# Project report on Internship done during 06th August,2025 to 08th October,2025

Name :    Aditya Banerjee

University Name: University Of Kalyani

Department Name: Department of Computer Science and Enfineering

Roll No.:90/BCS/220007

Registration No.:1090122 of 2022-2023

Session:2025-2026

Stream:B.Tech (CSE) 7th Semester

Project:  Design of a Novel CI/CD Simulator for Cloud Computing environment

Guide Name: Dr. Kousik Dasgupta

Associate Professor and H.O.D.

Department of Computer Science and Engineering

Kalyani Government Engineering College,Kalyani,Nadia

# Index

# Design of a Novel CI/CD Simulator for Cloud Computing environment

## A) Problem Statement:

Continuous Integration (CI) and Continuous Deployment (CD) pipelines are critical for modern software development, enabling rapid and reliable delivery of software updates. However, in cloud environments, inefficiencies in build strategies, load balancing, and scheduling can lead to:

- **Longer deployment times**, delaying product releases
- **Underutilization of cloud resources**, causing wasted computational power
- **Unbalanced load distribution**, which can overload some servers while leaving others idle
- **Inefficient scheduling of tasks**, increasing average waiting and turnaround times
- **Higher energy consumption**, due to idle resources or inefficient task execution

These challenges affect **both performance and operational costs** in cloud-native CI/CD pipelines. There is a need for a **simulator** that can evaluate

different strategies to optimize build times, distribute loads efficiently, and schedule tasks effectively.

## B) Introduction

Cloud computing has revolutionized software deployment, providing **elastic infrastructure** and **scalable resources**. CI/CD pipelines are widely used to automate software builds, testing, and deployment. A typical CI/CD process involves:

1. **Build Phase** – Compiling code, packaging containers, and creating deployable artifacts
2. **Load Balancing Phase** – Distributing incoming requests or tasks across multiple servers
3. **Scheduling Phase** – Allocating CPU resources to jobs or processes in an optimal way

Optimizing these phases improves **deployment speed**, **resource utilization**, and **energy efficiency**.

**Importance of the Project**

- Provides **quantitative metrics** to compare strategies
- Helps in **making informed decisions** for cloud-based CI/CD pipelines
- Enables **experimentation with large-scale workloads** (up to 20000 services and 10000 jobs)
- Can be extended to **energy-efficient deployments** and cloud cost optimization

## C) Literature Survey:

Several studies highlight the need for efficient CI/CD in cloud environments:

| Author | Year | Focus | Findings |
|---|---|---|---|
| Humble | 20 | CI/CD | Emphasized automation and |

| Author | Year | Focus | Findings |
|---|---|---|---|
| & Farley | 10 | principles | rapid feedback loops |
| Sharma et al. | 2018 | Cloud CI/CD scaling | Load balancing critical to reduce bottlenecks in deployments |
| Li et al. | 2020 | Task scheduling in cloud | SJF and SRTF scheduling minimized average turnaround time |
| Gupta & Kumar | 2022 | Energy-efficient cloud | Parallel builds reduce idle server energy consumption |

**Gap Identified**: Existing studies often focus on **single aspects** like build optimization or scheduling but **lack an integrated simulator** to evaluate all phases together under large-scale workloads.

## D) Proposed Methodology:

The simulator evaluates three main phases using Python:

### 1. Build Phase

Four build strategies are implemented:

| Strategy | Formula / Concept | Notes |
|---|---|---|
| Sequential Build | `total_time = num_services * avg_time` | Baseline, processes services one after another |
| Parallel Build | `total_time = avg_time` `speedup = seq_time / total_time` `efficiency = speedup / num_services` | Executes all services concurrently |
| Cached Build | `total_time = changed_services * avg_time + (num_services - changed_services) * (avg_time // 2)` | Reuses previous builds for unchanged services |
| Slim Image Build | `total_time = int(seq_time * slimming_factor)` | Creates smaller images for faster deployment |

