TECHNISCHE UNIVERSITÄT BERLIN

Scalability Engineering Prototyping Assignment (Summer Semester 2025)

Szymon Szendzielorz, Zacharias Athanasiadis

Project Overview

A highly scalable task computation API demonstrating load balancing, rate limiting, caching, and distributed task processing.

It features a load balancer, multiple backend API instances, worker processes, and a shared cache using the database.

Built with **FastAPI**, **PostgreSQL** and **Docker**.

Requirements

1. The application must manage some kind of state.

2. The application needs to be able to scale vertically and horizontally.

3. When the application is scaled up/out, it should not be possible to overload another component (i.e., there needs to be some mitigation strategy for very high load).

4. The application need to implement two more strategies, which were covered either in the lecture content or during the presentations, in addition to the strategies already implemented for requirements 1 to 3.

How our app fulfills the requirements

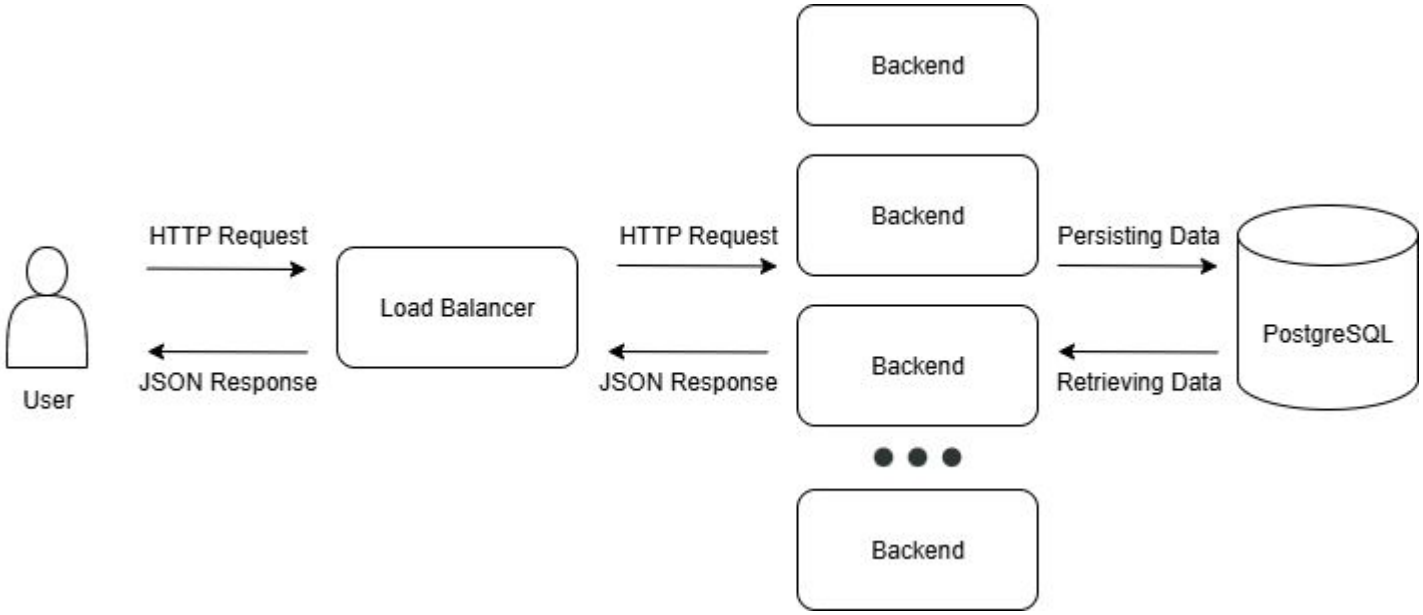
1. The application manages state by storing tasks and worker details, and cache entries in a PostgreSQL database.

2. The system can scale vertically and horizontally by running multiple backend API instances and worker processes using **Docker Compose**.

3. Overload is prevented by implementing **rate limiting** in the load balancer and **connection limiting** in database access layer, ensuring no component is overwhelmed under high load.

