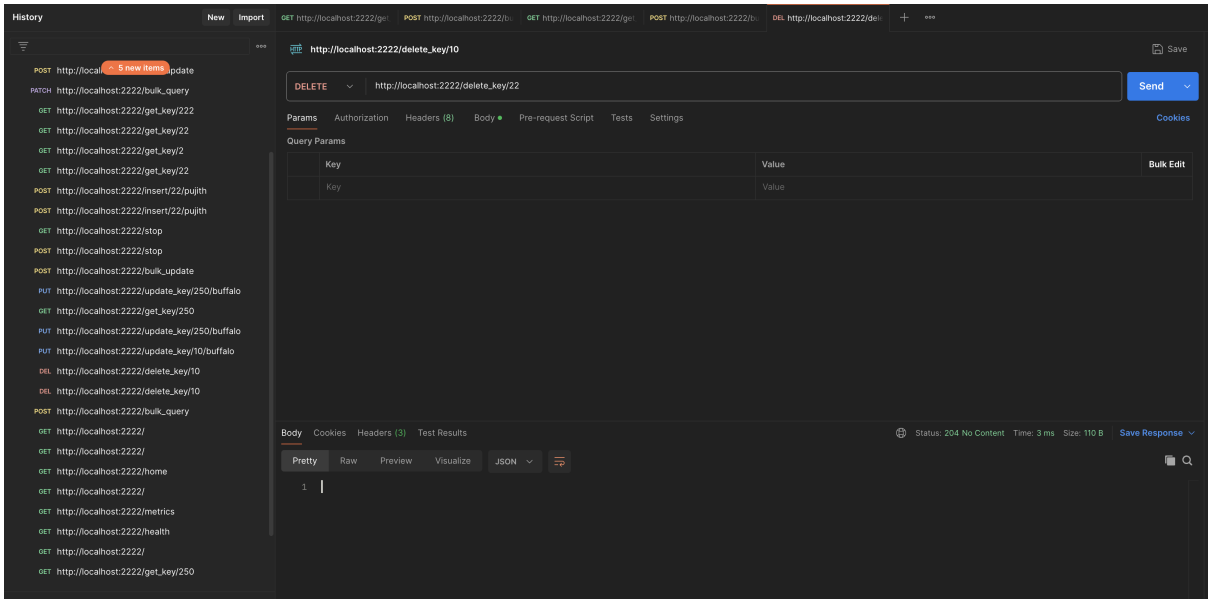

Figure 11: Successful deletion (204)

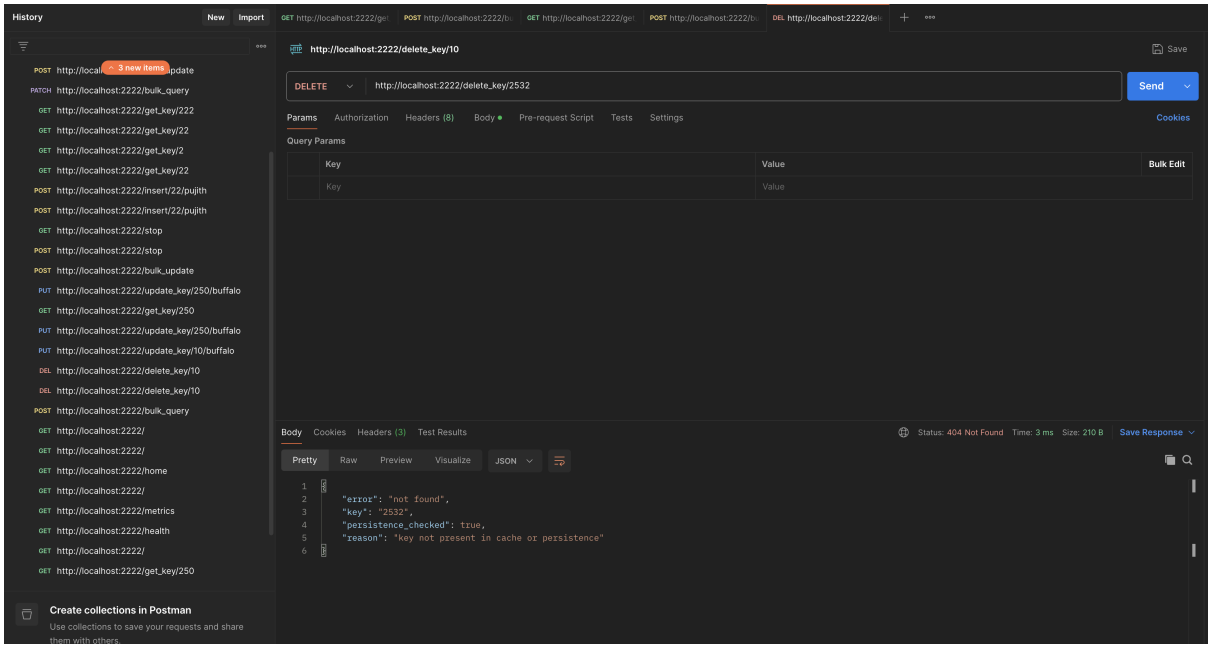

Figure 12: Deletion when key not found

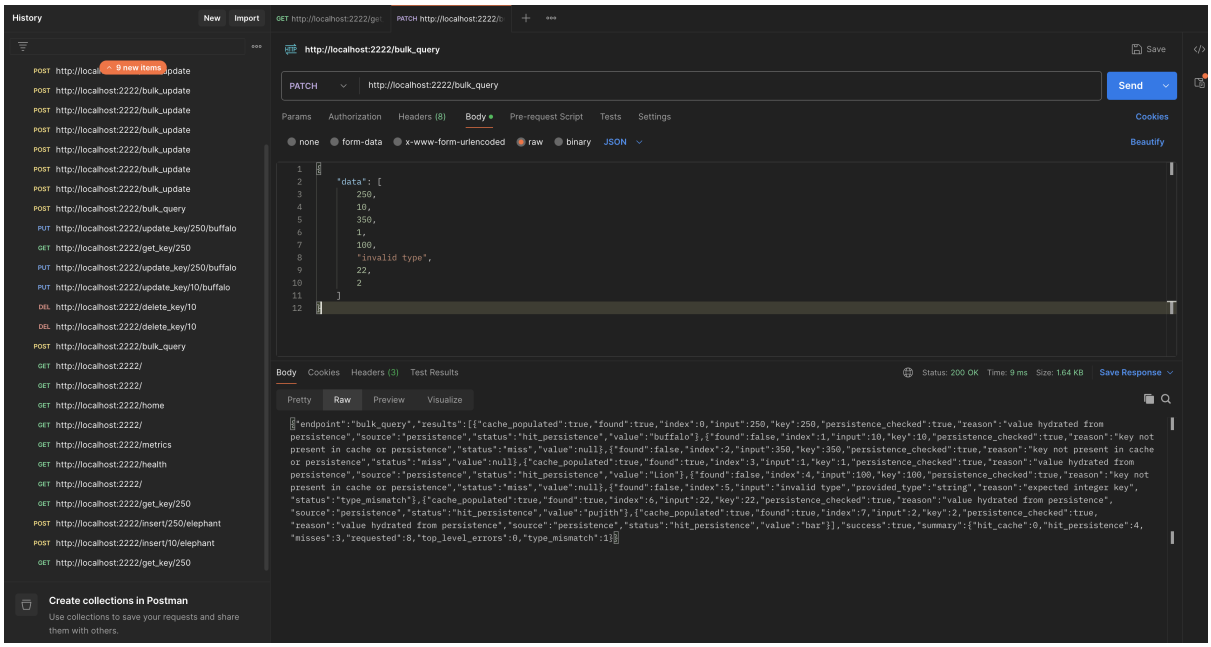

Figure 13: Bulk query results with mixed statuses

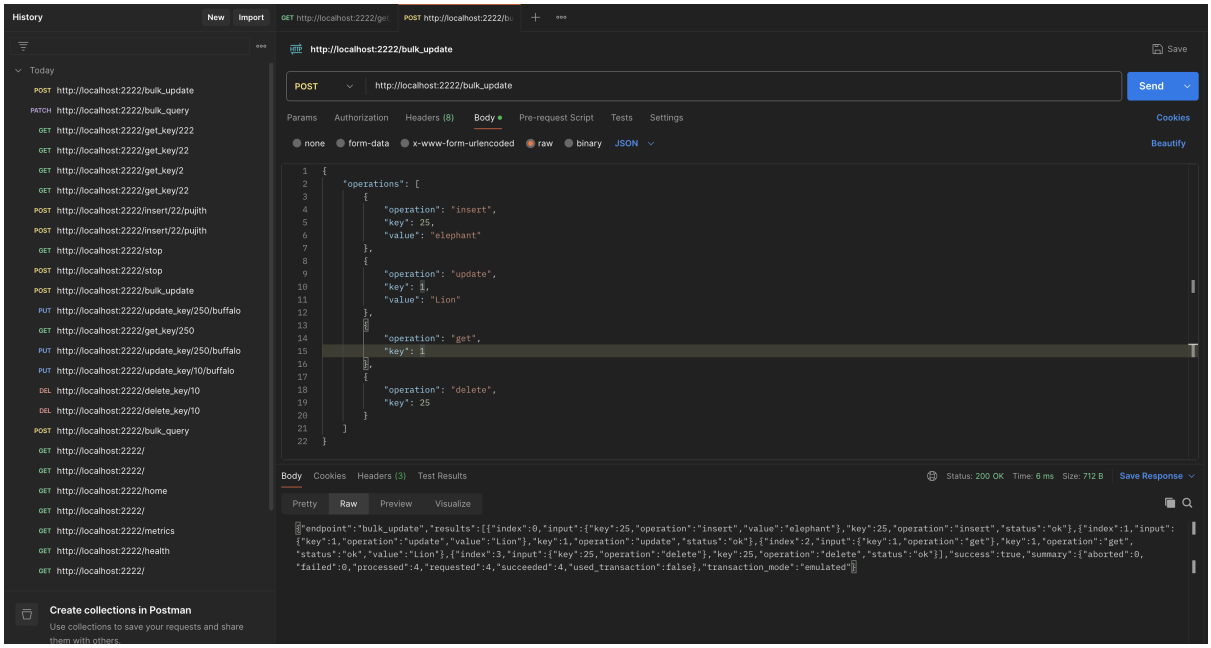

Figure 14: Bulk update transaction success report

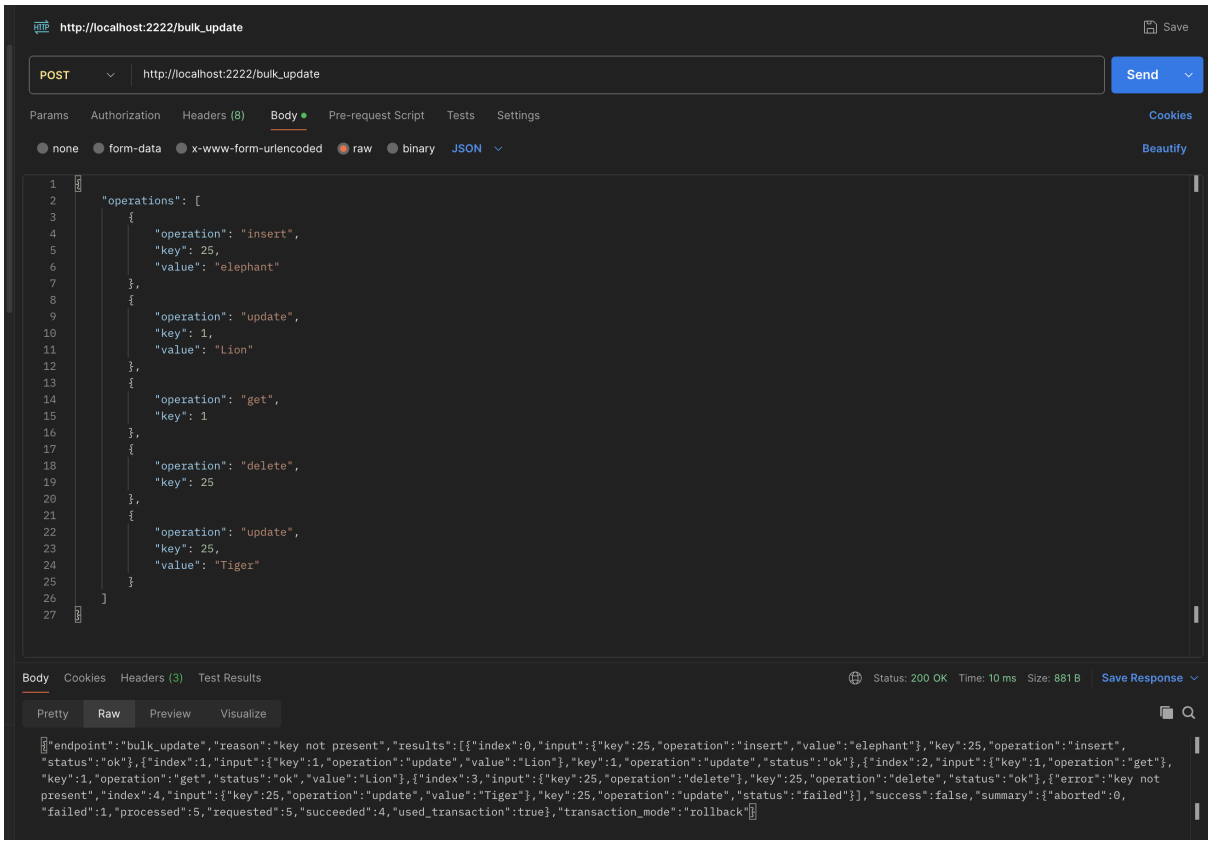

Figure 15: Bulk update transaction failure and rollback

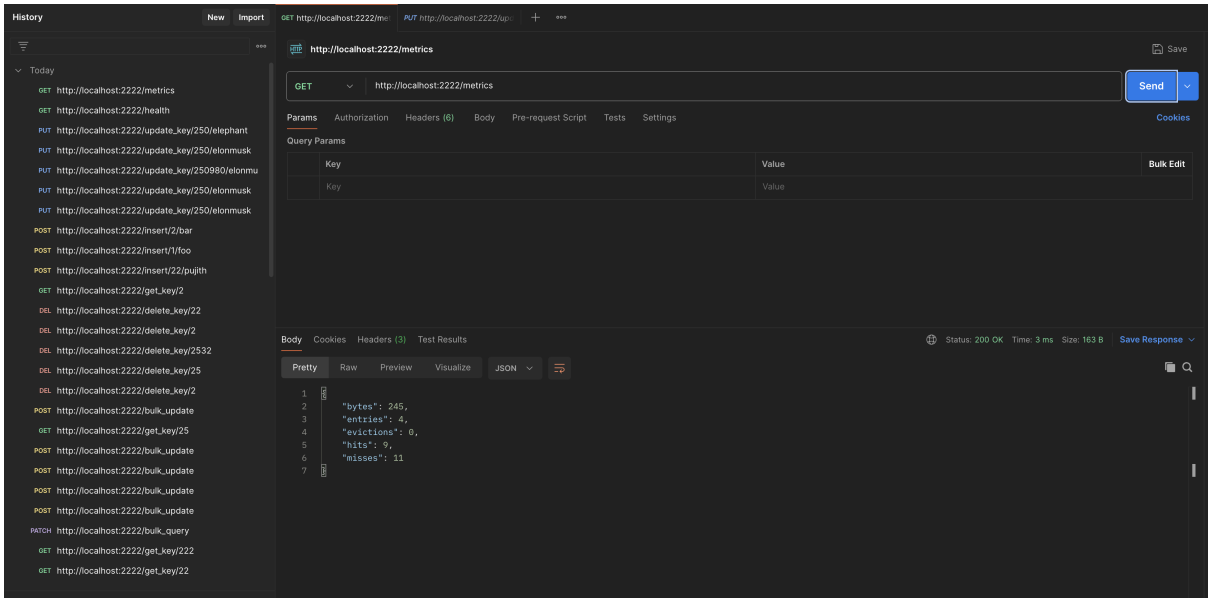

Figure 16: Metrics endpoint showing cache statistics

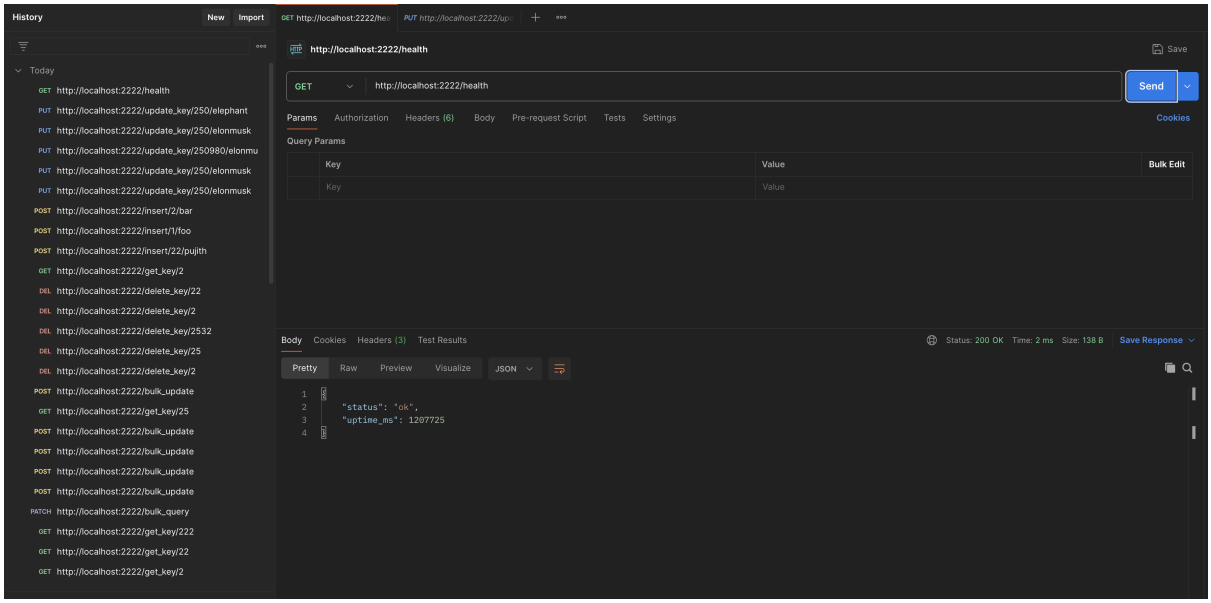

Figure 17: Health endpoint with uptime

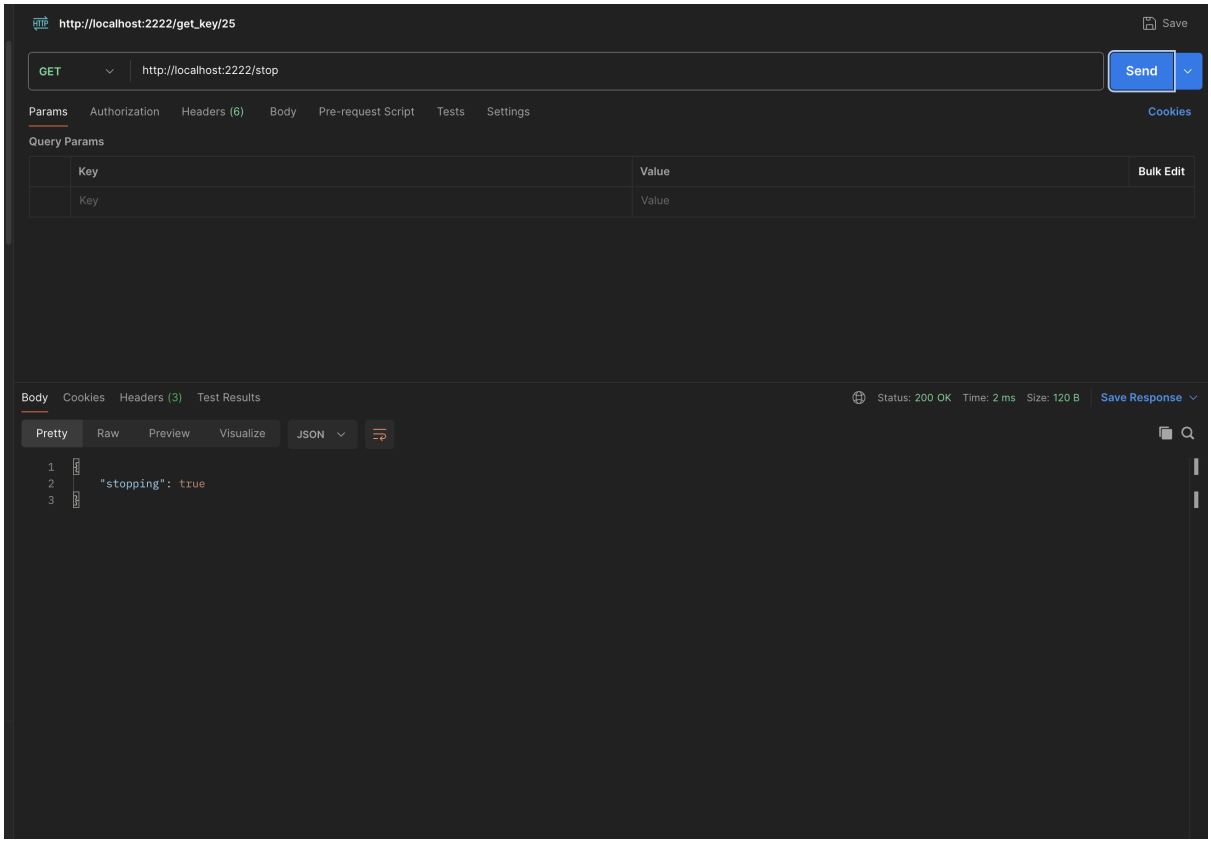

Figure 18: Stop endpoint (dev only)