Metrics Collected: Total Build Time, Speedup, Efficiency

**2. Load Balancing Phase**

Four algorithms:

| **Algorithm** | **Methodology** |
|---|---|
| Round Robin | Requests distributed cyclically among servers |
| Least Connections | Requests sent to server with minimum active connections |
| Random | Requests assigned randomly to any server |
| Genetic Algorithm | Optimizes request distribution using evolutionary selection and mutation |

Metrics Collected:

- Average Load
- Max Load
- Min Load
- Load Variance
- Fairness Index
- Load Imbalance

**Formulas**:

- **Average Load**: $\bar{x}=\sum xi/n$,
- **Variance**: $\sigma^2=n\sum(xi-\bar{x})^2$,
- **Fairness Index**: $FI=(\sum xi)^2/(n*(\sum xi)^2)$,
- **Load Imbalance**: $LI=\max(xi)-\min(xi)$

**3. Scheduling Phase:**

Simulates CPU scheduling of cloud jobs:

| **Algorithm** | **Concept** | **Notes** |
|---|---|---|
| FCFS | First Come First Serve | Baseline, processes jobs in arrival order |
| SJF | Shortest Job First | Non-preemptive, minimizes average turnaround time |
| SRTF | Shortest Remaining Time First | Preemptive SJF, adapts dynamically |
| HRRN | Highest Response Ratio Next | Balances fairness and efficiency |

Metrics Collected: Average Waiting Time, Average Turnaround Time, Average Response Time

Formulas:

- **Turnaround Time:** $TAT_i = CT_i - AT_i$
- **Waiting Time:** $Wt_i = TAT_i - BT_i$
- **Response Time:** $Rt_i = ST_i - AT_i$

## E) Results & Analysis

### 1. Technology Stack:

| Component | Version / Tool | Purpose |
|---|---|---|
| Python | 3.12.3 | Simulation logic and CSV management |
| Pandas | 2.1.0 | Data handling for results |
| Matplotlib | 3.8.0 | Graphical visualization |
| Docker | 24.0 | Containerized execution |
| Git | 2.42 | Version control |

## 2.Results:

## Scheduling Phase(Input data):

| Set Number | Number of inputs |
|---|---|
| 1 | 15 |
| 2 | 40 |
| 3 | 100 |
| 4 | 200 |
| 5 | 2000 |
| 6 | 4000 |
| 7 | 6000 |
| 8 | 8000 |
| 9 | 10000 |

## Build Phase(Input Data):

| Set Number | Number of Services | Avg build time per services(minutes) |
|---|---|---|
| 1 | 2500 | 6000 |
| 2 | 5000 | 8000 |
| 3 | 400 | 12000 |
| 4 | 50 | 20000 |

## Load Balancing Phase(Input Data):

| Set Number | Number of Service Instances | Total Incoming Requests |
|---|---|---|
| 1 | 2000 | 1340 |
| 2 | 8000 | 6240 |
| 3 | 12000 | 9600 |
| 4 | 15000 | 7500 |
| 5 | 20000 | 15000 |

**Scheduling Phase(Output data):**

| Set Number | Number of Inputs | Algorithm | avg_waiting | avg_turnaround | avg_response |
|---|---|---|---|---|---|
| 1 | 15 | FCFS | 8.13 | 11.20 | 8.13 |
| 1 | 15 | SJF | 6.13 | 9.20 | 6.13 |
| 1 | 15 | SRTF | 6.07 | 9.13 | 6.00 |

| Set Number | Number of Inputs | Algorithm | avg_waiting | avg_turnaround | avg_response |
|---|---|---|---|---|---|
| 1 | 15 | HRRN | 7.13 | 10.20 | 7.13 |
| 2 | 40 | FCFS | 165.12 | 174.07 | 165.12 |
| 2 | 40 | SJF | 96.78 | 105.72 | 96.78 |
| 2 | 40 | SRTF | 96.53 | 105.47 | 96.28 |
| 2 | 40 | HRRN | 104.65 | 113.60 | 104.65 |
| 3 | 100 | FCFS | 194.09 | 200.11 | 194.09 |
| 3 | 100 | SJF | 152. | 158.44 | 152.4 |