4. Additional strategies implemented include **load balancing** (round-robin and least-connections) and **caching.**

System Architecture



Task Workflow

1. **Submit a Task**: User sends a POST request to one of the task endpoints (currently /hash/md5, /hash/sha256, or /hash/argon2) with the data to be computed.

2. **Queueing**: The request is added to the database-backed task queue.

3. **Processing**: Worker processes poll the queue, perform the requested computation, and store the result in the shared cache and database.

4. **Retrieval**: User can check the status and result of the task via /task/{task_id}. Cached results are returned instantly if available.

Supported Algorithms

So far the system supports three types of tasks: MD5, SHA256, and Argon2. These tasks allow users to submit data (typically strings or passwords) to be securely hashed by the backend workers.

- **MD5:** A fast, legacy hash function. Suitable for checksums but not recommended for password storage due to vulnerabilities.
- **SHA256:** A secure cryptographic hash function from the SHA-2 family. Commonly used for data integrity and digital signatures.
- **Argon2:** A modern, memory-hard password hashing algorithm designed to resist GPU cracking attacks. Recommended for password storage.

Key Features

- ❏ Load balancing (round-robin, least-connections)
- ❏ Asynchronous task queue
- ❏ Worker scaling
- ❏ Shared cache for results
- ❏ Connection limiter to prevent databse overload
- ❏ Health checks and statistics endpoints
- ❏ Rate limiting and strategy switching via API

Load Balancing & Rate Limiting

- The **load balancer** applies the selected strategy to traffic the incoming request to the appropriate backend instances. Users can select either round-robin or least-connections as a strategy to configure the load balancer.

- **Rate limiting** is applied when a specific user (IP) makes many subsequent requests in a time window.

Connection Limiting

All the backend instances access one database. To prevent overloading the database we setup **a Connection limit** when initializing the database.

- Each instance utilizes up to 2 connections, 1 persistent connection for *TaskQueue* and 1 temporary for *SharedCache*.
- If the connection limit is reached we utilize **backoff** and **jitter** for the connection retires
- If after given given number of retries the database connection wasn’t enabled we return the information with error reason (Suggestion to try later)

Caching

Results of tasks are cached in the PostgreSQL database for fast retrieval.

When a user requests a task computation for data that has already been processed, the cached result is returned instantly from the shared cache that **all instances** can access.

Database Design

Database tables:

- **tasks**
  (id, task_type, parameters, status, created_at, started_at, completed_at, result, error)

- **workers**
  (id, worker_id, status, last_heartbeat, current_task_id)

- **cache entries**
  (cache_key, value_data, expires_at, created_at, access_count, last_accessed)

API Endpoints (Examples)

- **POST /hash/md5, /hash/sha256, /hash/argon2** — submit data for processing
- **GET /task/{task_id}** — check status/result
- **GET /health** — check backend health
- **GET /cache/stats** — get cache statistics
- Load balancer endpoints: **/lb/health, /lb/stats, /lb/rate-limits, /lb/strategy**

Setup & Configuration

Clone repo, configure .env file

Key .env settings:

- Database Configuration
- Application Configuration
- Load Balancer Configuration

Start with **docker compose up --build**

```
# PostgreSQL Superuser (for initial setup)
POSTGRES_DB=postgres
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres123

# Custom Database and User
DB_NAME=task_queue_db
DB_USER=db_user
DB_PASSWORD=db_password
DB_HOST=postgres
DB_PORT=5432

DB_CONNECTION_RETRIES=5
DB_BASE_BACKOFF=0.5
DB_USER_CONNECTION_LIMIT=4

# Application Configuration
APP_NAME=queue-api
WORKER_NAME=worker

# Load Balancer Configuration
LB_STRATEGY=round_robin
RATE_LIMIT_REQUESTS=100
RATE_LIMIT_WINDOW=60
```

