FaaSLight: A Serverless Computing Simulation Framework

May 23, 2025

1 Introduction

FaaSLight is a Python-based framework designed to simulate and evaluate serverless computing models. It compares five platforms—FaaSLight Original, FaaSLight Enhanced, SAND++, AWS Fargate, and Unikernel—across compute-intensive, I/O-bound, and memory-intensive workloads. The project measures key performance metrics, including latency, CPU/memory usage, security, cost, and a composite score, to demonstrate the efficacy of the optimized FaaSLight Enhanced model. Using Docker containers, Flask APIs, and Plotly visualizations, FaaSLight provides a portable and scalable environment for performance analysis.

This document details the project's objectives, architecture, implementation, and results, targeting an academic audience evaluating serverless computing frameworks.

2 Objectives

The primary objectives of FaaSLight are:

- To simulate realistic serverless workloads (compute, I/O, memory) across five platforms.
- To quantify performance metrics: average and P95 latency, CPU/memory usage, security score, cost, and composite score.
- To demonstrate the superiority of FaaSLight Enhanced, achieving lower latencies (e.g., $\sim$150ms compute, $\sim$4.20 I/O composite score with two replicas).
- To provide interactive visualizations for performance comparison.

3 Architecture

The FaaSLight architecture leverages Docker for containerization, Flask for API endpoints, and a Python simulator for workload generation and analysis:

- **Docker Containers**: Five services (`faaslight_original`, `faaslight_enhanced`, `sand_plus`, `fargate_inspired`, `unikernel_inspired`) are defined in `docker-compose.yml`, built from a `Dockerfile` using Python 3.9-slim.
- **Flask APIs**: The `app.py` script hosts a Flask server in each container, exposing endpoints (`/compute`, `/io`, `/memory`, `/verify`) to simulate serverless workloads.

- **Simulator**: The `faaslight_simulator.py` script sends 200 HTTP requests per service, collects metrics, and generates visualizations using Plotly.
- **Network**: A bridge network (`faaslight-net`) maps container port 5000 to localhost ports 32768–32772.
- **Output**: Results are stored in `results/metrics.csv` (raw metrics) and `results/dashboard.ht` (interactive plots).

4 Implementation

4.1 Docker Configuration

The `docker-compose.yml` defines five services with tailored resource limits:

- `faaslight_enhanced`: 0.75 CPU, 768MB memory, port 32769.
- `faaslight_original`: 0.5 CPU, 512MB memory, port 32768.
- `sand_plus`: 0.5 CPU, 512MB memory, port 32770.
- `fargate_inspired`: 0.6 CPU, 640MB memory, port 32771.
- `unikernel_inspired`: 0.4 CPU, 384MB memory, port 32772.

Security features include `no-new-privileges`, capability dropping (`cap_drop: ALL`), and minimal capabilities (`cap_add: NET_BIND_SERVICE`).

4.2 Flask Application

The `app.py` script implements four endpoints:

- `/compute`: Simulates CPU-intensive tasks via NumPy matrix multiplication (default size: 500x500).
- `/io`: Simulates I/O-bound tasks with a random delay (10–50ms).
- `/memory`: Simulates memory-intensive tasks by sorting a large list (default size: 1,000,000).
- `/verify`: Performs a security check using SHA-256 hashing, returning a 200 OK response.

4.3 Simulator Logic

The `faaslight_simulator.py` script orchestrates the simulation:

- `send_request(url, func_type, is_cold)`: Sends HTTP GET requests, applies cold-start penalties (e.g., 10ms for FaaSLight Enhanced), and measures latency, CPU/memory usage, and security overhead.
- `run_simulation()`: Executes 200 requests per service using `ThreadPoolExecutor` (10 workers), with 10% cold starts.

- `compute_metrics(metrics)`: Computes average/P95 latency, CPU/memory usage, security score, cost, and composite score:

$$\text{Cost} = (\text{avg\_cpu} + \text{avg\_memory}) \times \frac{\text{avg\_latency}}{1000} \times (0.8 \text{ if FaaSLight Enhanced, else } 1)$$

$$\text{Composite} = \frac{\text{security} \times 1000}{\text{avg\_latency} + \text{p95\_latency} + \text{cost}}$$

- `plot_results(results)`: Generates a bar chart (composite scores) and scatter plot (latency vs. security) using Plotly, saved to `results/dashboard.html`.

4.4 Dependencies

Dependencies are listed in `requirements.txt`:

- `flask==2.3.3`: Web server for APIs.
- `numpy==1.26.4`: Matrix operations for compute tasks.
- `plotly==5.24.1`: Interactive visualizations.
- `requests==2.32.3`: HTTP requests for simulation.
- `psutil==6.0.0`: System resource monitoring.

5 Execution Flow

1. **Setup**: Clone repository, set up Python virtual environment, install dependencies, and start Docker Desktop.
2. **Start Containers**: Run `docker-compose up -d -build` to launch services.
3. **Run Simulation**: Execute `python faaslight_simulator.py`, which:
   - Sends 200 requests per service, with 20 cold starts.
   - Collects metrics (latency, CPU/memory, security).
   - Computes performance indicators (average/P95 latency, cost, composite score).
   - Generates visualizations in `results/dashboard.html`.
4. **View Results**: Open `results/dashboard.html` for interactive plots or inspect `results/metrics.csv`.
5. **Clean Up**: Run `docker-compose down` and deactivate the virtual environment.

6 Results

The simulation results highlight FaaSLight Enhanced's performance:

- **Single Replica**:
  - FaaSLight Enhanced: $\sim$749ms (compute), $\sim$564ms (I/O), composite $\sim$0.39–0.49.
  - Unikernel: $\sim$1760ms (compute), composite $\sim$0.13.