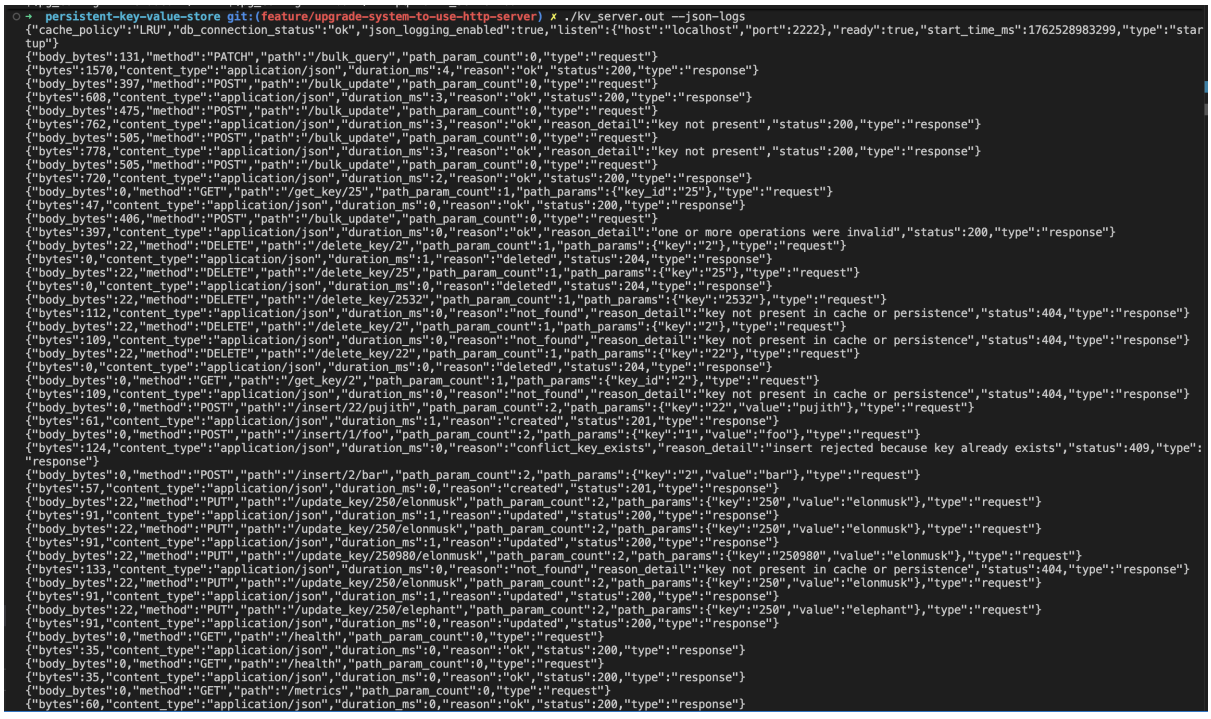

Figure 19: Structured JSON logging output sample

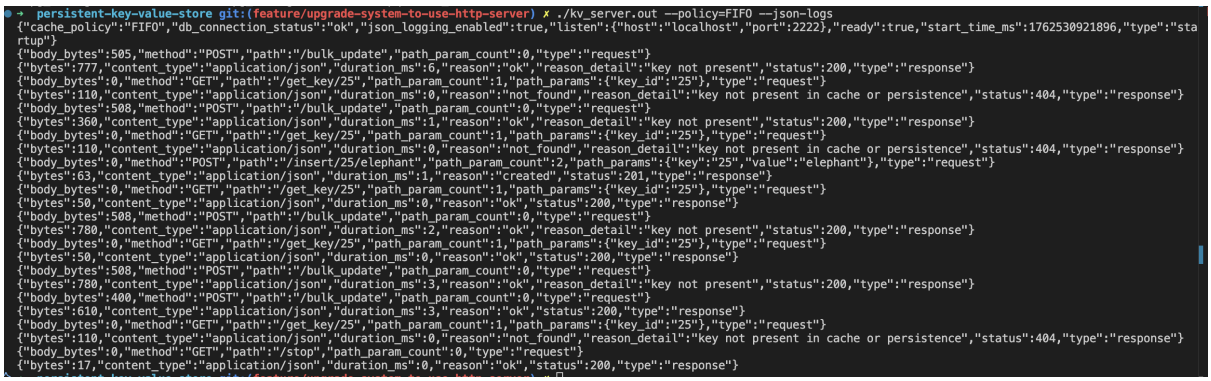

Figure 20: Server startup with custom cache eviction policy.

8 Load Test Results

The following graphs illustrate the system's performance under different workloads and testing modes.