Example .env configuration

Request Workflow (Not Cached)

```
loadbalancer-1  | Incoming POST request from 172.18.0.1 for /hash/md5 routed to http://app2:8000 (strategy: round_robin)
app2-1          | [queue-api-2] Received MD5 request for string: test_value
app2-1          | [queue-api-2] Checking cache for MD5: test_value
app2-1          | [queue-api-2] Getting cache key: 93ef37dba716c3a85f2417274019144e
app2-1          | Connected to database successfully on attempt 1
app2-1          | [queue-api-2] Cache miss for key: 93ef37dba716c3a85f2417274019144e
app2-1          | [queue-api-2] Cache miss, adding task to queue
app2-1          | [queue-api-2] add_task() called for MD5: test_value
loadbalancer-1  | HTTP Request: POST http://app2:8000/hash/md5 "HTTP/1.1 200 OK"
app2-1          | [queue-api-2] Attempting to get database connection...
loadbalancer-1  | Response 200 from http://app2:8000 for POST /hash/md5
app2-1          | Connected to database successfully on attempt 1
loadbalancer-1  | INFO:     172.18.0.1:37964 - "POST /hash/md5 HTTP/1.1" 200 OK
app2-1          | [queue-api-2] Database connection attempt result: success=True
app2-1          | [queue-api-2] Database connection successful, entering context
app2-1          | [queue-api-2] Inside database context, adding task to queue for MD5: test_value
app2-1          | [QueueManager] Adding task to queue: md5 with parameters: {'string': 'test_value'}
app2-1          | Task added with ID: 2
app2-1          | [queue-api-2] Task ID: 2
app2-1          | [queue-api-2] MD5 task 2 queued for string: test_value
app2-1          | [queue-api-2] Closing database connection
app2-1          | Database connection closed.
app2-1          | INFO:     172.18.0.9:59234 - "POST /hash/md5 HTTP/1.1" 200 OK
worker1-1       | [worker-1] *** PICKED UP TASK 2 (type: md5) ***
worker1-1       | [worker-1] Processing task 2 with parameters: {'string': 'test_value'}
worker1-1       | [cache] Setting cache key: 93ef37dba716c3a85f2417274019144e with TTL: 3600s
worker1-1       | Connected to database successfully on attempt 1
worker1-1       | [cache] Successfully set cache key: 93ef37dba716c3a85f2417274019144e
worker1-1       | Task 2 marked as completed
worker1-1       | [worker-1] *** COMPLETED TASK 2 *** (Total processed: 2)
```

```
$ curl -X POST "http://localhost:8000/hash/md5" -H "Content-Type: application/json" -d '{"string": "test_value"}'
```

Request Workflow (Cached)

$ curl -X POST "http://localhost:8000/hash/md5" -H "Content-Type: application/json" -d '{"string": "test_value"}'

```
loadbalancer-1  | Incoming POST request from 172.18.0.1 for /hash/md5 routed to http://app3:8000 (strategy: round_robin)
app3-1          | [queue-api-3] Received MD5 request for string: test_value
app3-1          | [queue-api-3] Checking cache for MD5: test_value
app3-1          | [queue-api-3] Getting cache key: 93ef37dba716c3a85f2417274019144e
app3-1          | Connected to database successfully on attempt 1
app3-1          | [queue-api-3] Cache hit for key: 93ef37dba716c3a85f2417274019144e
loadbalancer-1  | HTTP Request: POST http://app3:8000/hash/md5 "HTTP/1.1 200 OK"
app3-1          | [queue-api-3] MD5 result found in cache for string: test_value
loadbalancer-1  | Response 200 from http://app3:8000 for POST /hash/md5
app3-1          | [queue-api-3] Cache hit, returning cached result
```

```
❯ curl -X POST "http://localhost:8000/hash/md5" -H "Content-Type: application/json" -d '{"string": "test_value"}'
{"result":{"original_string":"test_value","md5_hash":"8dd066a9072cfaca57bcedd7f233432f","execution_time_seconds":1.7881393432617188e-05},"source":"cache"}
```

Request Workflow (Db connection limit exceeded)

