CI/CD Speed Simulator - Cloud Computing Project Documentation

Project Overview

The CI/CD Speed Simulator is a performance analysis tool designed to evaluate different strategies in build phases, load balancing, and scheduling algorithms that directly impact the efficiency of cloud-based CI/CD pipelines.

Why this project matters:
- Faster deployments
- Better utilization of compute resources
- Reduced waiting and turnaround times
- Fairer load distribution across servers

This simulator measures key performance metrics such as Total Time & Speedup, Efficiency, Load Imbalance, Fairness Index, and Scheduling Times.

Technology Stack

- Python 3.12.3
- Pandas, Matplotlib
- Docker, docker-compose
- Git for version control

Features

1. Build Phase Simulation (Sequential, Parallel, Cached, Slim Image)
2. Load Balancing Simulation (Round Robin, Least Connections, Random, Genetic Algorithm)
3. Scheduling Algorithms Simulation (FCFS, SJF, SRTF, HRRN)
4. CSV Data Logging (logs/results.csv)
5. Graphical Analysis (Matplotlib plots for clear visualization)

Methodology

1. Build Phase: Simulates container builds in CI/CD pipelines.
2. Load Balancing Phase: Distributes cloud traffic with multiple strategies.
3. Scheduling Phase: Simulates CPU scheduling in cloud VMs and containers.

Graphs

(1) Build Phase: build_total_time.png, build_speedup.png, build_efficiency.png
(2) Load Balancing Phase: load_avg_load.png, load_variance.png, load_fairness.png, load_imbalance.png
(3) Scheduling Phase: scheduling_times.png, scheduling_combined.png

Results and Analysis

- Build Phase: Parallel builds show massive speedup, Slim builds provide consistent efficiency, Cached builds help incremental cases.
- Load Balancing: Round Robin and Least Connections ensure fairness; Genetic Algorithm helps in dynamic workloads.(experiments are done upto 20000 service instances till now)
- Scheduling: SJF/SRTF minimize turnaround; HRRN balances fairness; FCFS baseline but inefficient.(experiments are done upto 10000 jobs till now)

How to Run

1. Running Locally(currently):
   - Open terminal
   - Run: python3 -m simulator.build_simulator
   - Then: python3 -m simulator.plot_results

2. Running with Docker(optional but then handle .dockerignore and .gitigonre carefuuly,if needed then update these two files):
   - Run: docker-compose up --build
   - Use docker logs to monitor execution

3. Data and Logs:
   - Results stored in logs/results.csv
   - Graphs in ./graphs/

Current .gitignore

Python

__pycache__/

*.pyc

*.pyo

*.pyd

*.egg-info/

*.egg

Virtual environment

.venv/

env/

venv/

Logs and results

logs/*.log

logs/*.txt

results.csv

IDE / Editor files

.vscode/

.idea/

System files

.DS_Store

Thumbs.db

Current .dockerignore

Ignore Python cache

__pycache__/

```
*.pyc
*.pyo
*.pyd
# Ignore venv
.venv/
env/
venv/
# Ignore git and docker files
.git
.gitignore
.dockerignore
# Ignore test and logs
tests/
logs/
*.log
*.txt
results.csv
```

Future Work

- Cloud Deployment on AWS/GCP/Azure
- Web Dashboard (Flask/Django/React)
- Larger workload scalability tests
- CI/CD tool integration (Jenkins, Kubernetes)

Energy Consumption

This simulator also allows analysis of energy efficiency in CI/CD pipelines. By comparing build strategies and scheduling algorithms, we can estimate power savings:

- Parallel builds reduce idle time of servers, lowering wasted energy.

- Load balancing prevents hotspots, optimizing cooling and server utilization.
- Scheduling strategies like SJF/SRTF reduce waiting processes, improving CPU efficiency.

Future work can integrate energy models (watts per core, per VM) to derive realistic energy cost comparisons.

Proposal

This work can evolve into a research publication. The proposal can focus on:
- Improving cloud-native CI/CD performance
- Reducing deployment energy costs
- Benchmarking scheduling fairness
- Offering an open-source reproducible simulator