8.1 Workload 1: Read-Heavy (85% GET)

8.1.1 Performance (Throughput & Latency)

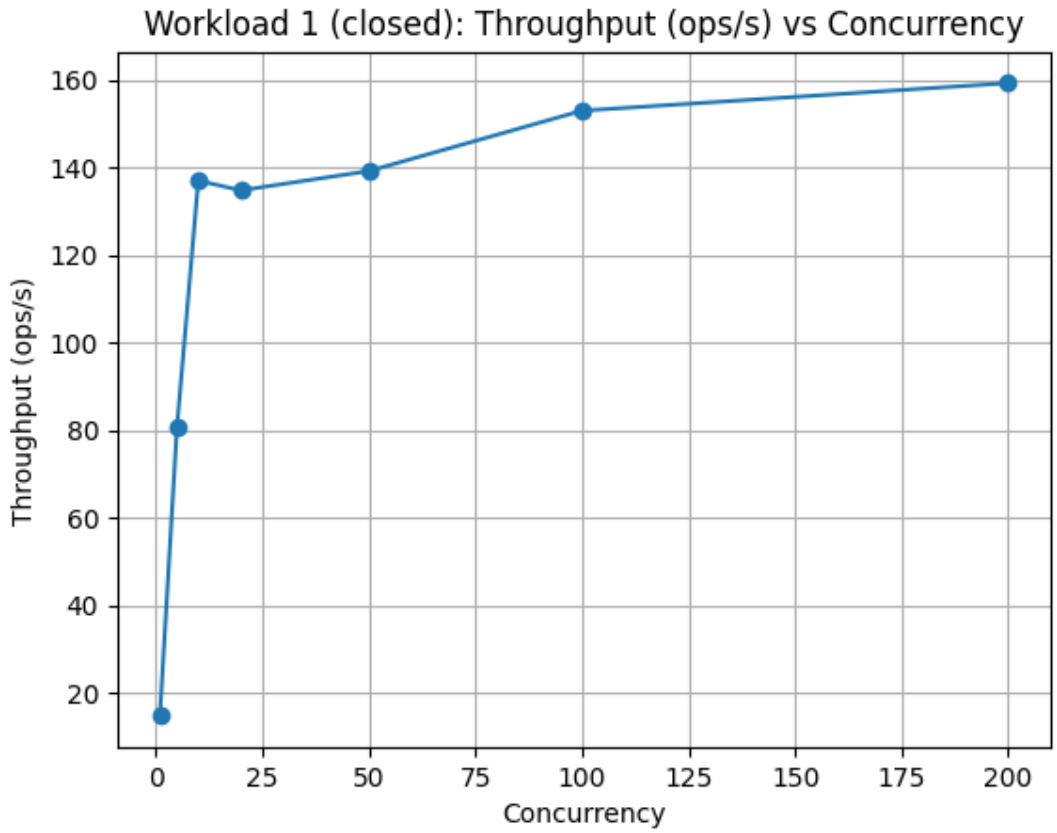

Figure 21: Closed Loop: Throughput vs Concurrency

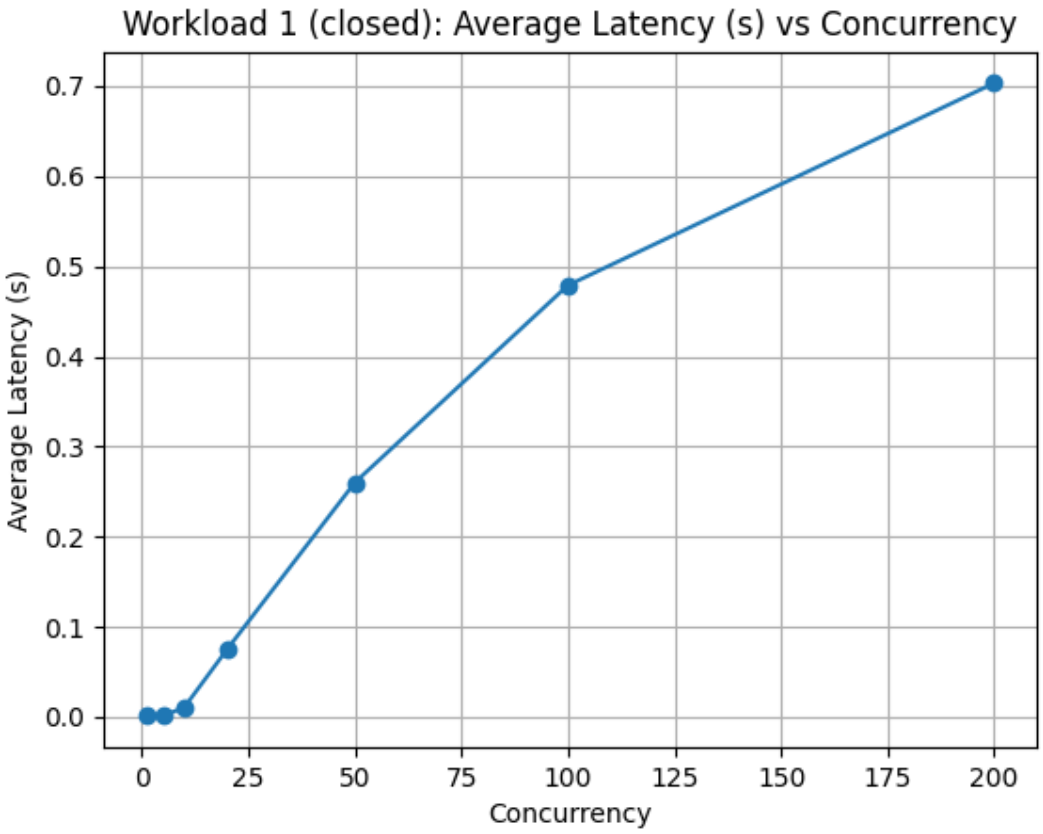

Figure 22: Closed Loop: Avg Latency vs Concurrency

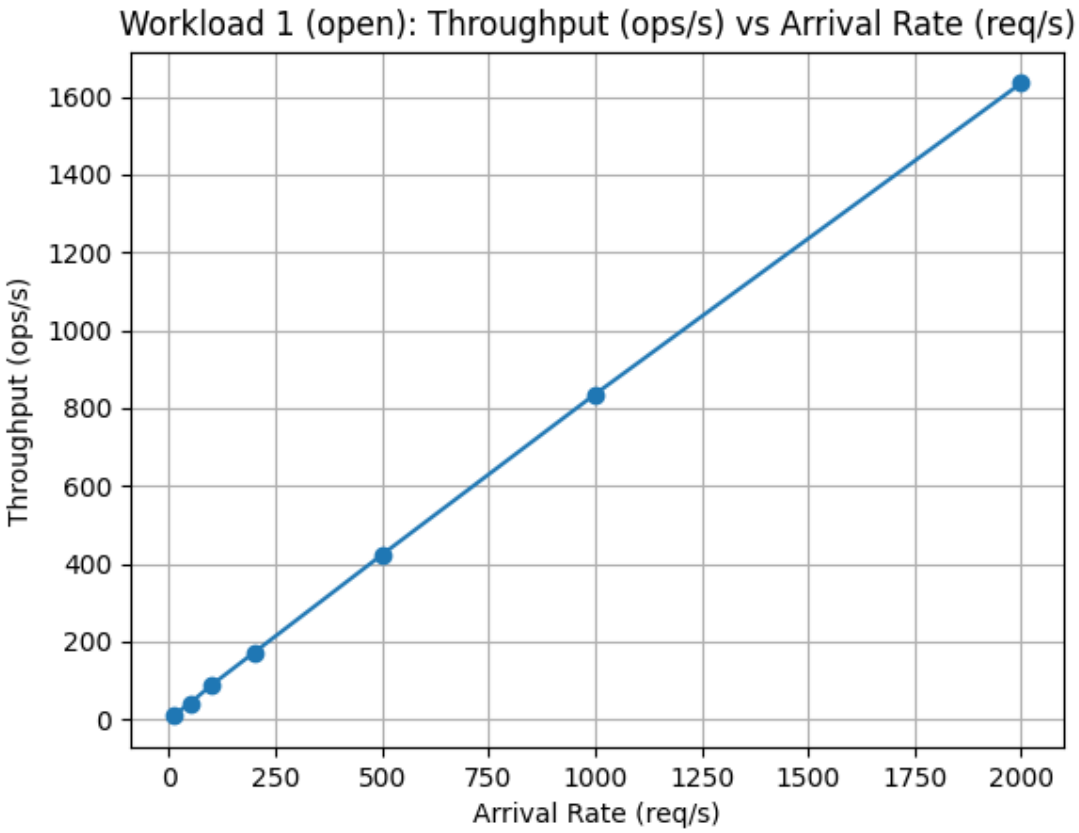

Figure 23: Open Loop: Throughput vs Arrival Rate

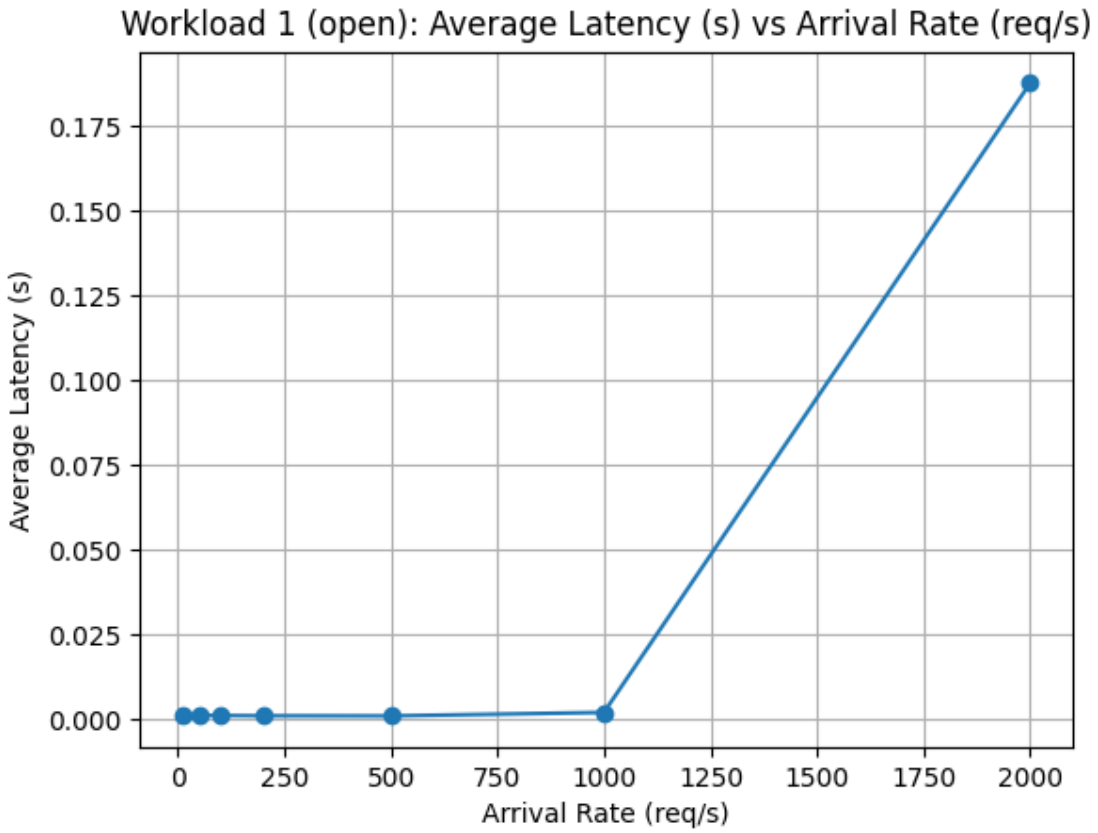

Figure 24: Open Loop: Avg Latency vs Arrival Rate

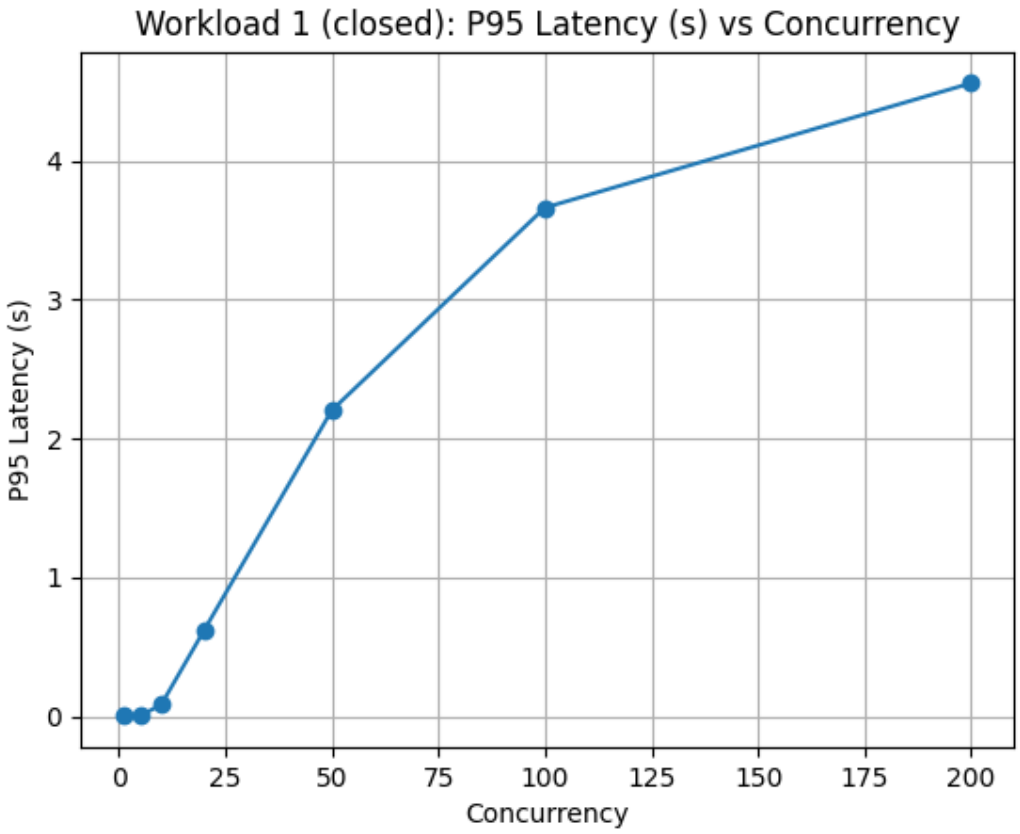

Figure 25: Closed Loop: P95 Latency

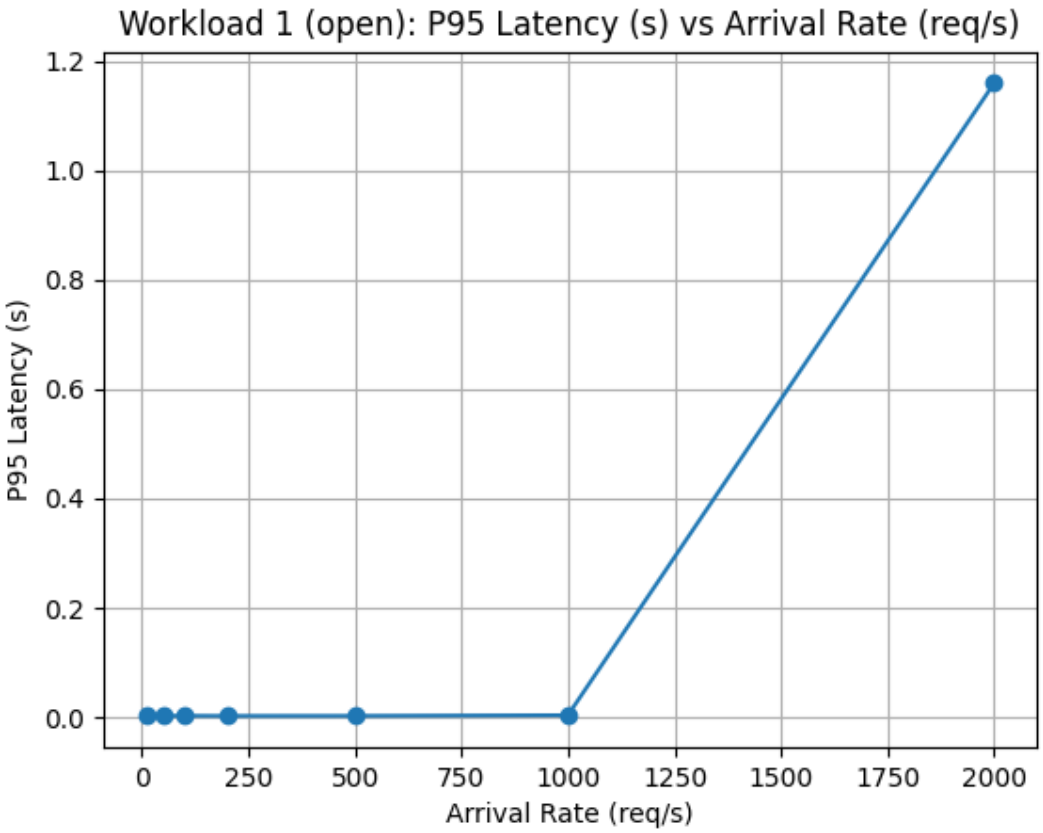

Figure 26: Open Loop: P95 Latency

8.1.2 System Resources (CPU & Disk)

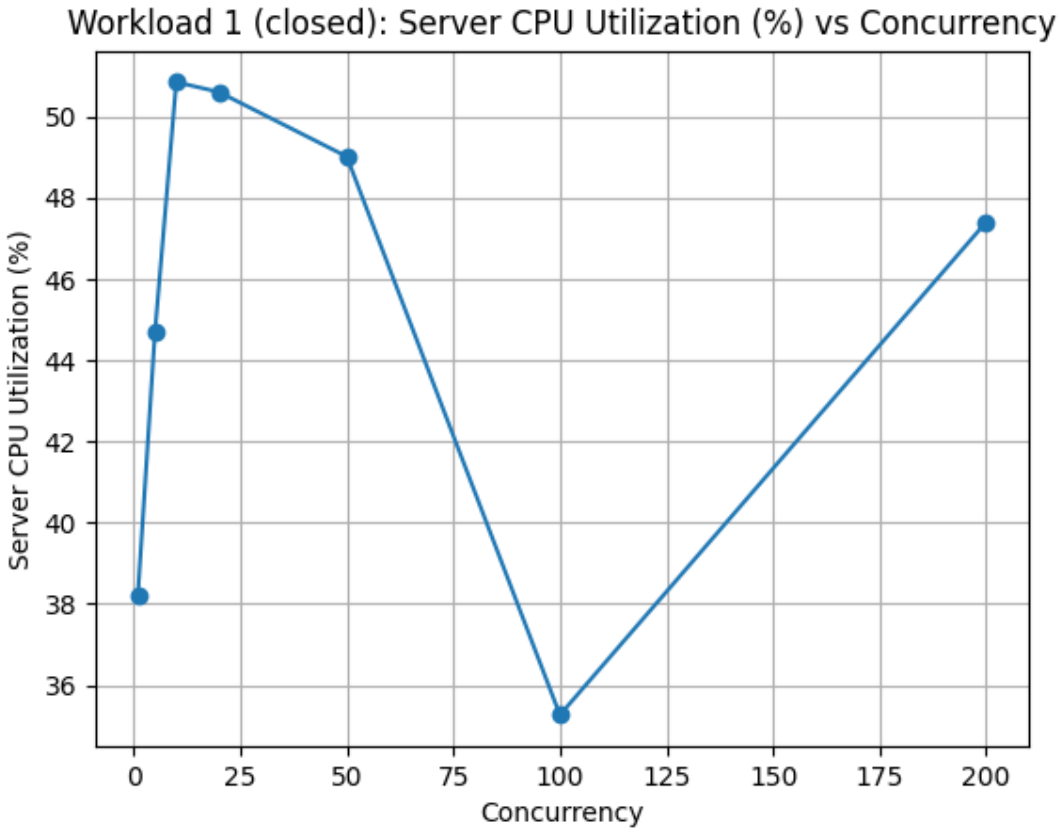

Figure 27: Closed Loop: CPU Utilization

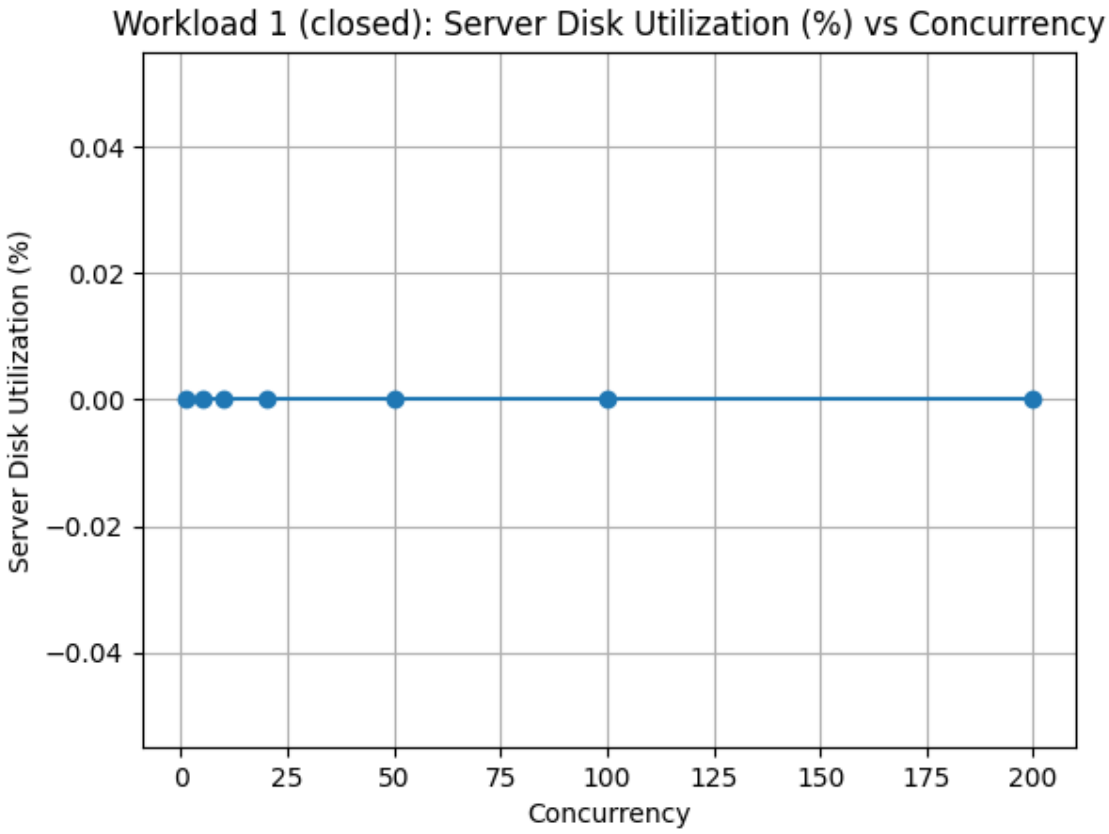

Figure 28: Closed Loop: Disk Utilization

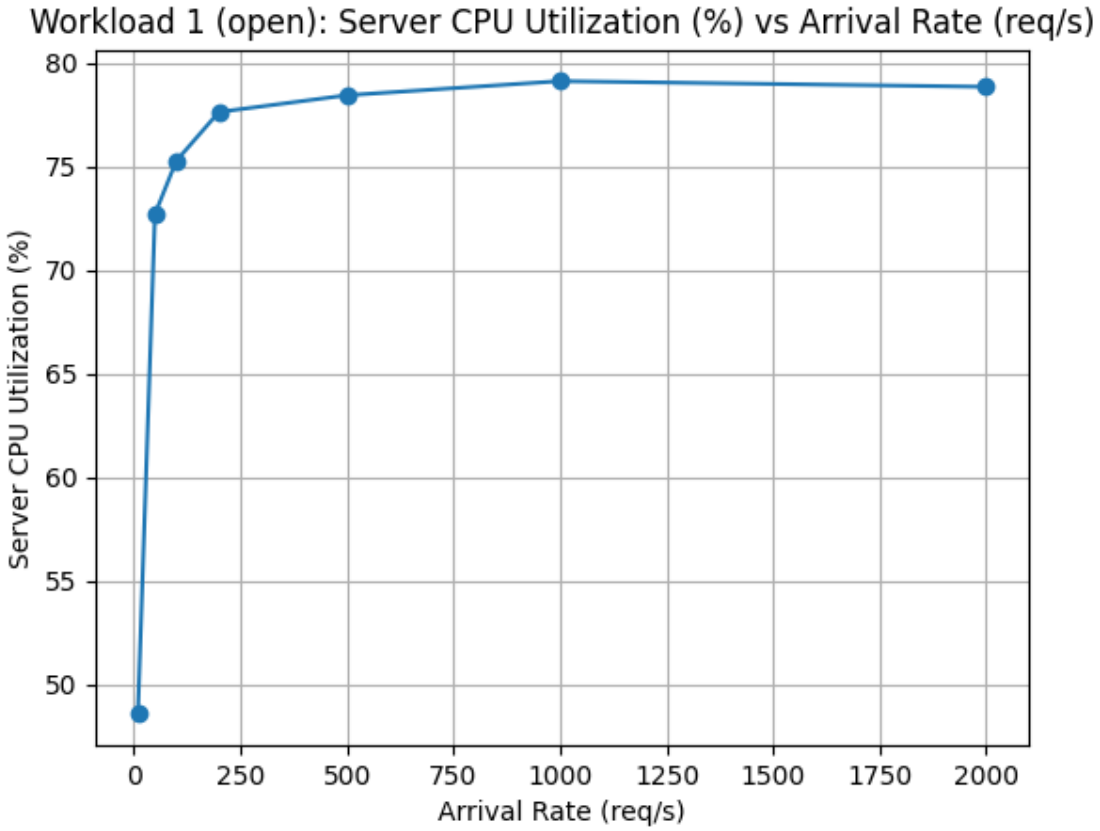