$ curl -X POST "http://localhost:8000/hash/md5" -H "Content-Type: application/json" -d '{"string": "test_value"}'

```
app3-1          | [queue-api-3] Attempting to get database connection...
app3-1          | Error while connecting to PostgreSQL: connection failed: connection to server at "172.18.0.2", port 5432 failed: FATAL:  too many connections for role "db_user"
app3-1          | Connection limit reached. Attempt 1/5
app3-1          | Database connection limit reached. Backing off for 1.45 seconds... (attempt 1)
postgres_db     | 2025-07-14 22:54:59.834 UTC [412] FATAL:  too many connections for role "db_user"
app3-1          | Error while connecting to PostgreSQL: connection failed: connection to server at "172.18.0.2", port 5432 failed: FATAL:  too many connections for role "db_user"
app3-1          | Connection limit reached. Attempt 2/5
app3-1          | Database connection limit reached. Backing off for 1.17 seconds... (attempt 2)
postgres_db     | 2025-07-14 22:55:01.010 UTC [413] FATAL:  too many connections for role "db_user"
app3-1          | Error while connecting to PostgreSQL: connection failed: connection to server at "172.18.0.2", port 5432 failed: FATAL:  too many connections for role "db_user"
app3-1          | Connection limit reached. Attempt 3/5
app3-1          | Database connection limit reached. Backing off for 2.09 seconds... (attempt 3)
postgres_db     | 2025-07-14 22:55:03.115 UTC [414] FATAL:  too many connections for role "db_user"
app3-1          | Error while connecting to PostgreSQL: connection failed: connection to server at "172.18.0.2", port 5432 failed: FATAL:  too many connections for role "db_user"
app3-1          | Connection limit reached. Attempt 4/5
app3-1          | Database connection limit reached. Backing off for 4.14 seconds... (attempt 4)
postgres_db     | 2025-07-14 22:55:07.269 UTC [423] FATAL:  too many connections for role "db_user"
app3-1          | Error while connecting to PostgreSQL: connection failed: connection to server at "172.18.0.2", port 5432 failed: FATAL:  too many connections for role "db_user"
loadbalancer-1  | HTTP Request: POST http://app3:8000/hash/md5 "HTTP/1.1 500 Internal Server Error"
app3-1          | Connection limit reached. Attempt 5/5
loadbalancer-1  | Response 500 from http://app3:8000 for POST /hash/md5
```

Request Workflow (Rate limit exceeded)

$ curl -X POST "http://localhost:8000/hash/md5" -H "Content-Type: application/json" -d '{"string": "test_value"}'

```
) curl -X POST "http://localhost:8000/hash/md5" -H "Content-Type: application/json" -d '{"string": "example_value"}'
Rate limit exceeded. Maximum 30 requests per 60 seconds.%
```

Benchmark

Test suite: **Locust**

- **Requests per second**: 1
- **Request window:** 60 seconds
- **Request limit:** 30

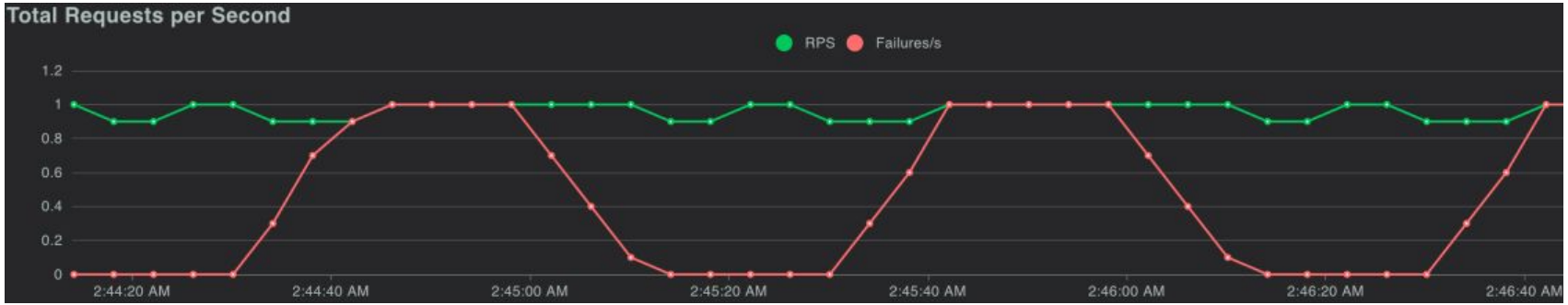


Figure: **Locust** testing graph