Figure 29: Open Loop: CPU Utilization

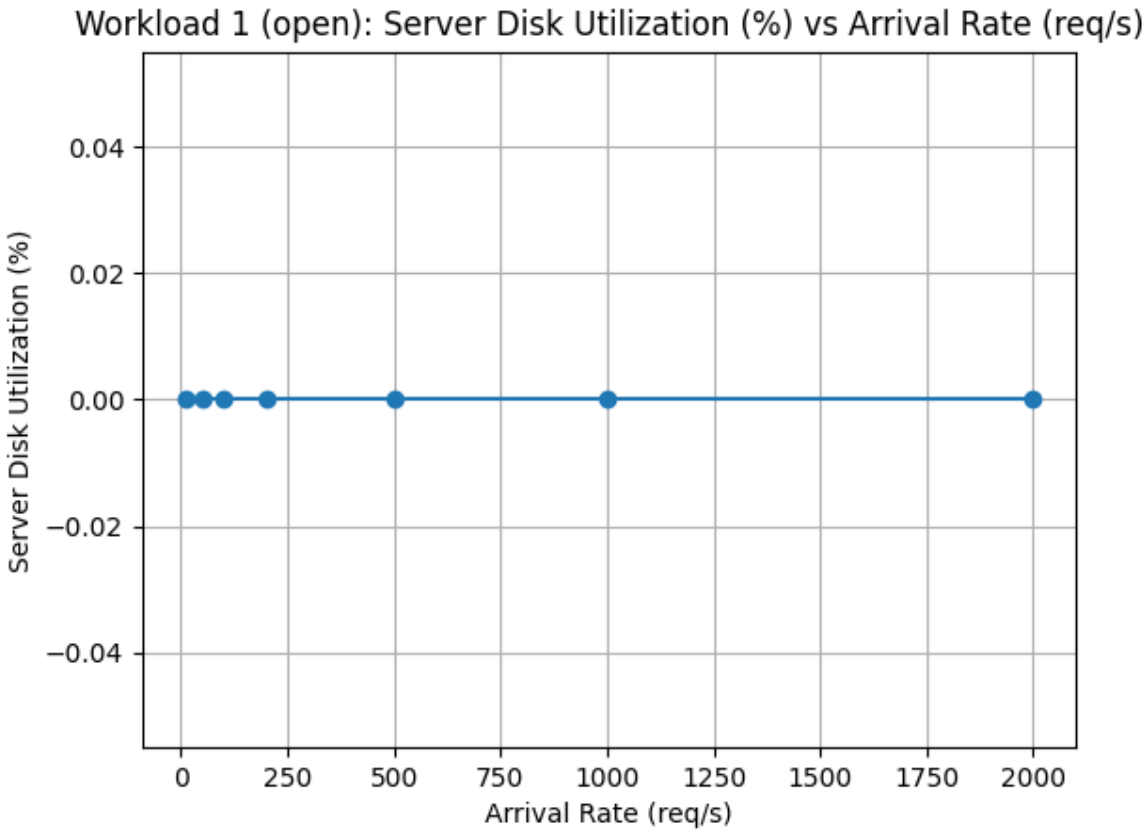

Figure 30: Open Loop: Disk Utilization

8.1.3 Cache & Memory

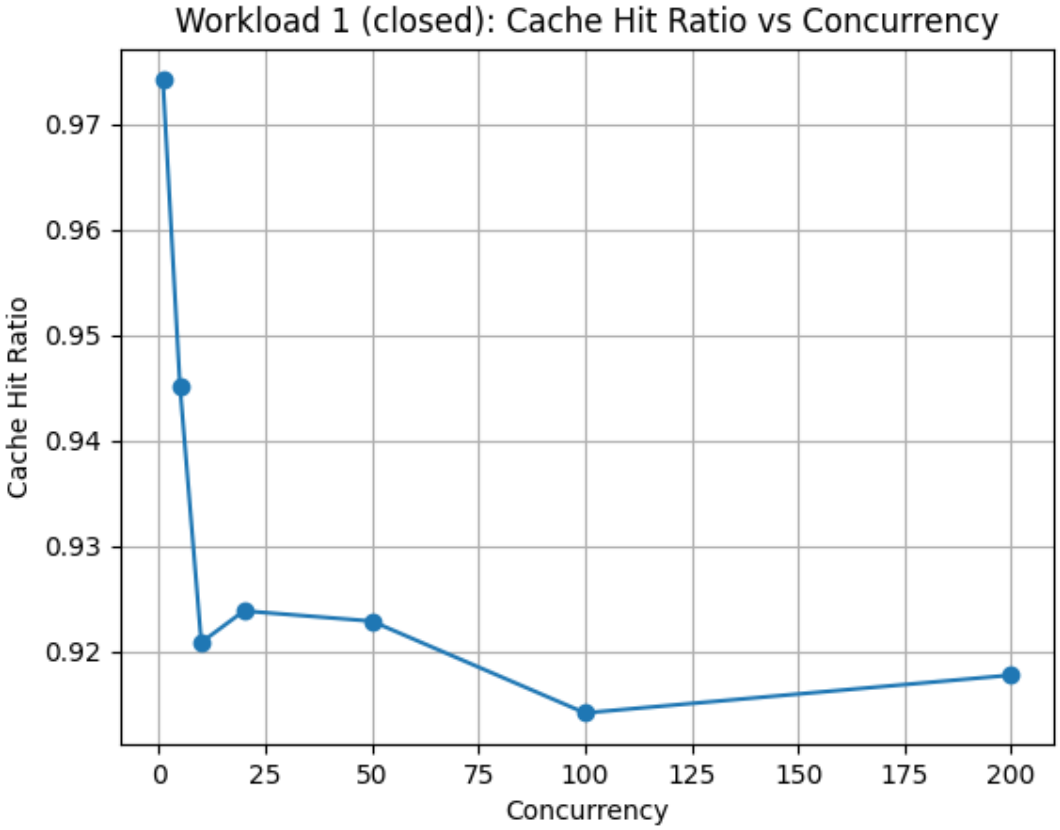

Figure 31: Closed Loop: Cache Hit Ratio

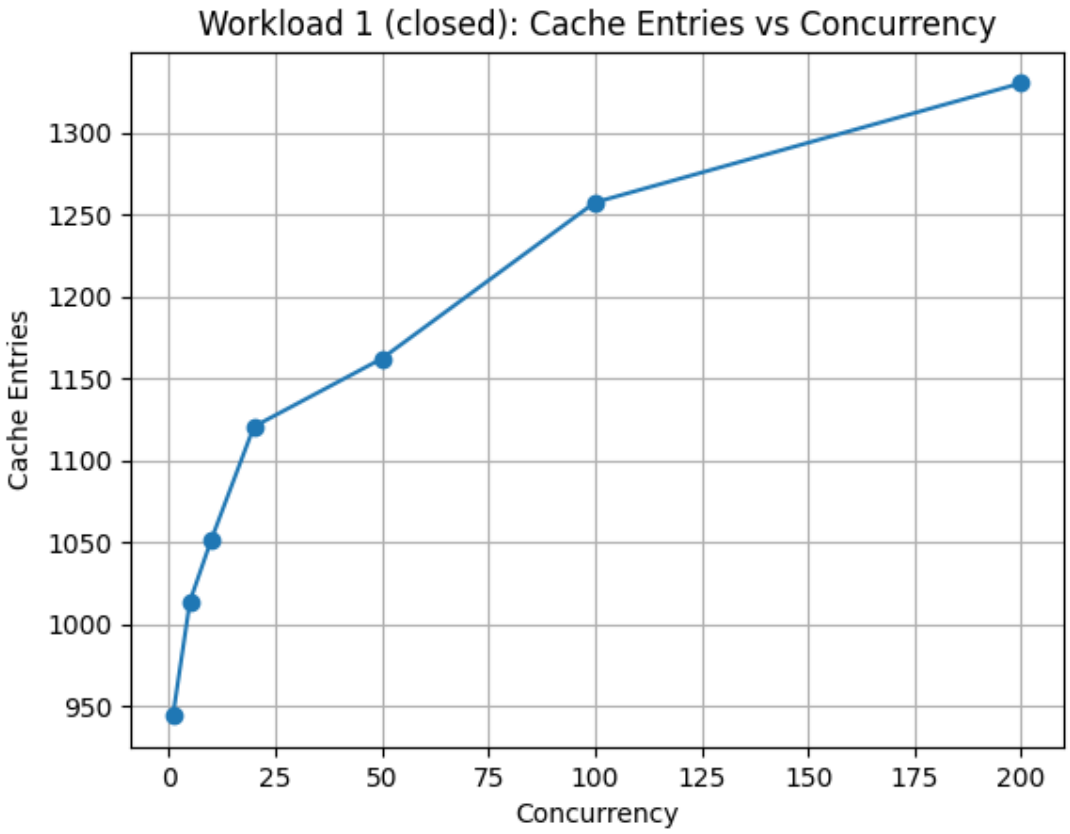

Figure 32: Closed Loop: Cache Entries

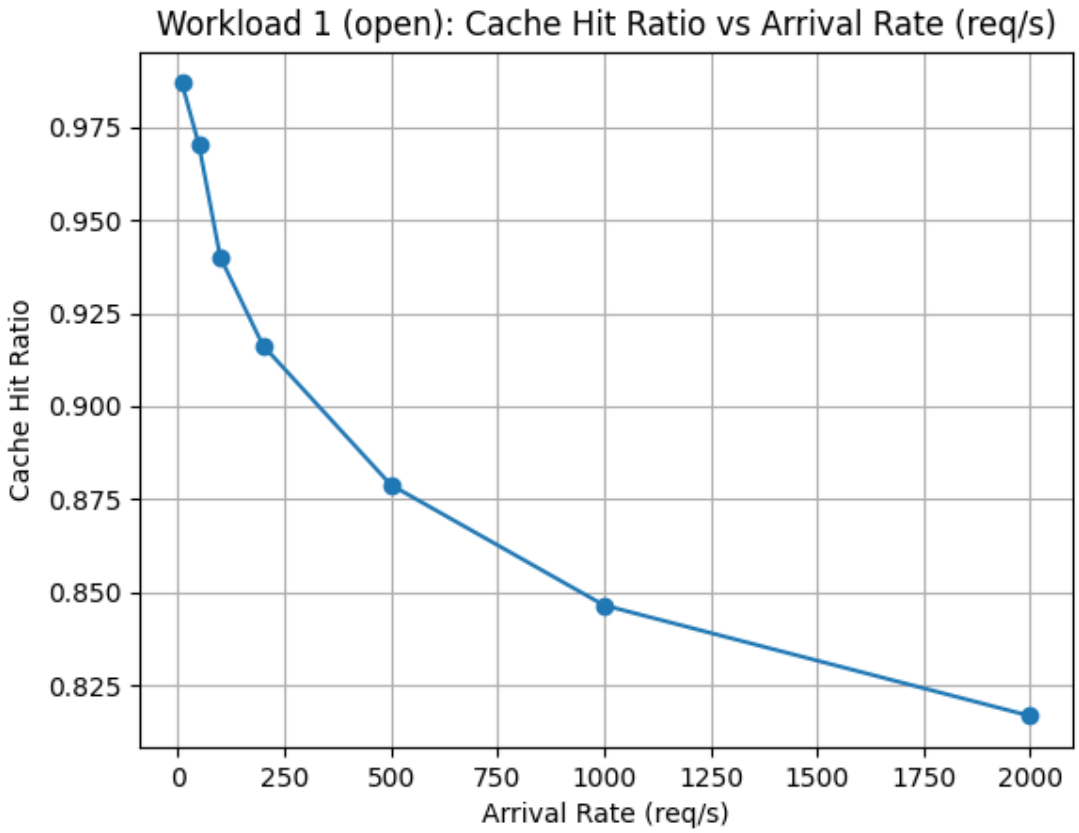

Figure 33: Open Loop: Cache Hit Ratio

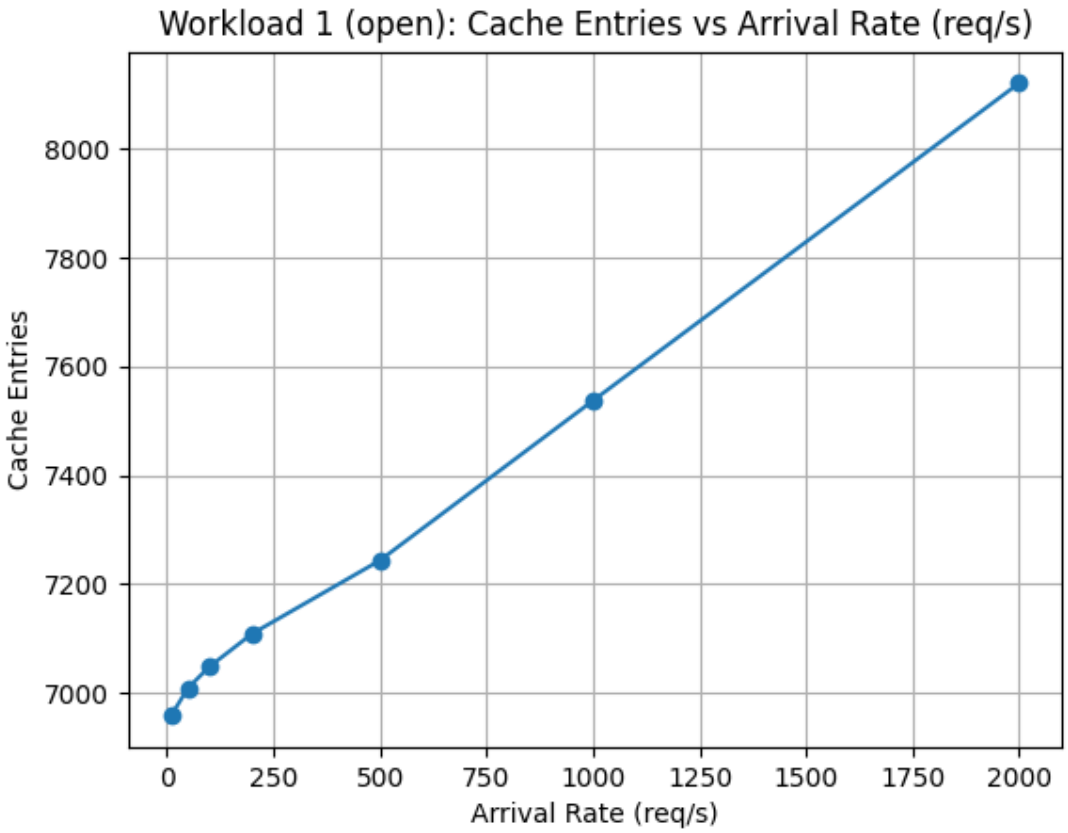

Figure 34: Open Loop: Cache Entries

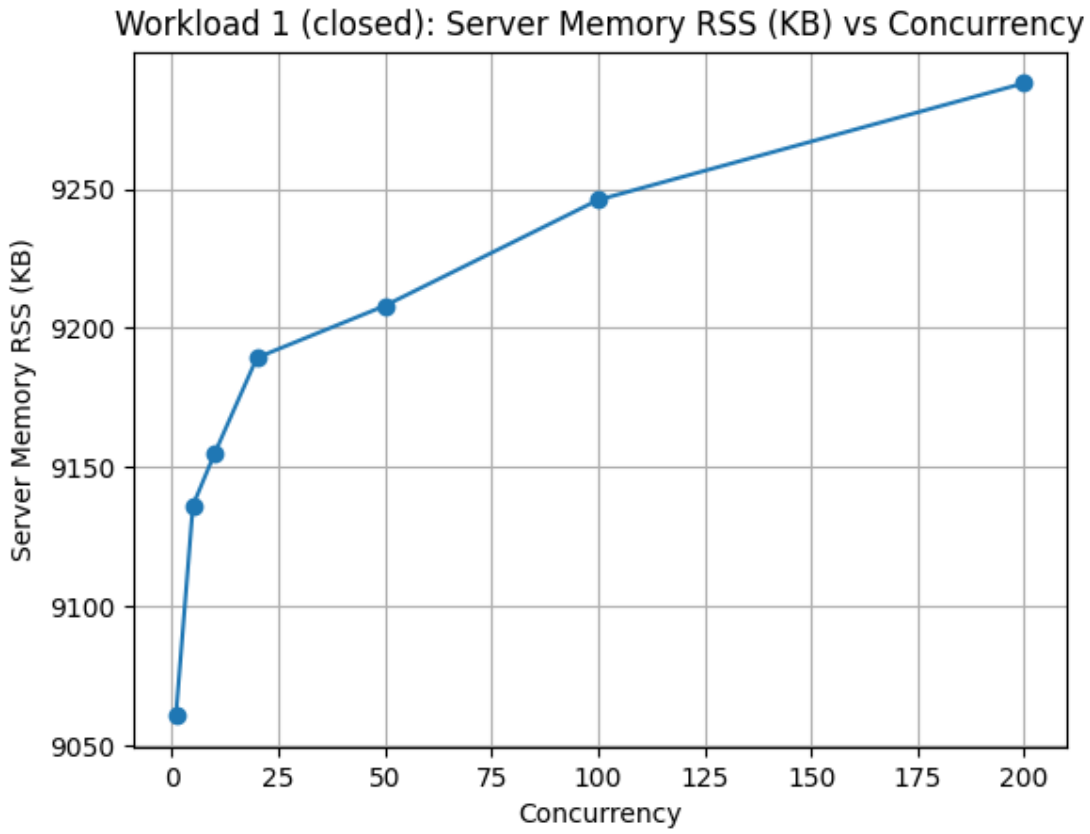

Figure 35: Closed Loop: Memory RSS

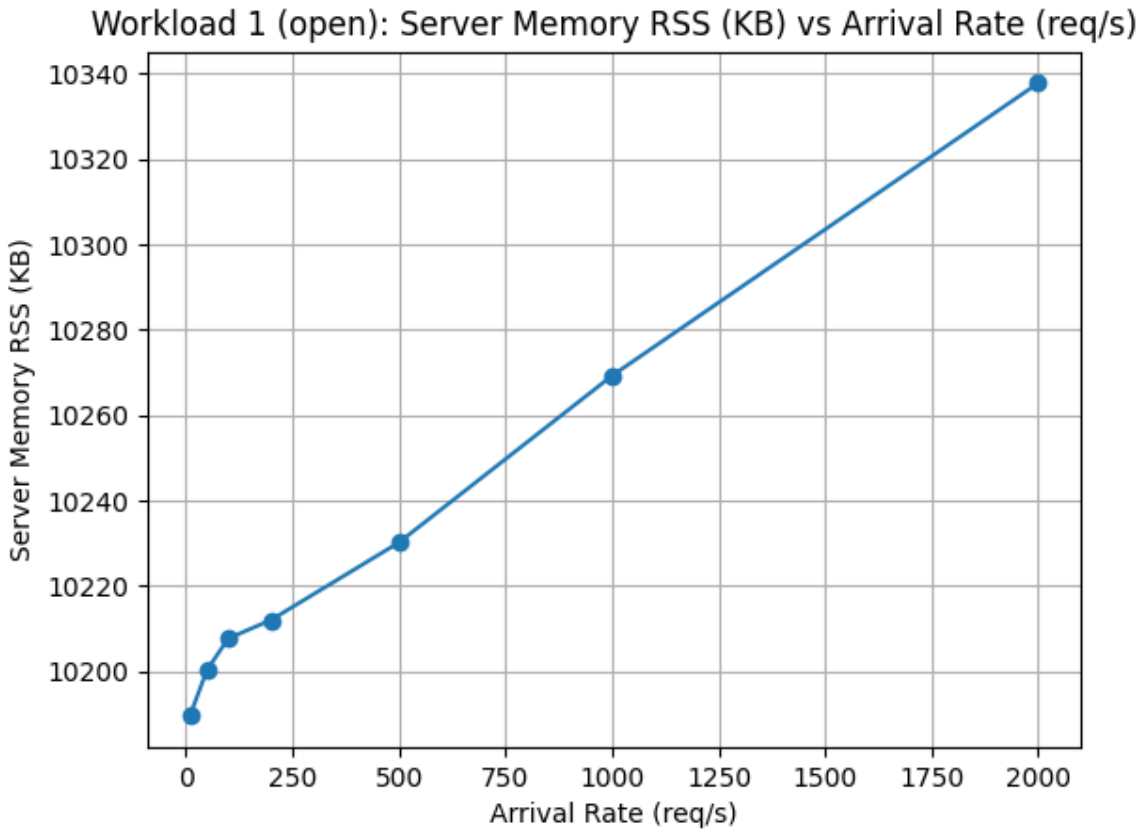

Figure 36: Open Loop: Memory RSS

8.2 Workload 2: Write-Heavy (50% GET)

8.2.1 Performance (Throughput & Latency)

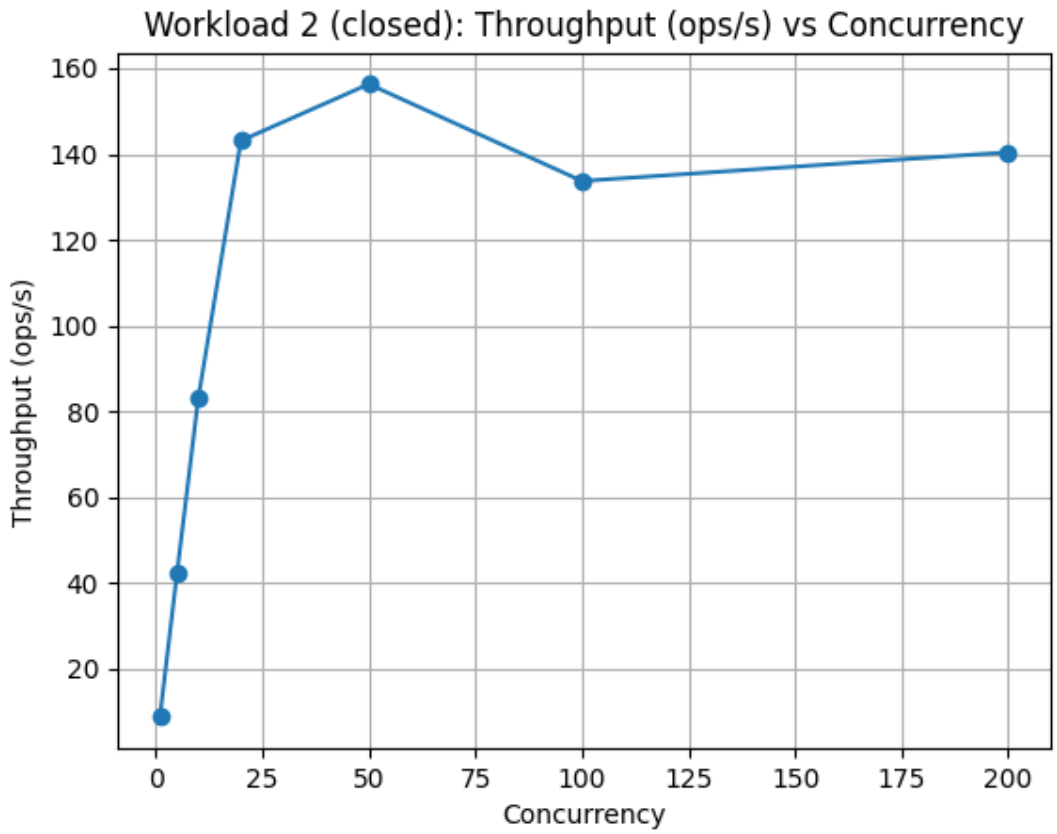

Figure 37: Closed Loop: Throughput vs Concurrency

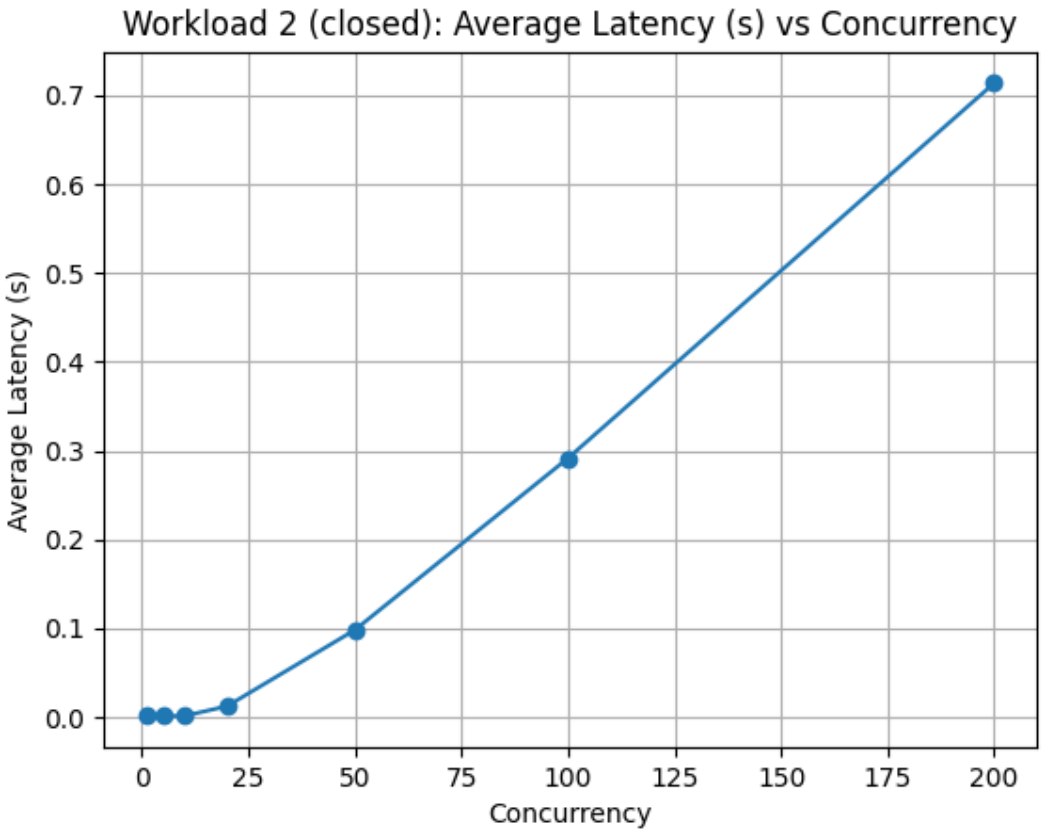

Figure 38: Closed Loop: Avg Latency vs Concurrency

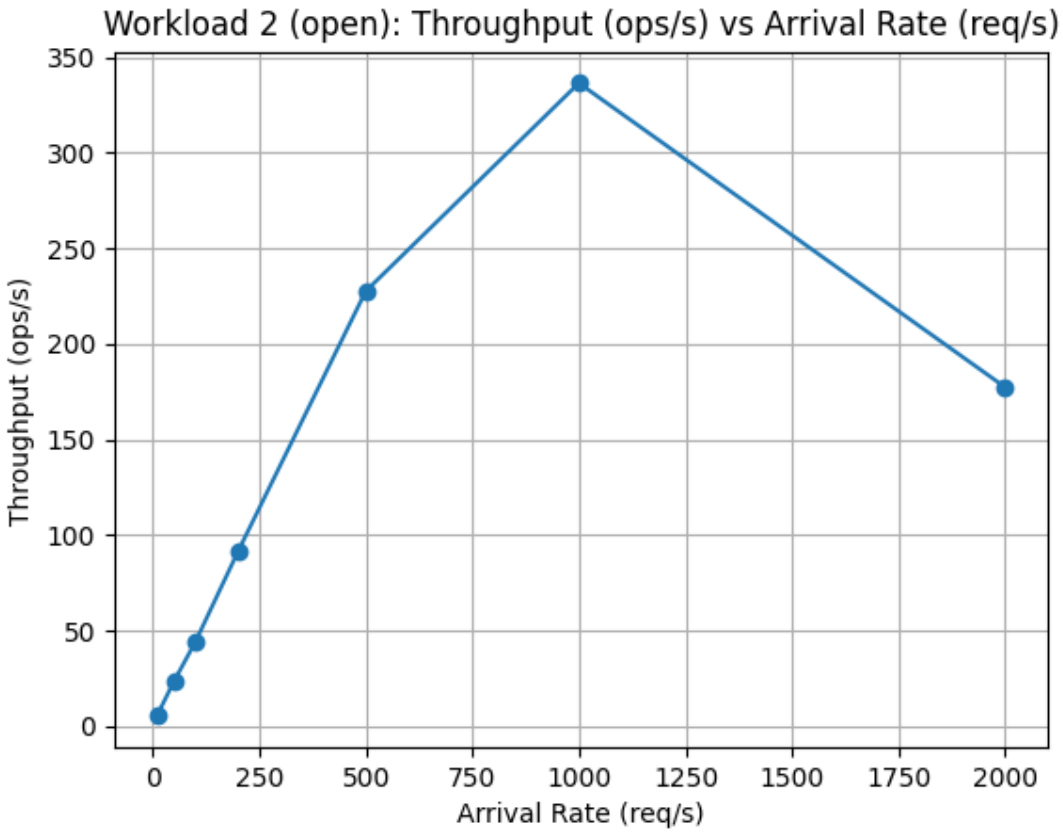

Figure 39: Open Loop: Throughput vs Arrival Rate

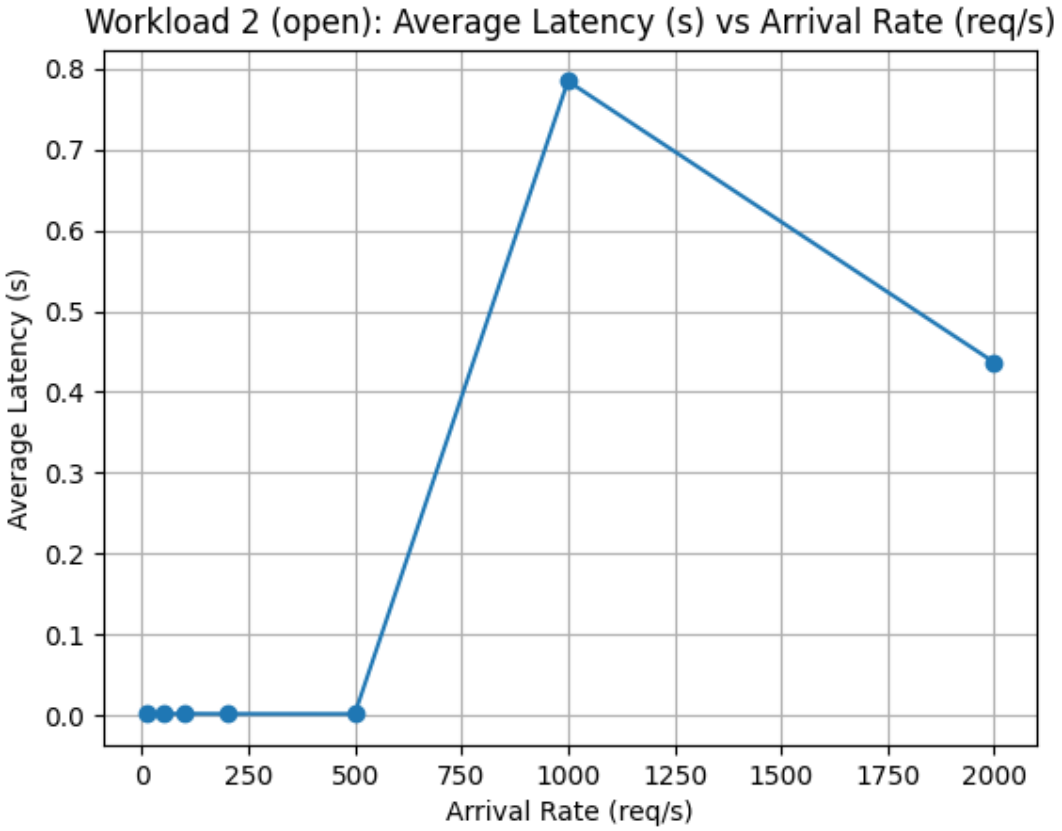

Figure 40: Open Loop: Avg Latency vs Arrival Rate

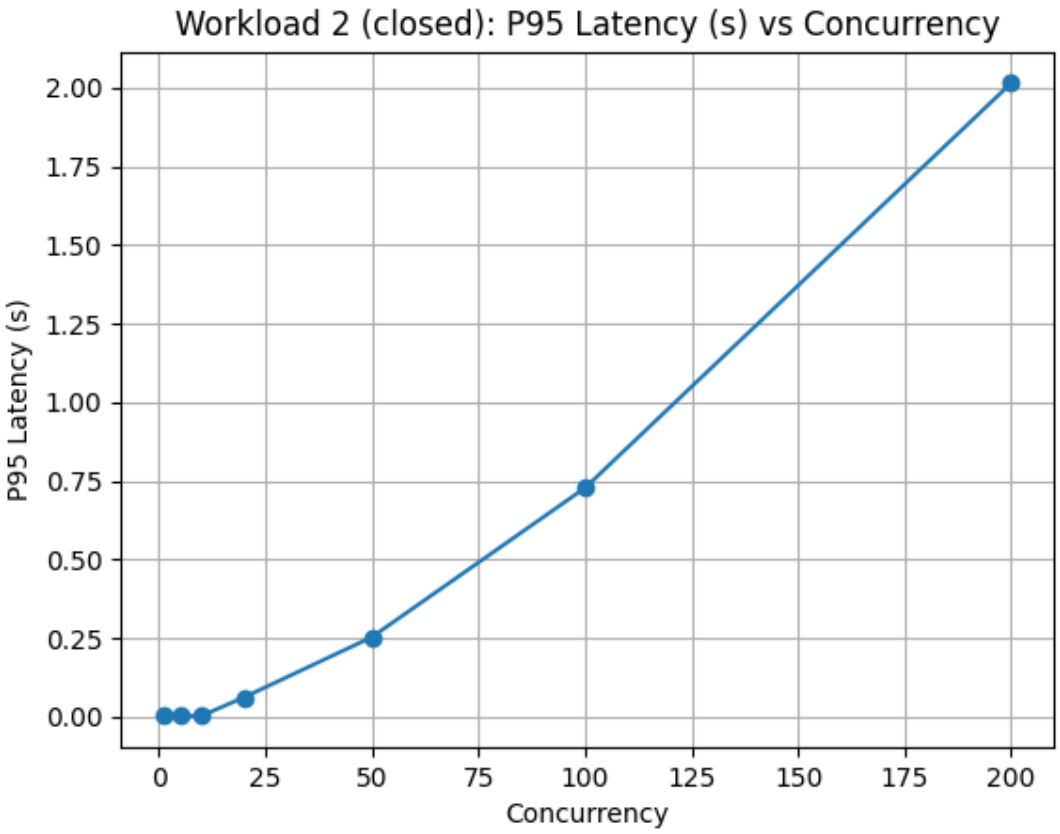

Figure 41: Closed Loop: P95 Latency

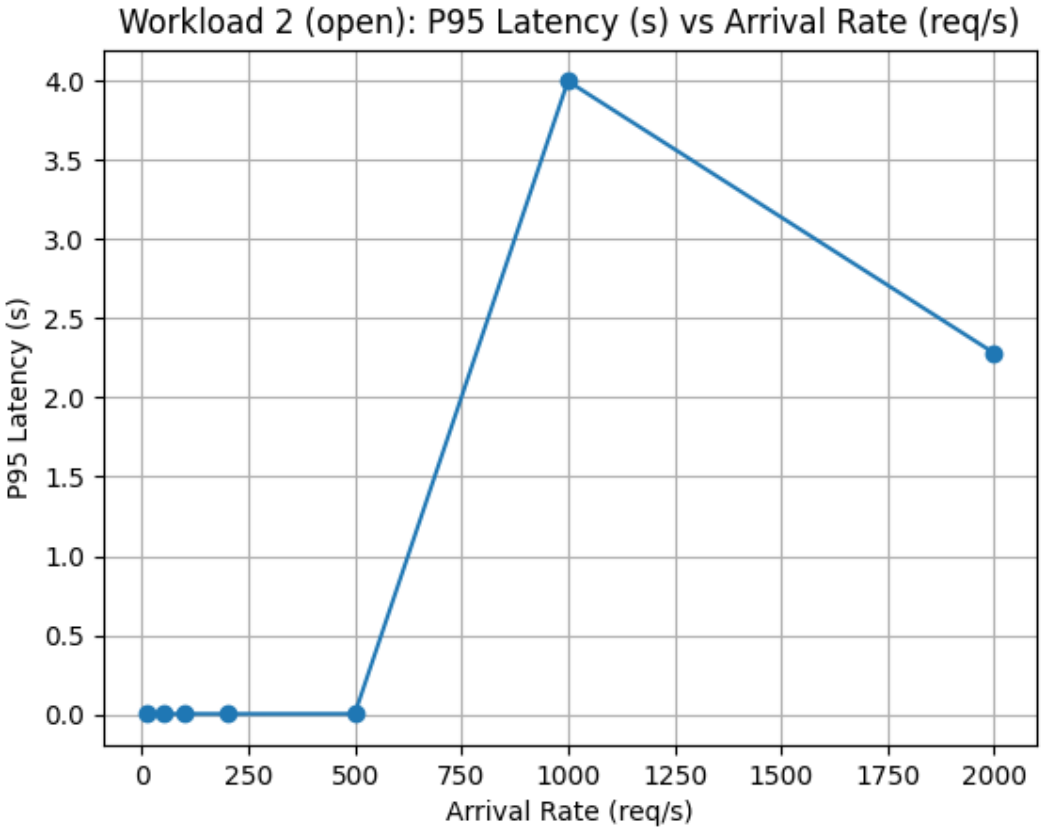

Figure 42: Open Loop: P95 Latency

8.2.2 System Resources (CPU & Disk)

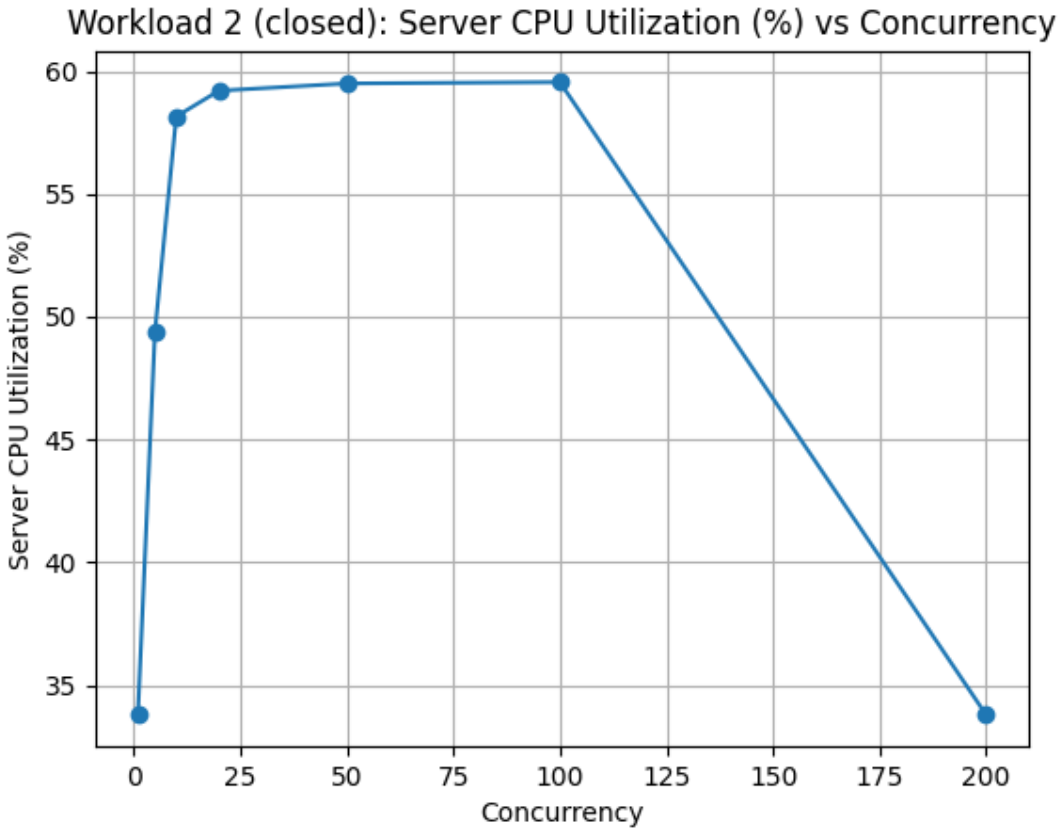

Figure 43: Closed Loop: CPU Utilization

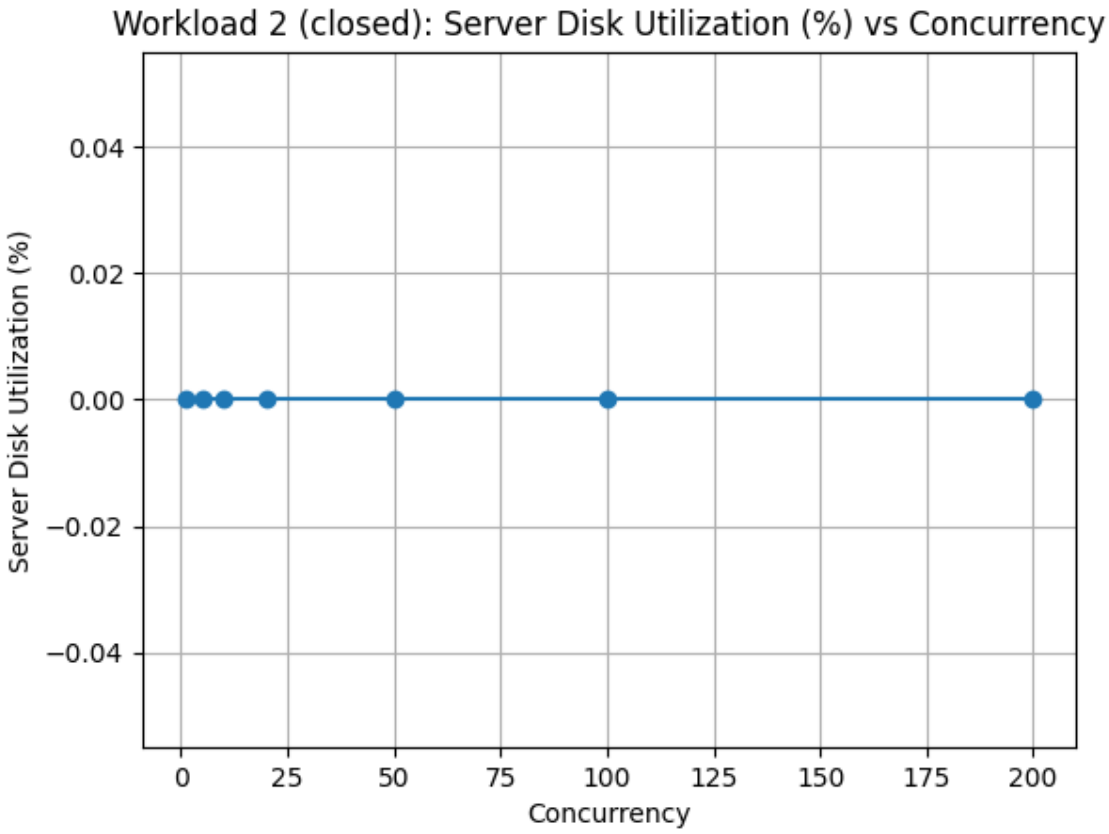

Figure 44: Closed Loop: Disk Utilization

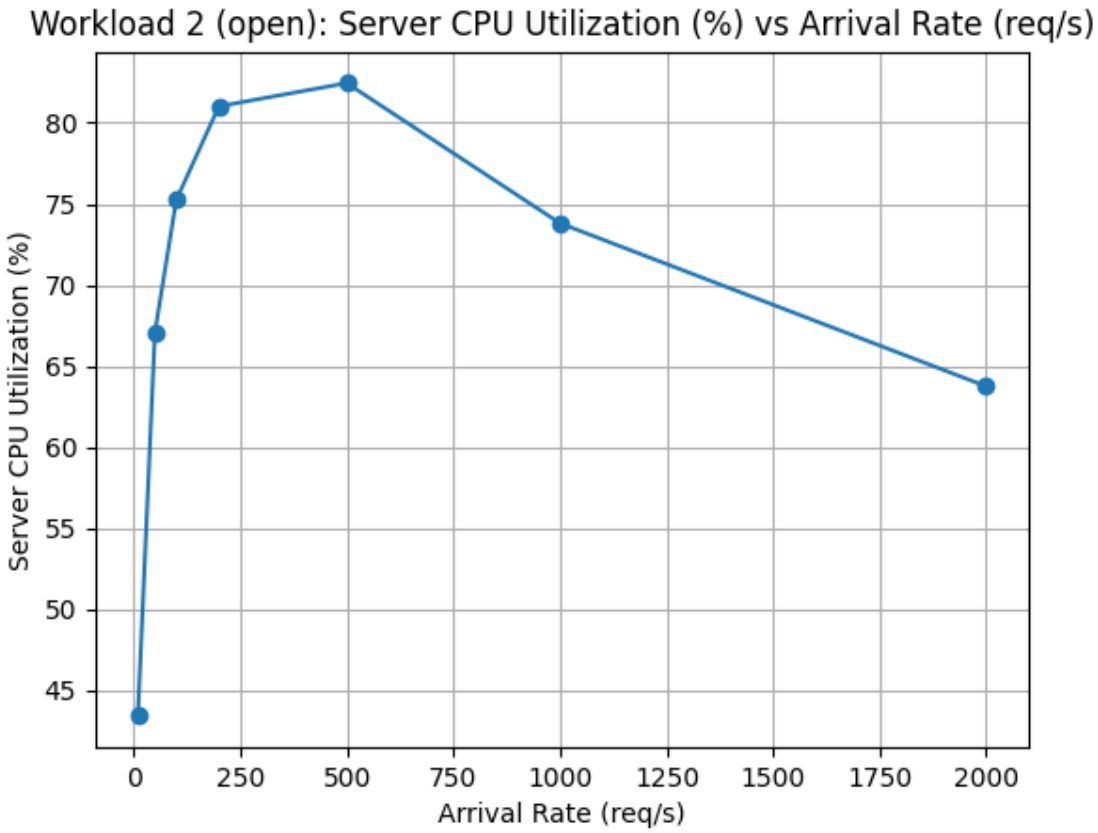

Figure 45: Open Loop: CPU Utilization

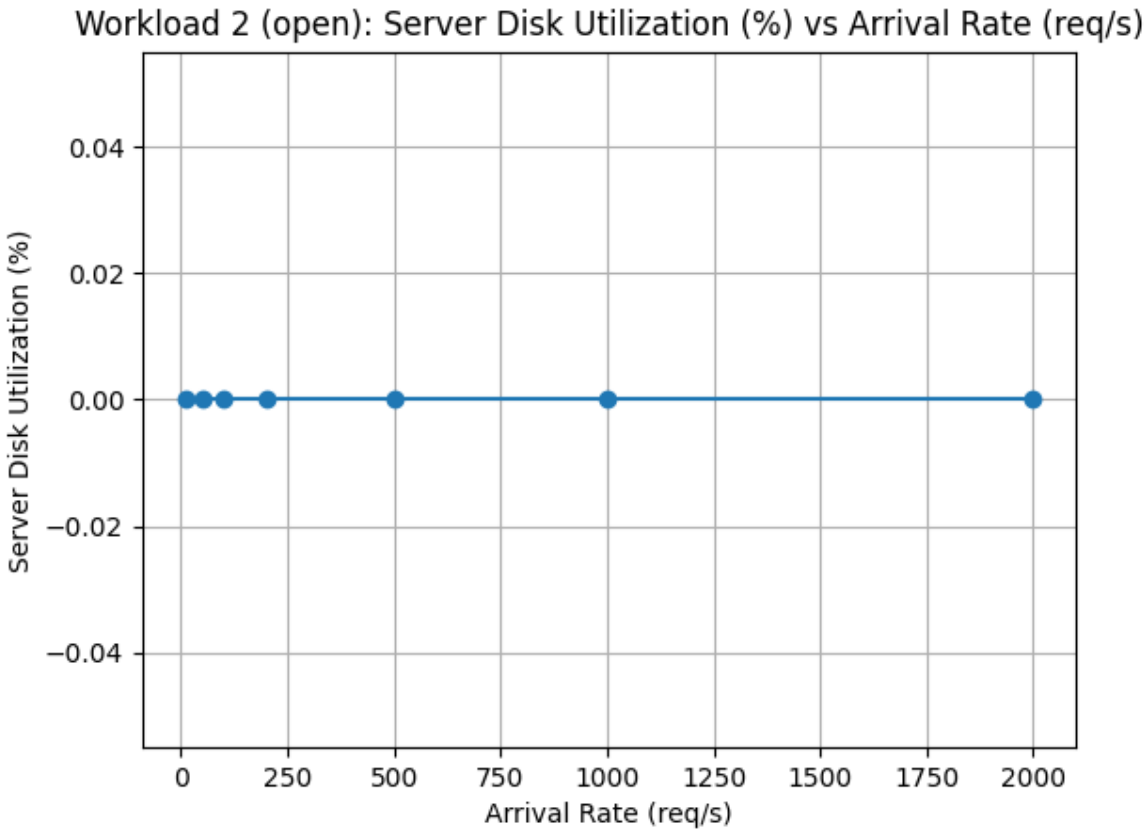

Figure 46: Open Loop: Disk Utilization

8.2.3 Cache & Memory

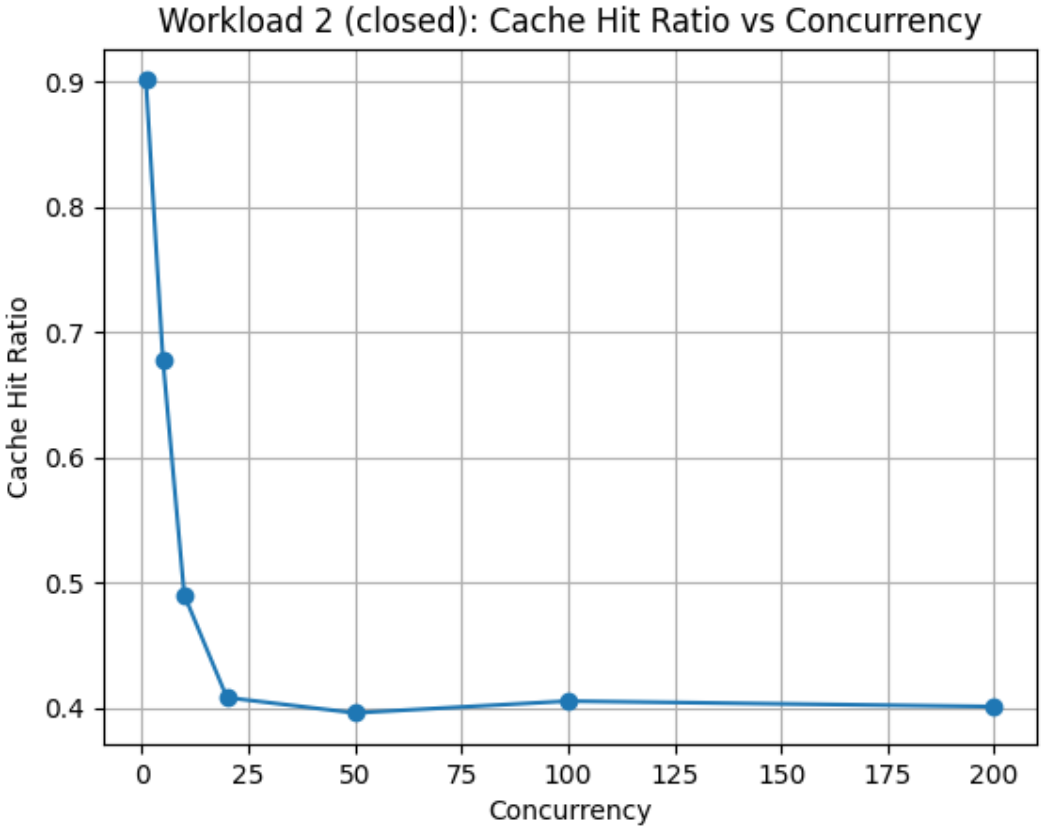

Figure 47: Closed Loop: Cache Hit Ratio

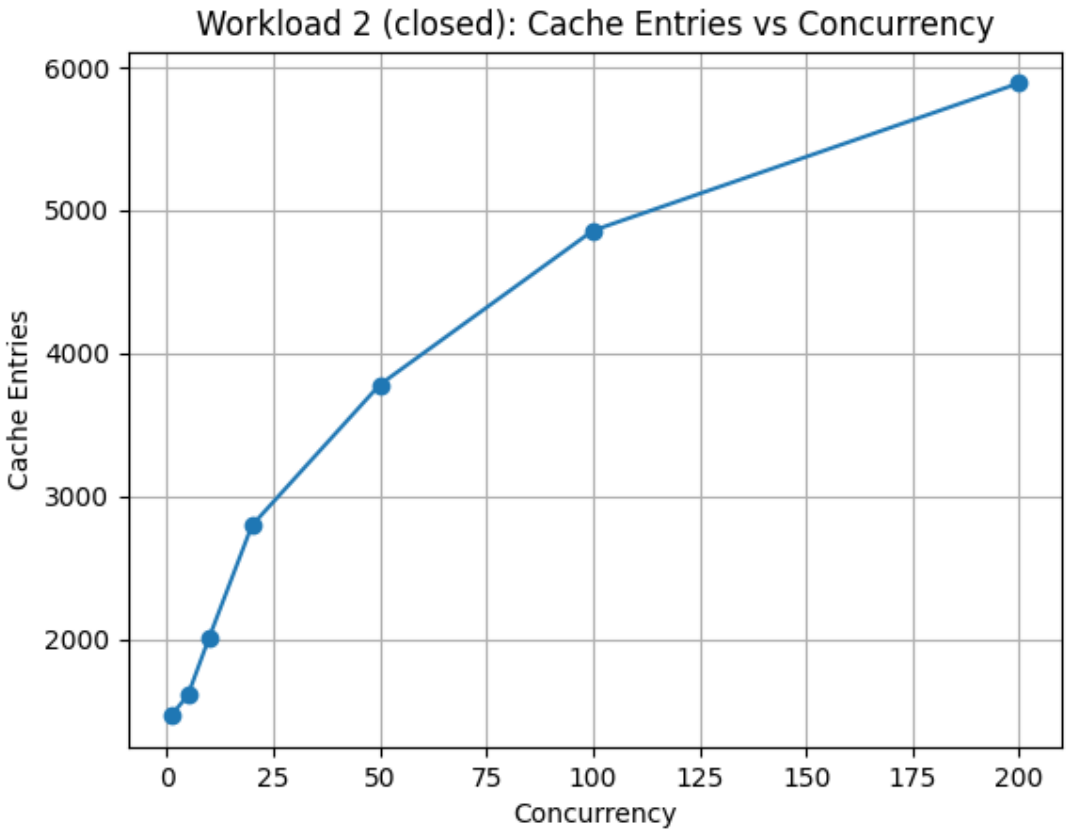

Figure 48: Closed Loop: Cache Entries

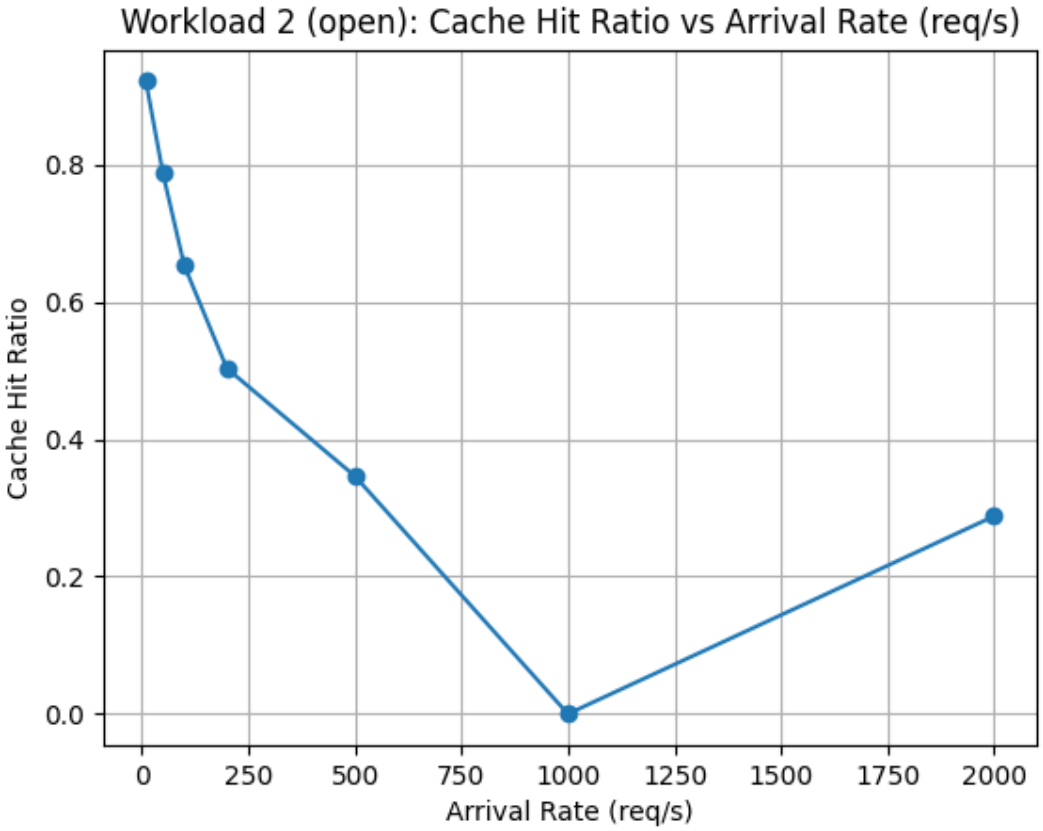

Figure 49: Open Loop: Cache Hit Ratio

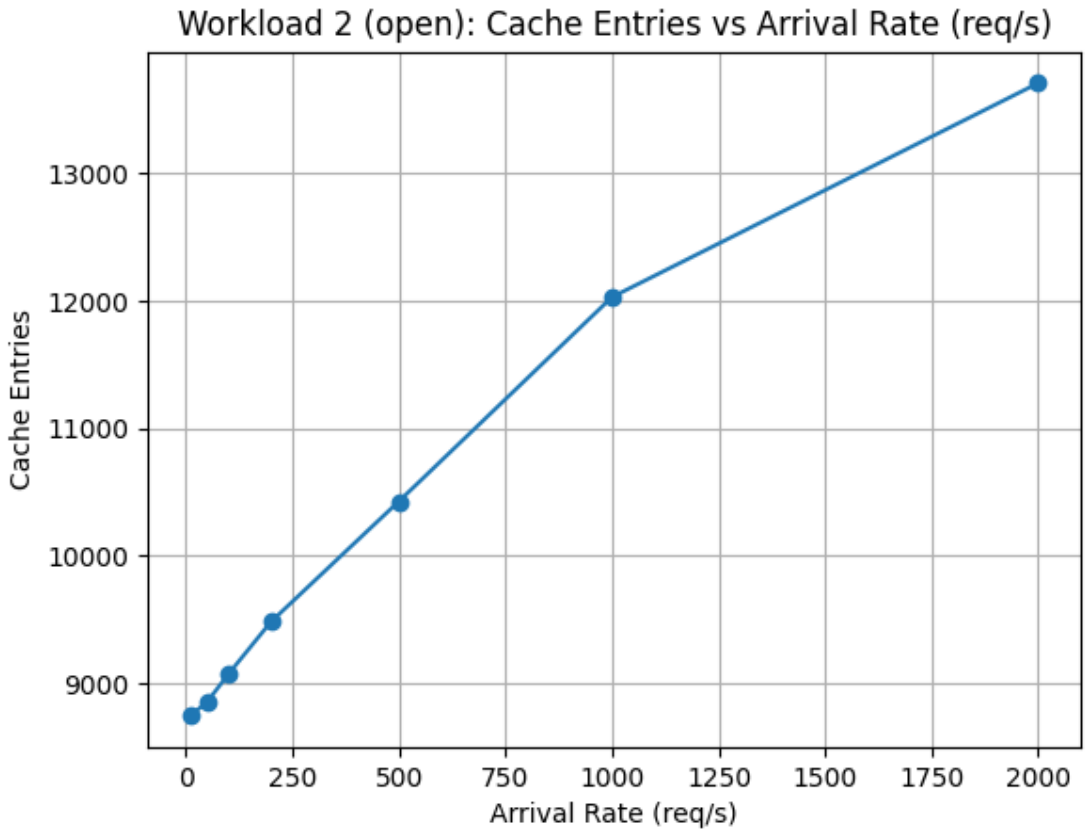

Figure 50: Open Loop: Cache Entries

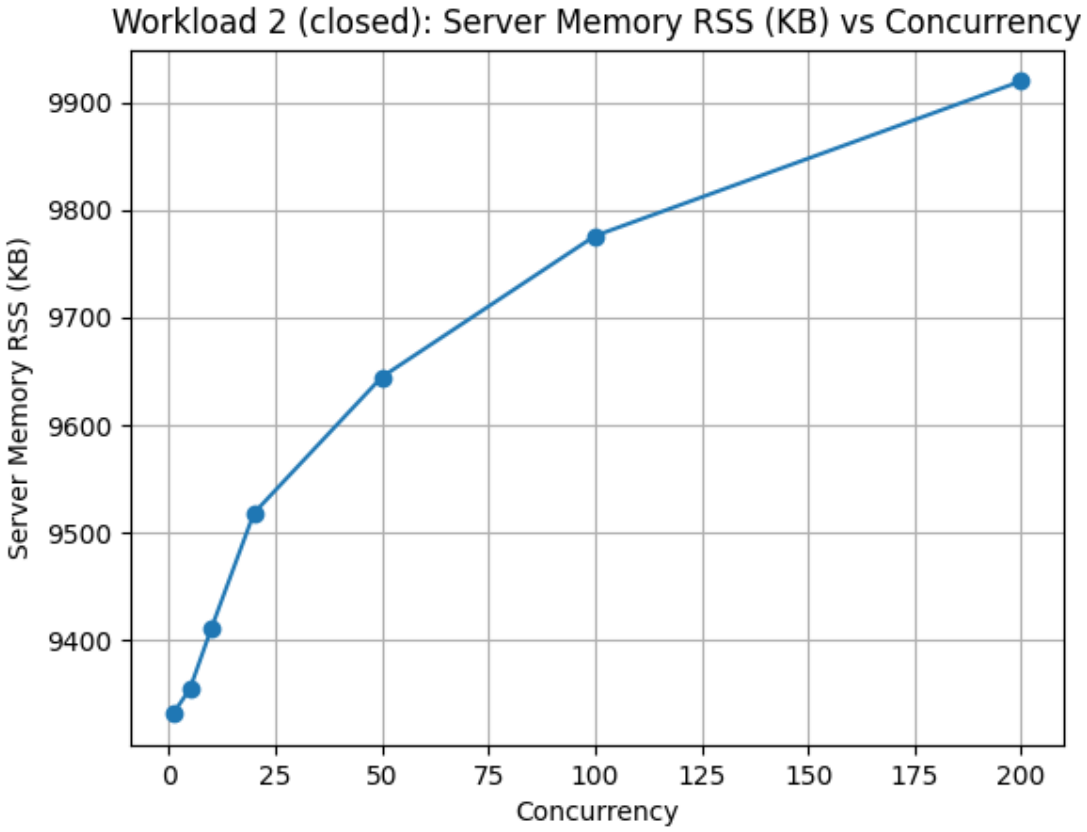

Figure 51: Closed Loop: Memory RSS

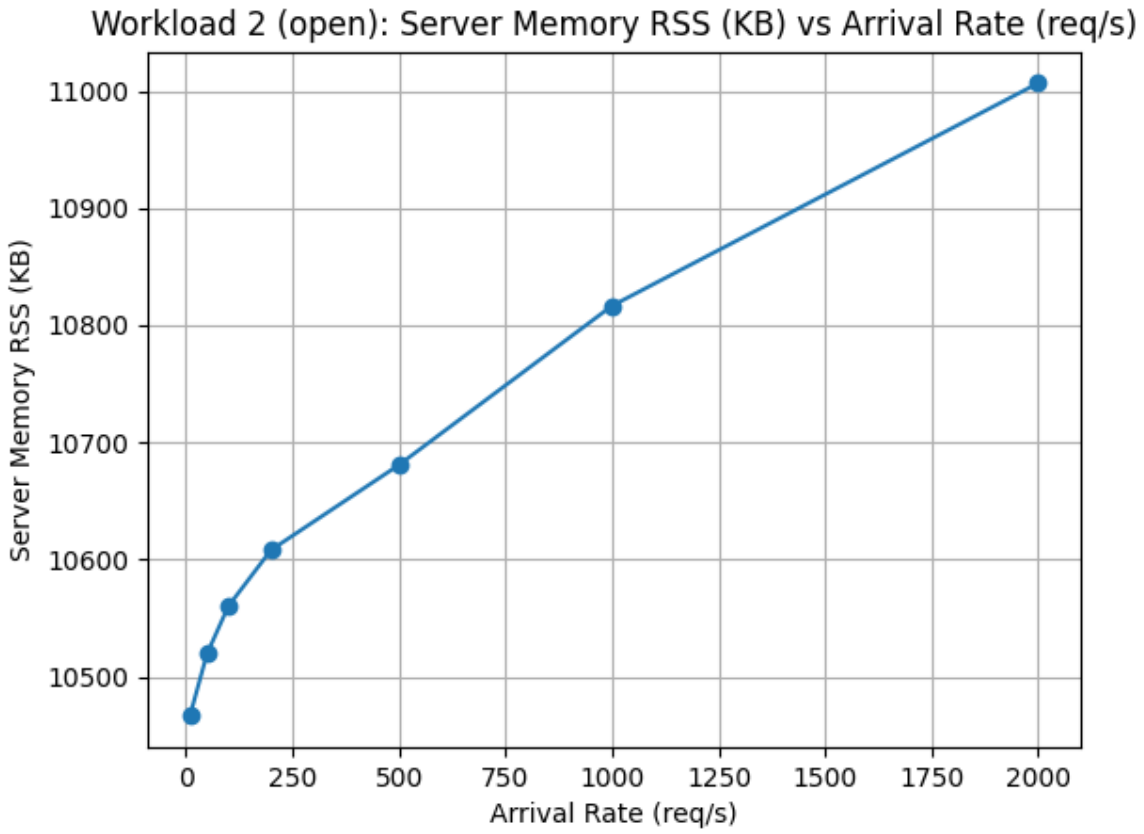

Figure 52: Open Loop: Memory RSS

8.3 Workload 3: Balanced (Mixed)

8.3.1 Performance (Throughput & Latency)

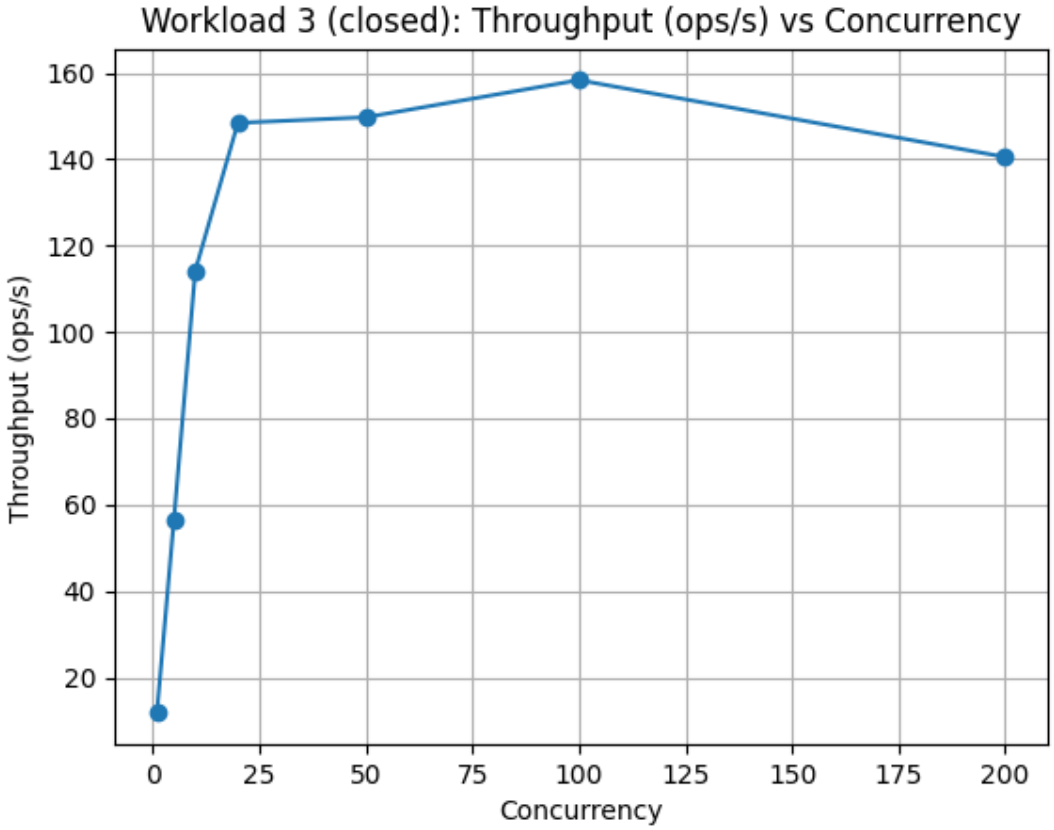

Figure 53: Closed Loop: Throughput vs Concurrency

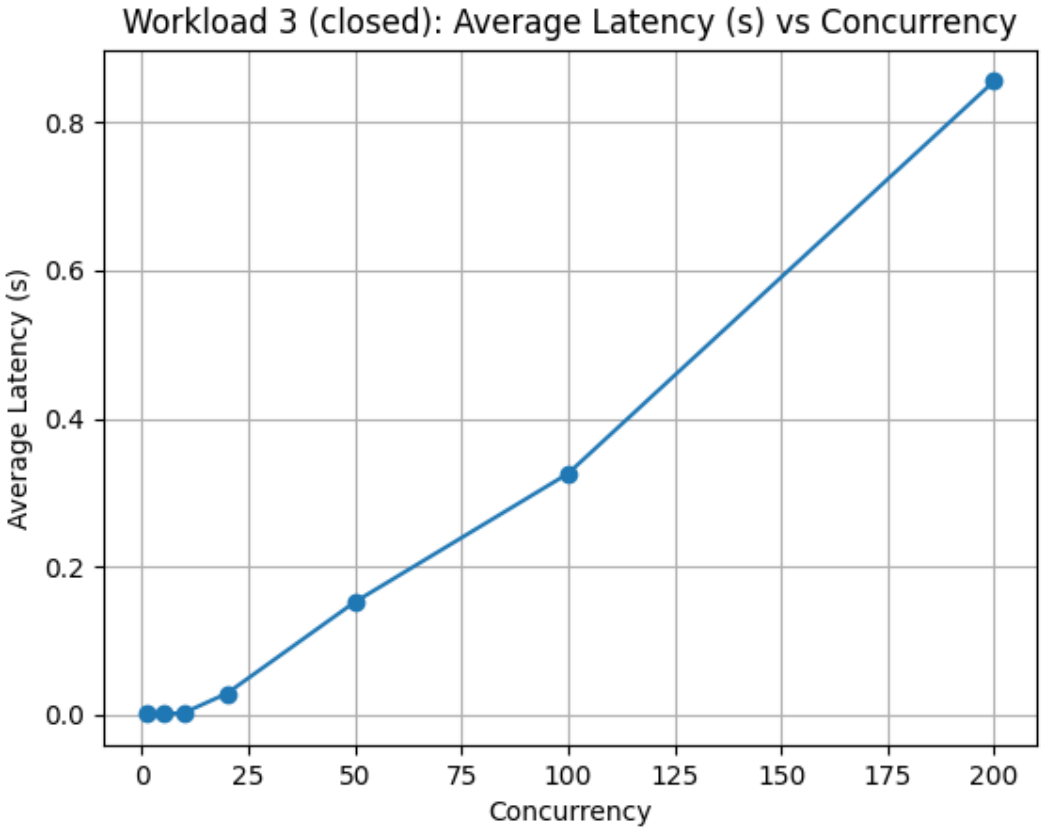

Figure 54: Closed Loop: Avg Latency vs Concurrency

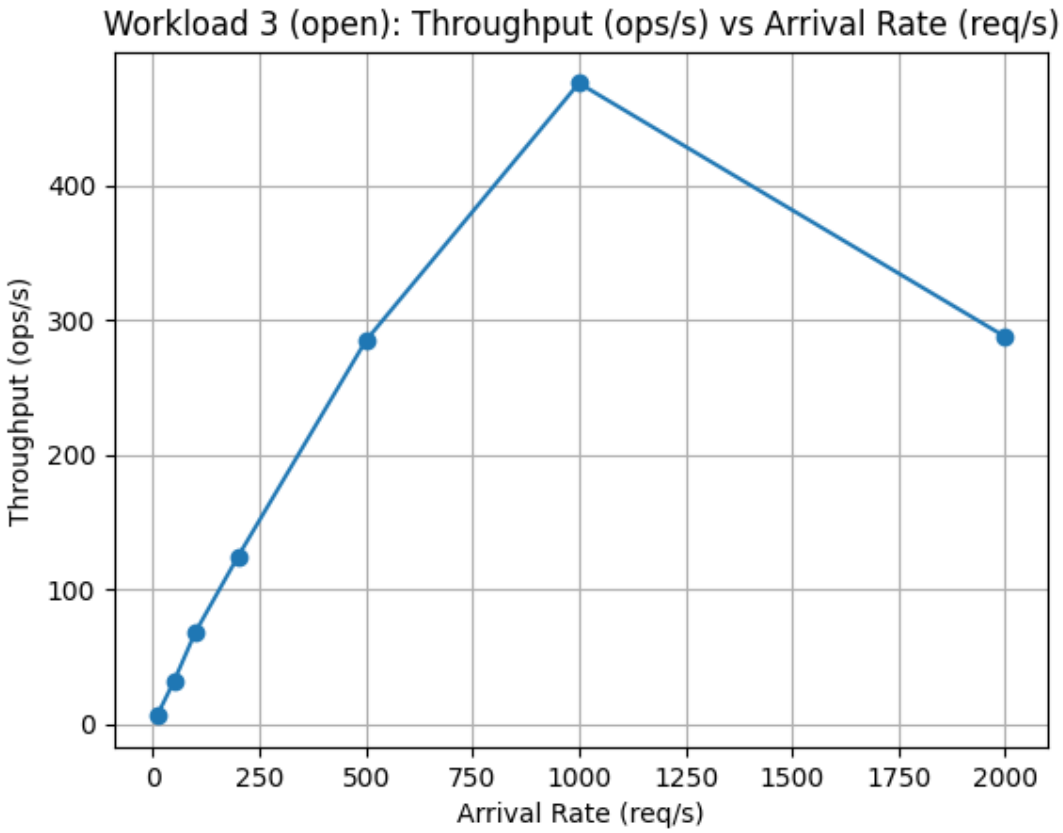

Figure 55: Open Loop: Throughput vs Arrival Rate

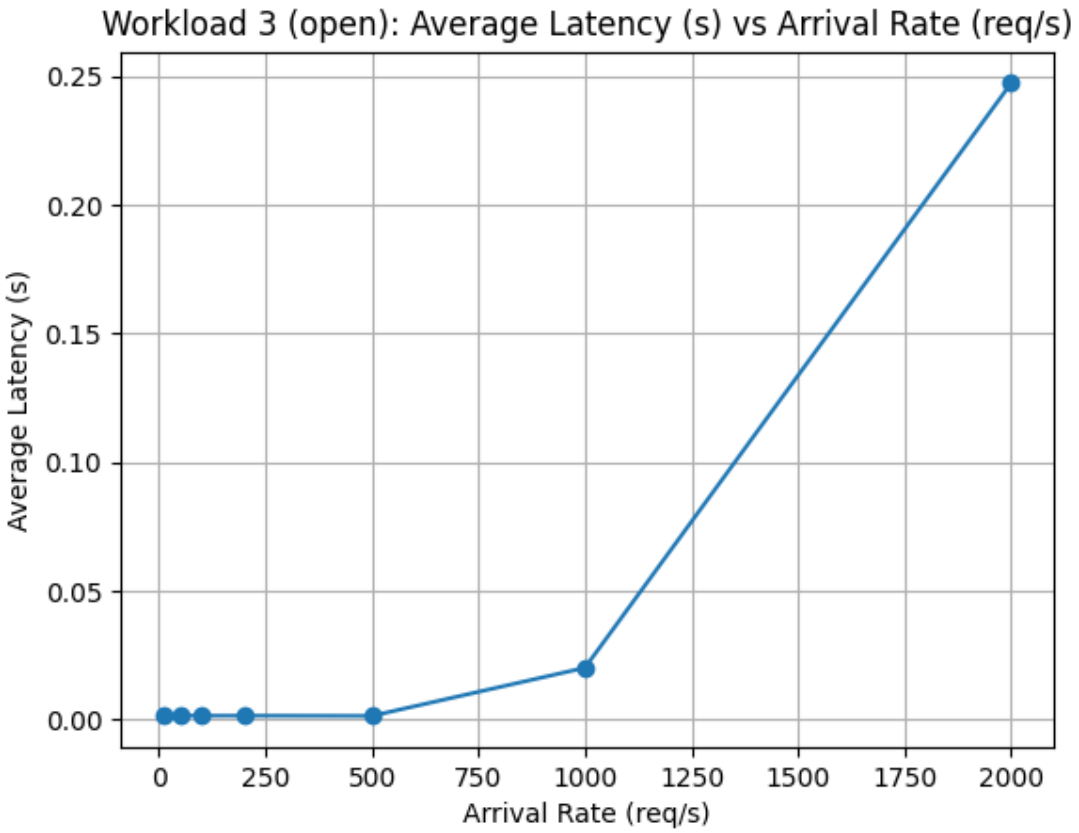

Figure 56: Open Loop: Avg Latency vs Arrival Rate

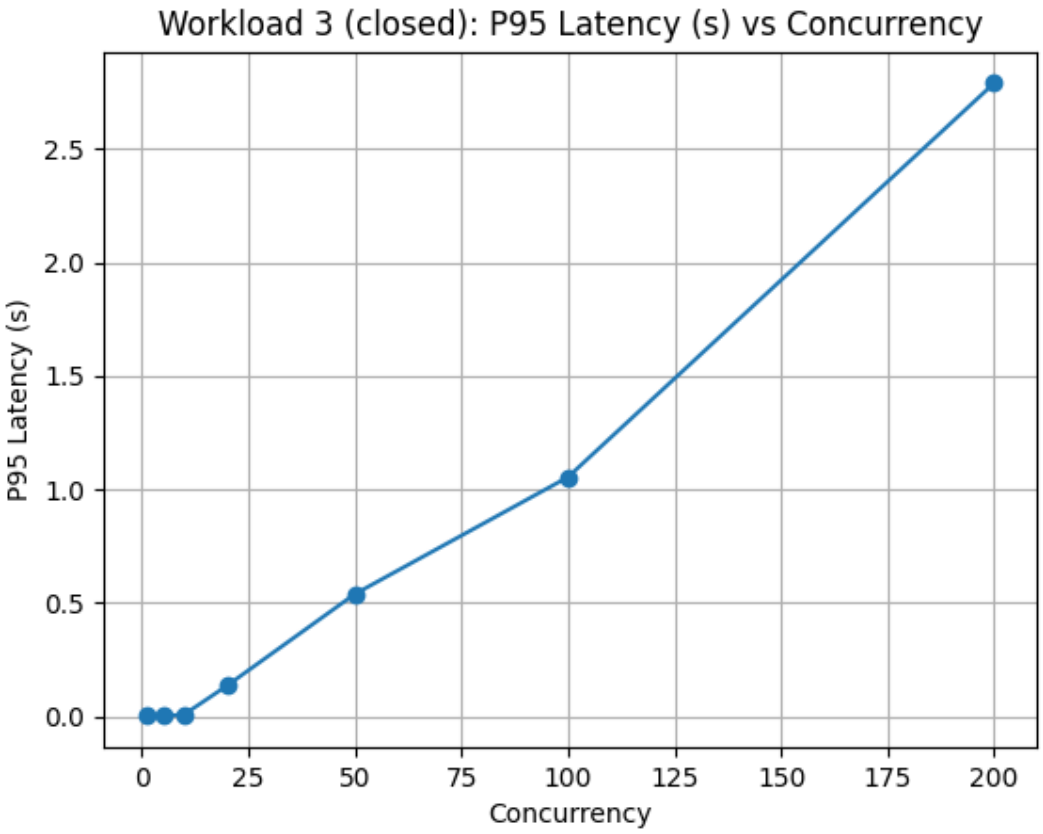

Figure 57: Closed Loop: P95 Latency

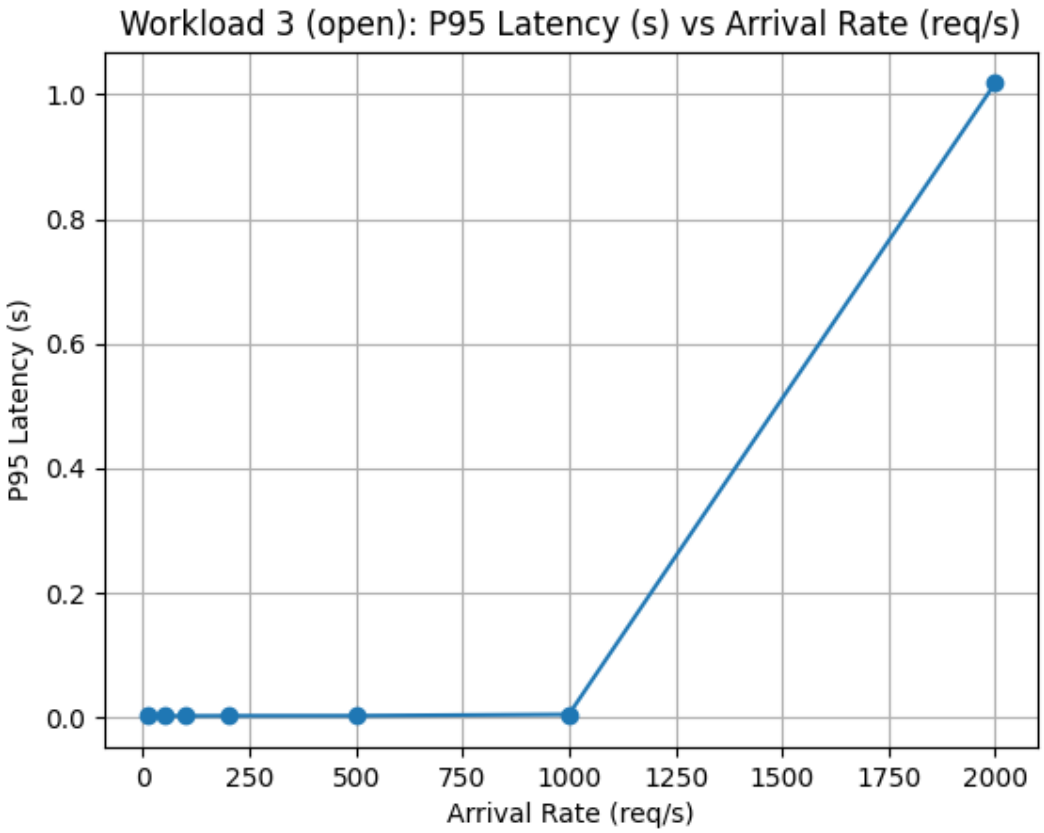

Figure 58: Open Loop: P95 Latency

8.3.2 System Resources (CPU & Disk)

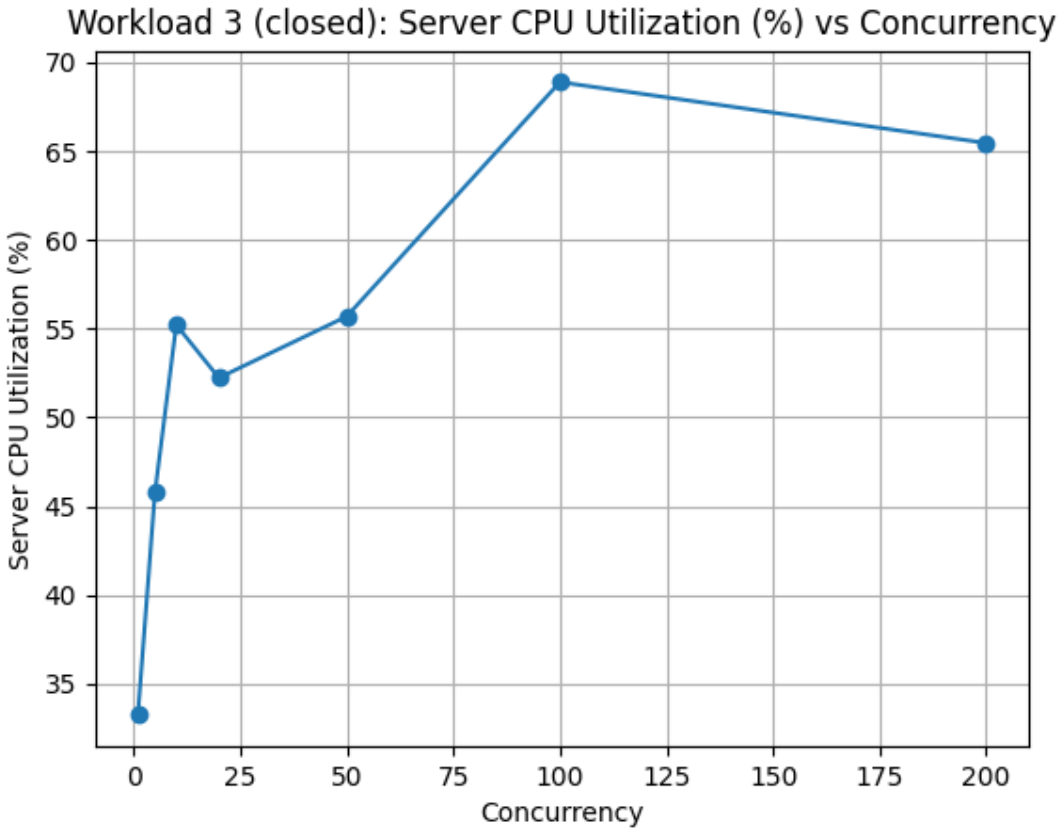

Figure 59: Closed Loop: CPU Utilization

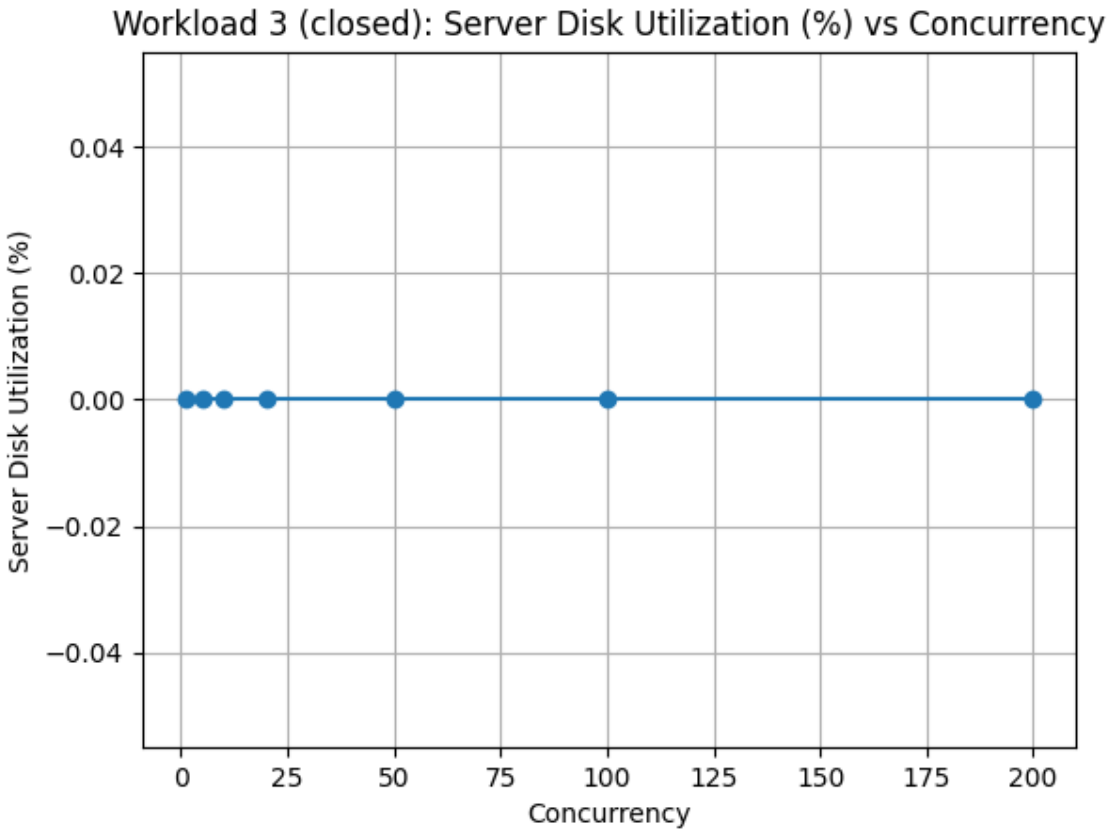

Figure 60: Closed Loop: Disk Utilization

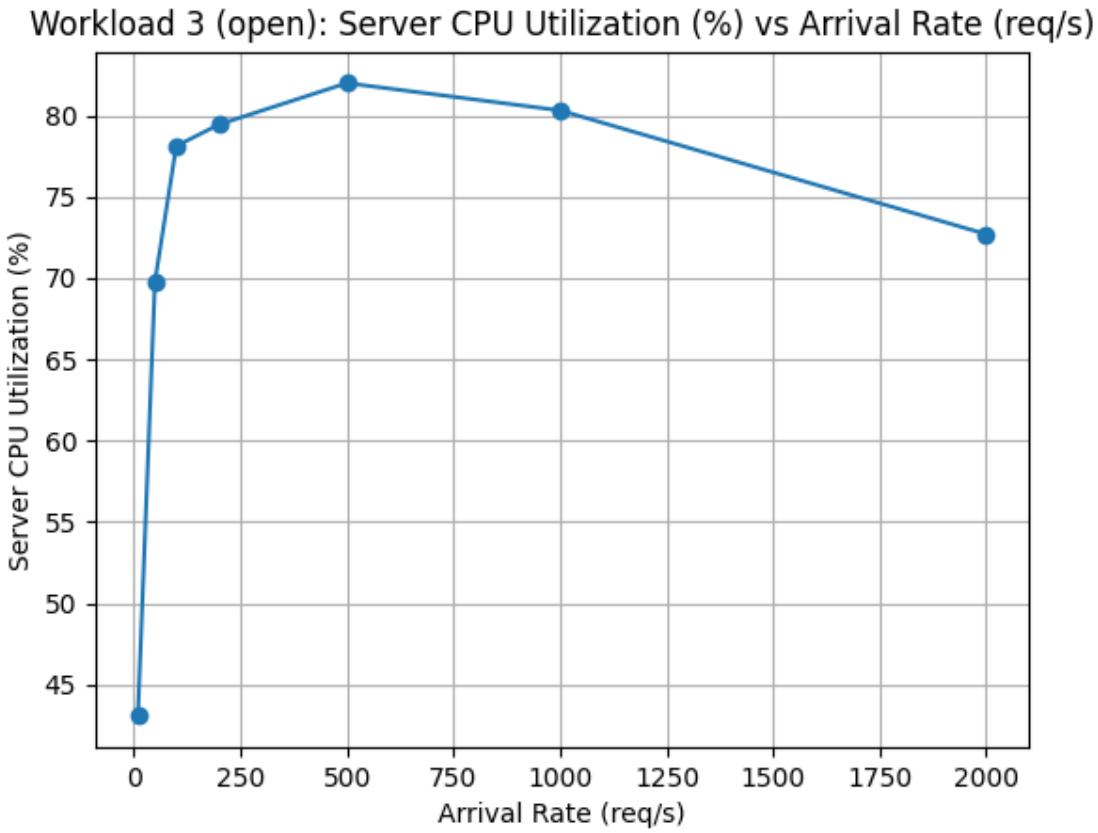

Figure 61: Open Loop: CPU Utilization

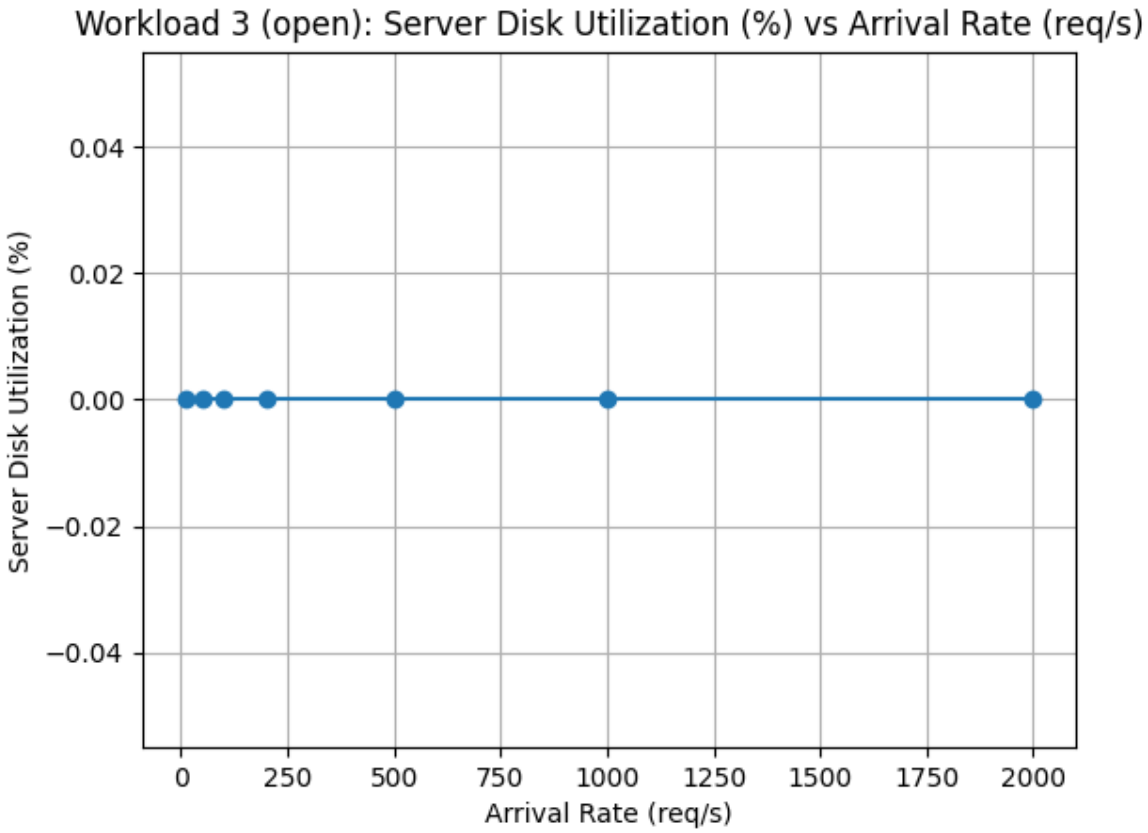

Figure 62: Open Loop: Disk Utilization

8.3.3 Cache & Memory

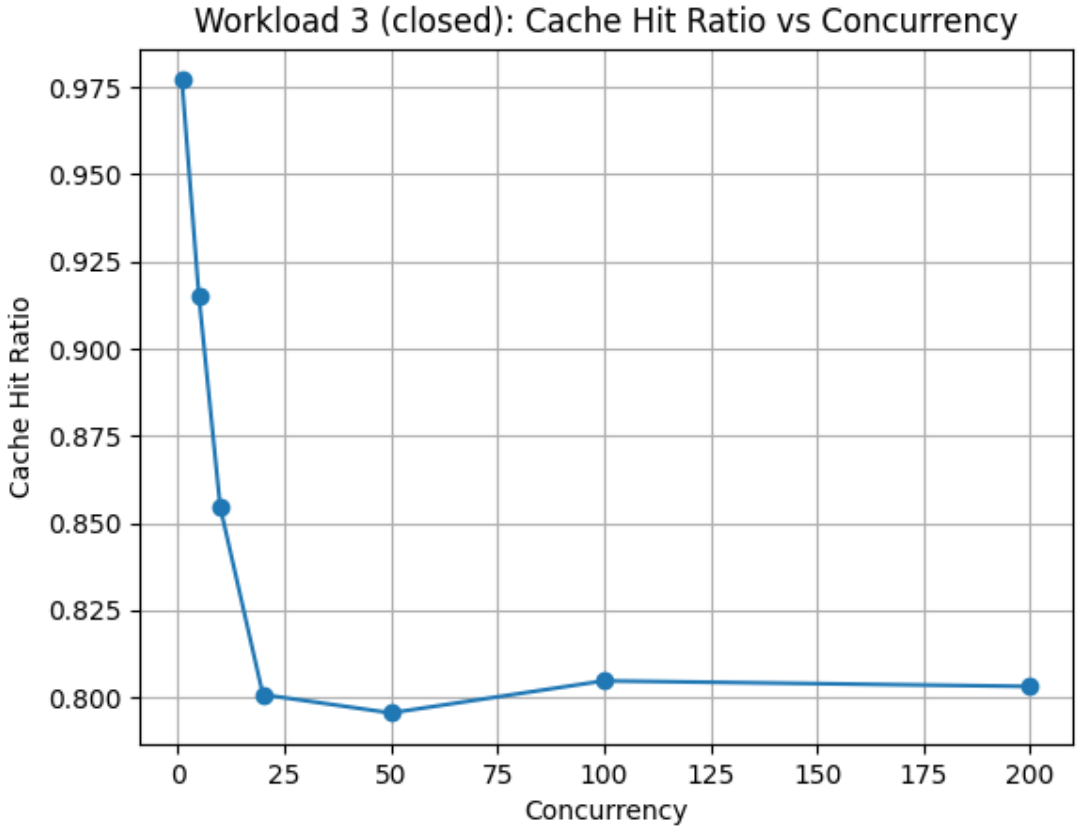

Figure 63: Closed Loop: Cache Hit Ratio

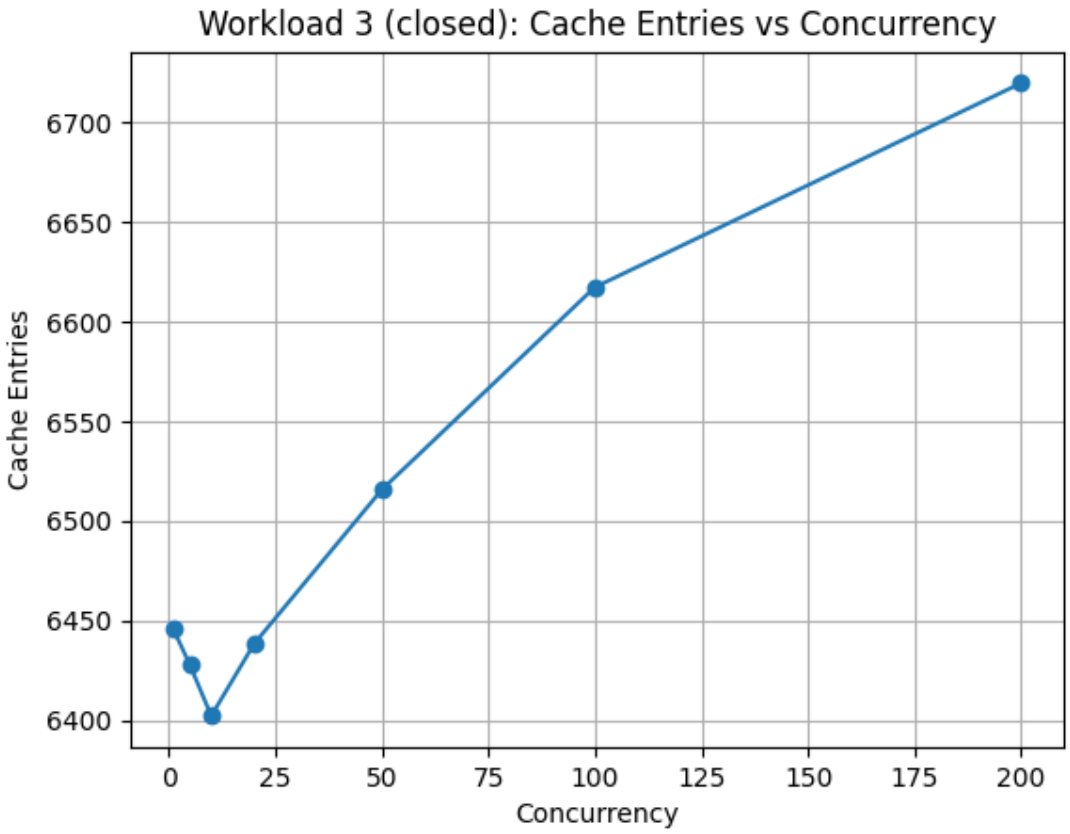

Figure 64: Closed Loop: Cache Entries

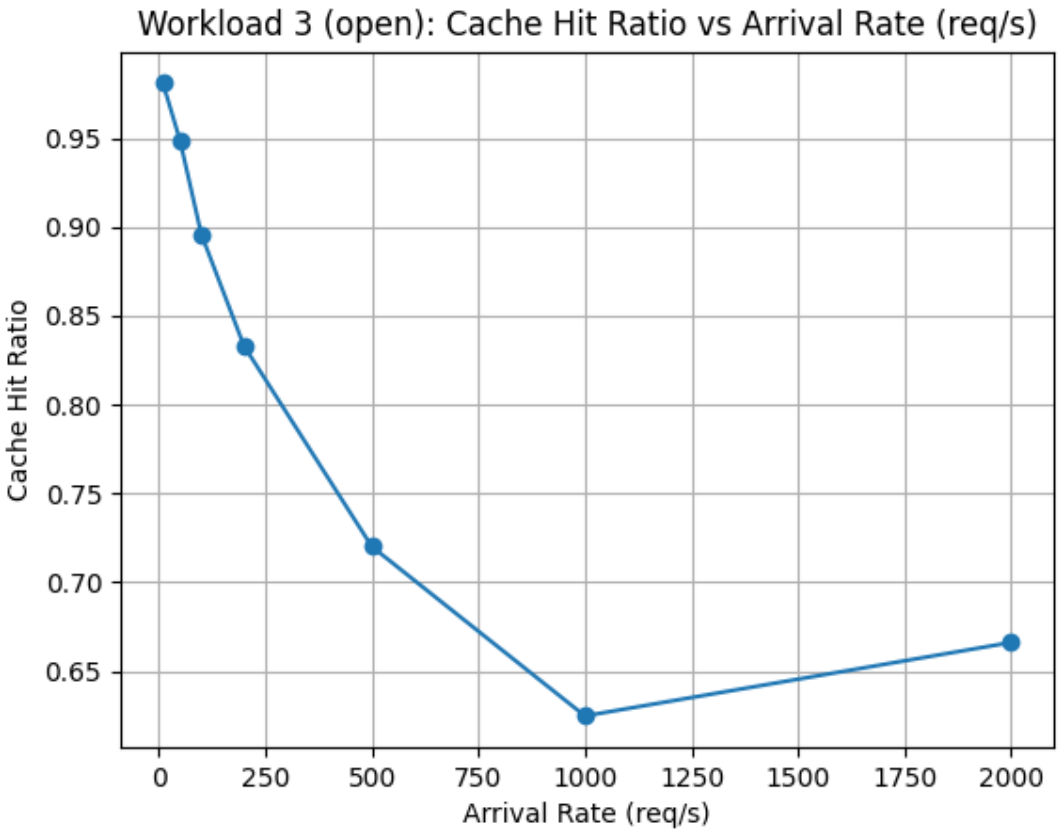

Figure 65: Open Loop: Cache Hit Ratio

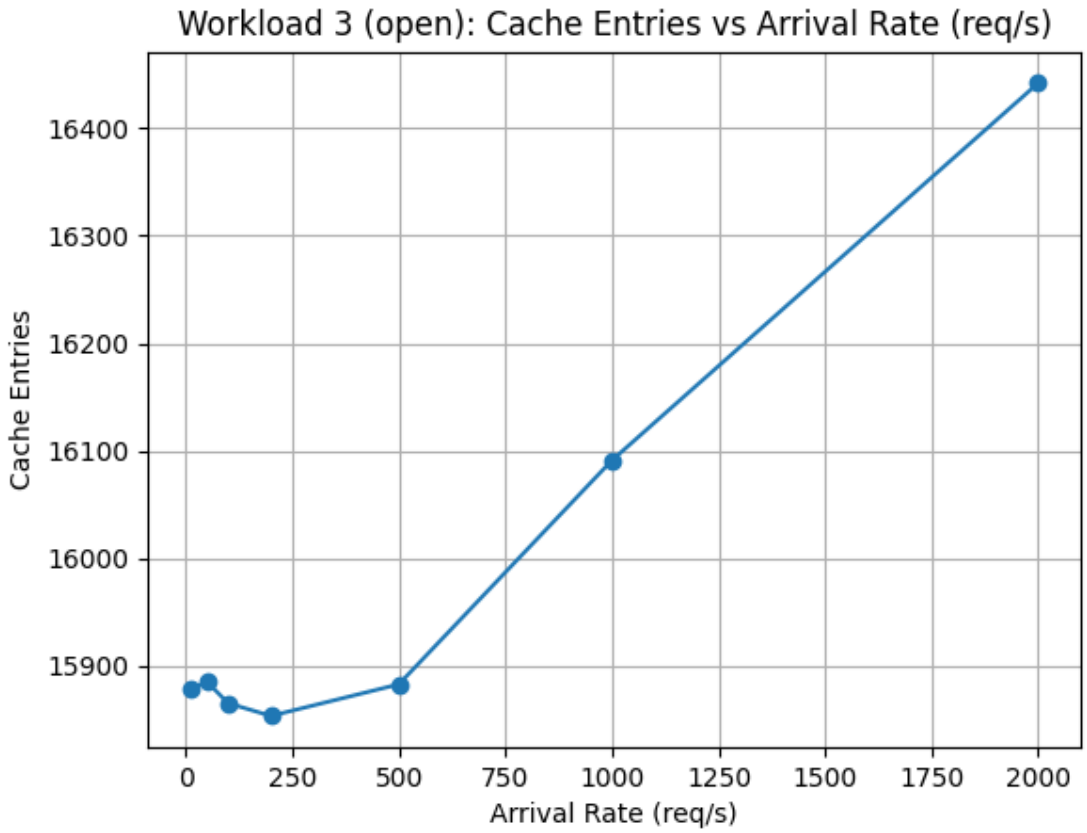

Figure 66: Open Loop: Cache Entries

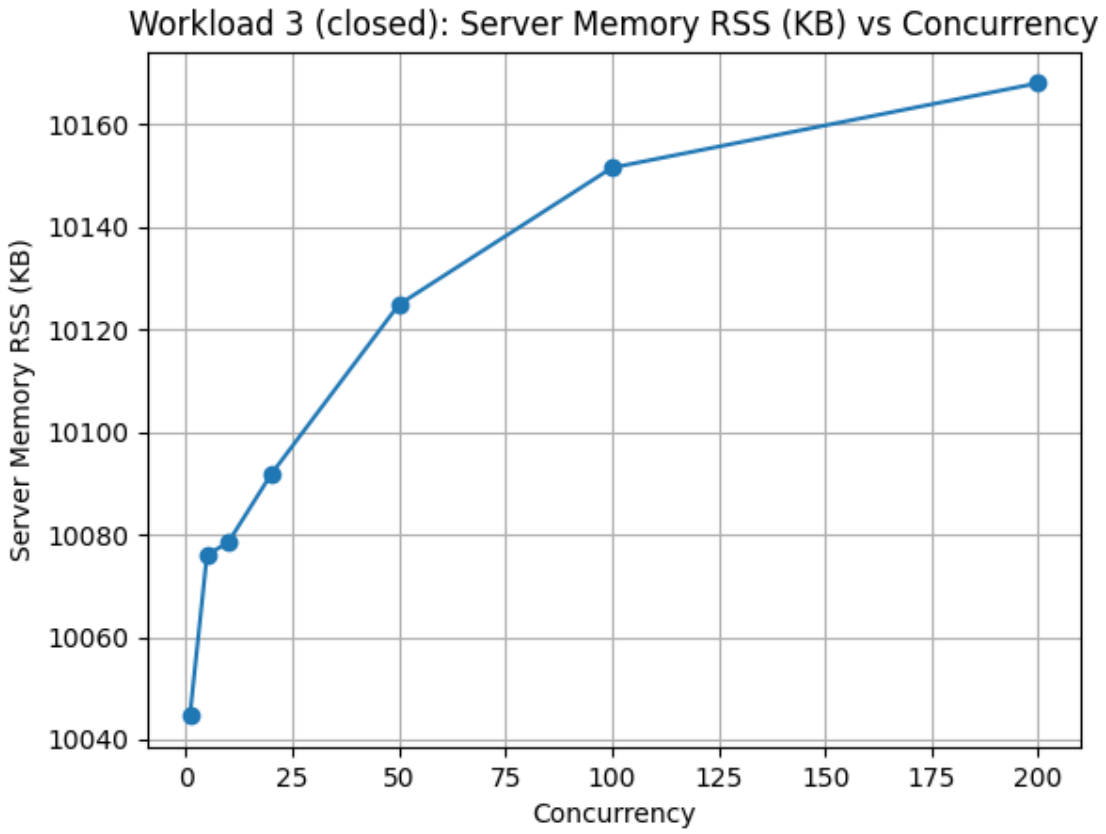

Figure 67: Closed Loop: Memory RSS

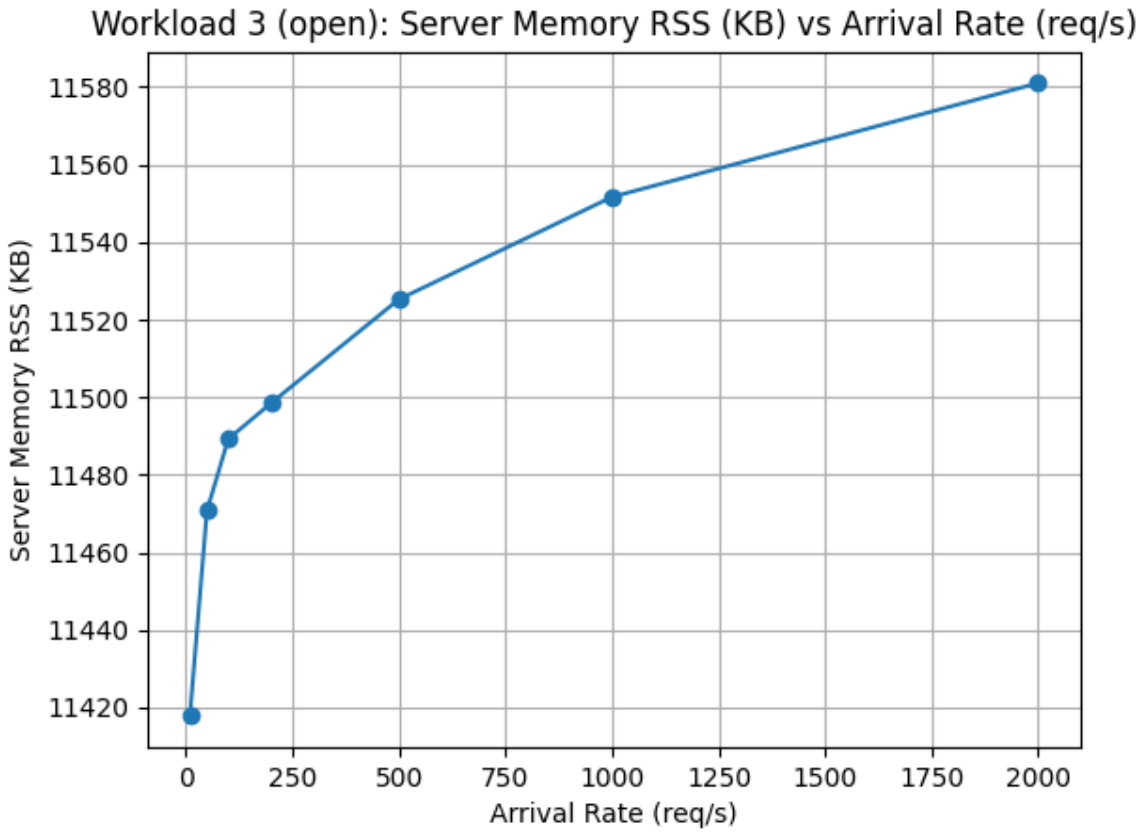

Figure 68: Open Loop: Memory RSS

8.4 Workload 4: CPU Saturation (100% GET Hot Keys)

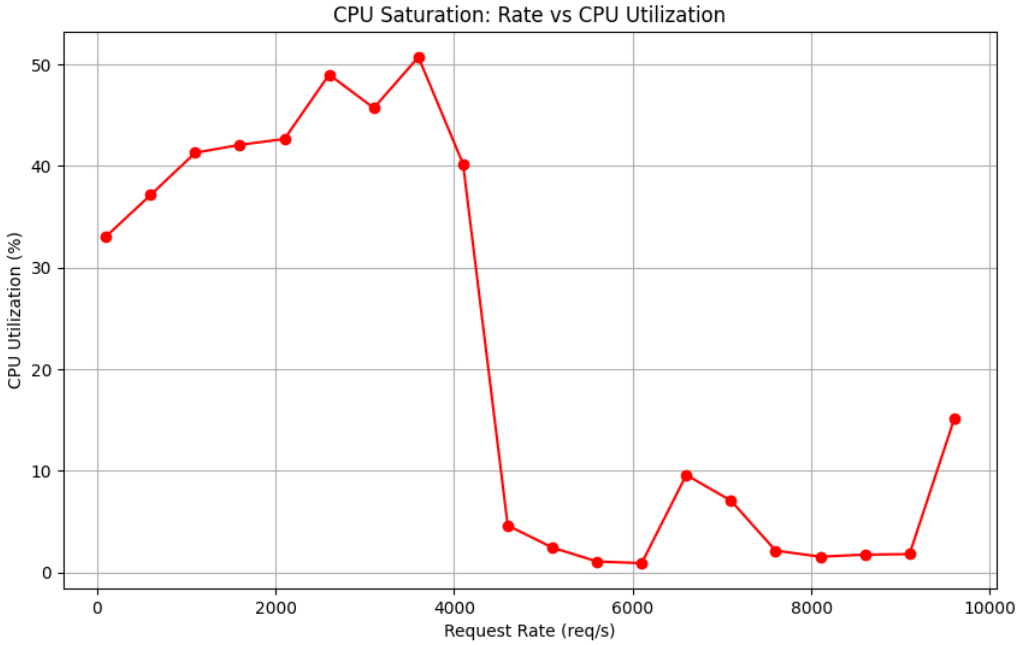

Figure 69: Rate vs CPU Utilization

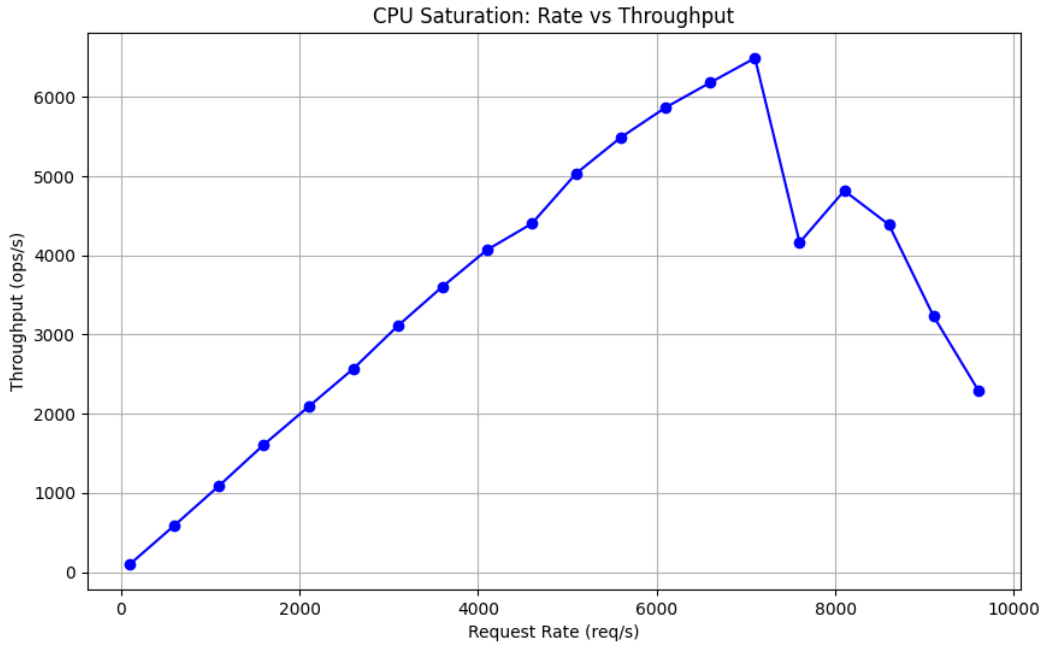

Figure 70: Rate vs Throughput

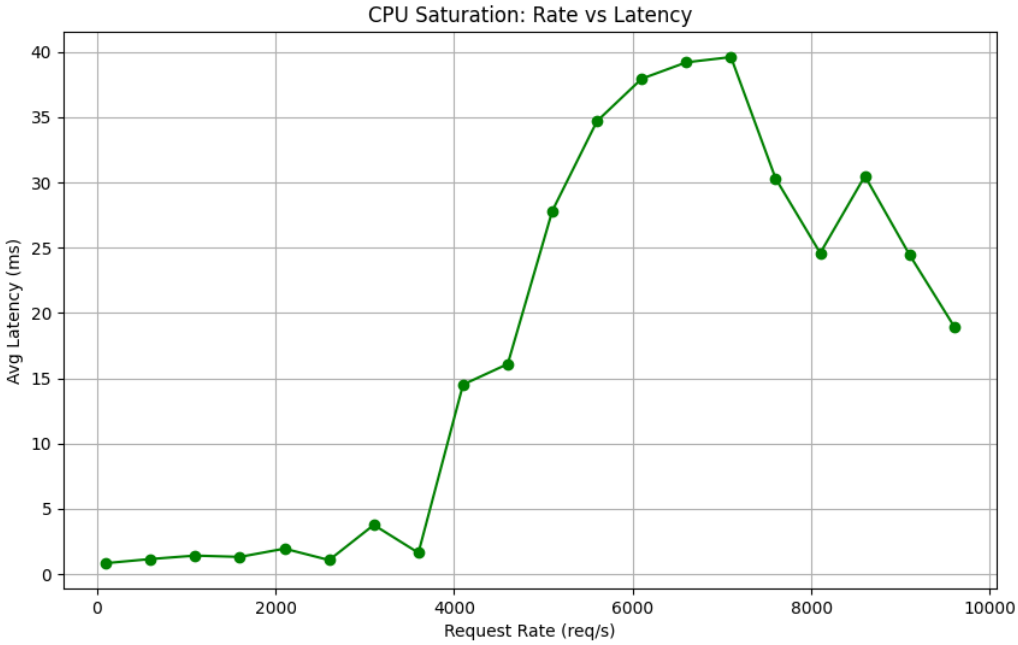

Figure 71: Rate vs Latency

8.5 Workload 5: Disk Saturation (100% Write 1KB)

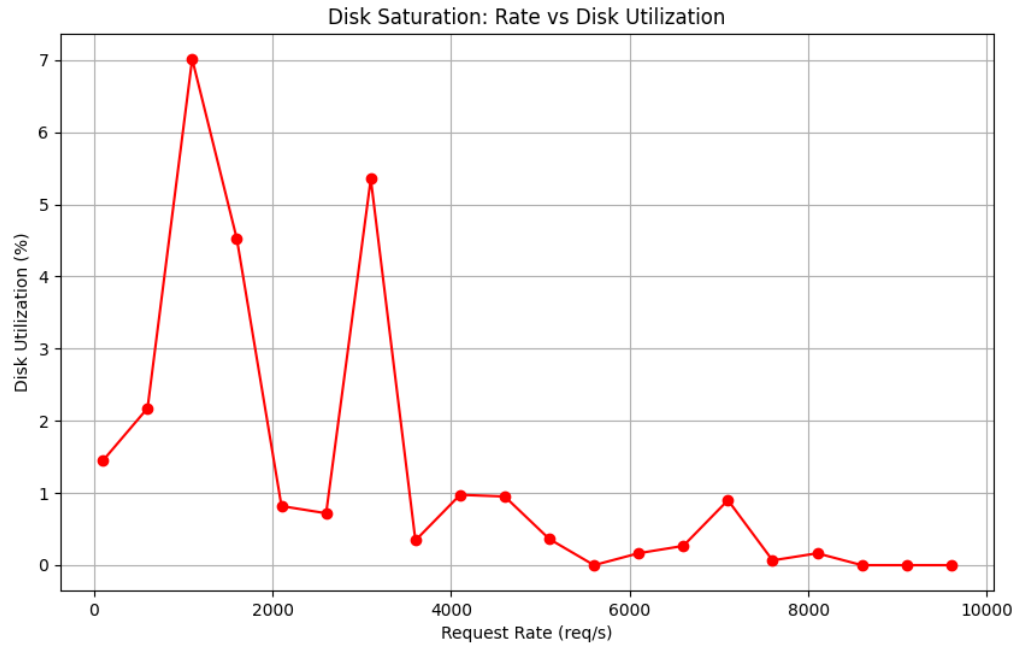

Figure 72: Rate vs Disk Utilization

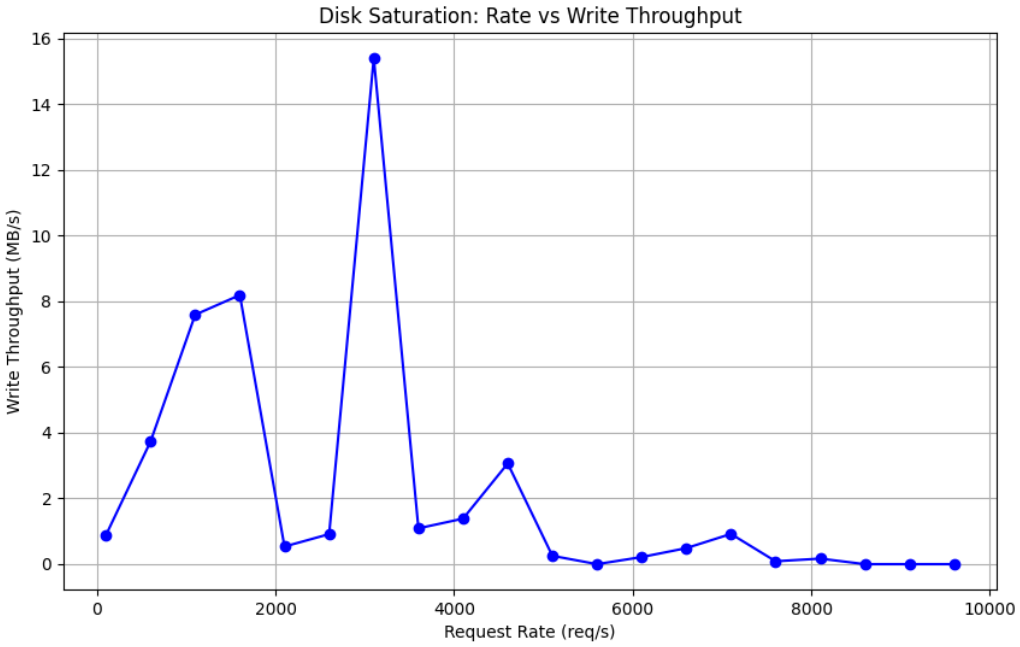

Figure 73: Rate vs Write Throughput (MB/s)

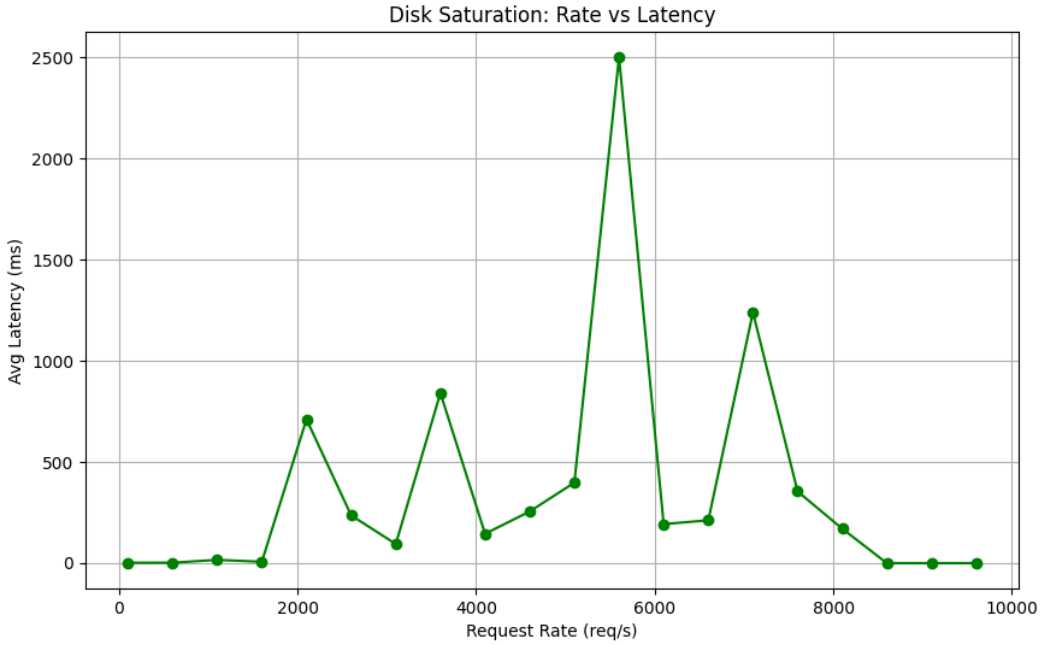

Figure 74: Rate vs Latency

9 Credits

Third-party libraries: nlohmann/json and cpp-httplib. Licensed under MIT (see project LICENSE).