| Set Number | Number of Inputs | Algorithm | avg_waiting | avg_turnaround | avg_response |
|---|---|---|---|---|---|
| | | | 42 | | 2 |
| 3 | 100 | SRTF | 152.34 | 158.36 | 152.27 |
| 3 | 100 | HRRN | 161.32 | 167.34 | 161.32 |
| 4 | 200 | FCFS | 12772.44 | 12949.81 | 12772.44 |
| 4 | 200 | SJF | 12095.06 | 12272.42 | 12095.06 |
| 4 | 200 | SRTF | 12095.06 | 12272.42 | 12095.06 |
| 4 | 200 | HRRN | 12095.80 | 12273.16 | 12095.80 |

| Set Number | Number of Inputs | Algorithm | avg_waiting | avg_turnaround | avg_response |
|---|---|---|---|---|---|
| 5 | 2000 | FCFS | 9942.28 | 9952.78 | 9942.28 |
| 5 | 2000 | SJF | 6715.68 | 6726.24 | 6715.68 |
| 5 | 2000 | SRTF | 6909.62 | 6920.31 | 6906.78 |
| 5 | 2000 | HRRN | 6738.98 | 6749.52 | 6738.98 |
| 6 | 4000 | FCFS | 20831.93 | 20842.57 | 20831.93 |
| 6 | 4000 | SJF | 14103.47 | 14114.09 | 14103.47 |
| 6 | 4000 | SRT | 1363 | 13644. | 13630 |

| Set Number | Number of Inputs | Algorithm | avg_waiting | avg_turnaround | avg_response |
|---|---|---|---|---|---|
| | | F | 4.35 | 71 | .91 |
| 6 | 4000 | HRRN | 13814.51 | 13824.98 | 13814.51 |
| 7 | 6000 | FCFS | 31468.68 | 31479.25 | 31468.68 |
| 7 | 6000 | SJF | 21040.88 | 21051.42 | 21040.88 |
| 7 | 6000 | SRTF | 20873.00 | 20883.46 | 20872.98 |
| 7 | 6000 | HRRN | 20940.16 | 20950.61 | 20940.16 |
| 8 | 8000 | FCFS | 41115.05 | 41125.53 | 41115.05 |

| Set Number | Number of Inputs | Algorithm | avg_waiting | avg_turnaround | avg_response |
|---|---|---|---|---|---|
| 8 | 8000 | SJF | 28334.07 | 28344.61 | 28334.07 |
| 8 | 8000 | SRTF | 28671.90 | 28682.51 | 28671.89 |
| 8 | 8000 | HRRN | 27870.12 | 27880.56 | 27870.12 |
| 9 | 10000 | FCFS | 52346.98 | 52357.51 | 52346.98 |
| 9 | 10000 | SJF | 35610.50 | 35621.03 | 35610.50 |
| 9 | 10000 | SRTF | 35260.18 | 35270.63 | 35259.95 |
| 9 | 10000 | HRR | 3509 | 35108. | 35097 |

| Set Number | Number of Inputs | Algorithm | avg_waiting | avg_turnaround | avg_response |
|---|---|---|---|---|---|
| | | N | 7.79 | 23 | .79 |

## Build Phase(Output Data):

| Set Number | Number of Services | Avg build time per services(minutes) | strategy | total_time | speedup | efficiency |
|---|---|---|---|---|---|---|
| 1 | 2500 | 6000 | Sequential Build | 15000000.0 | 1.00 | 1.00 |
| 1 | 2500 | 6000 | Parallel Build | 2500.0 | 6000.00 | 1.00 |
| 1 | 2500 | 6000 | Cached Build | 9500000.0 | 1.58 | 1.58 |
| 1 | 2500 | 6000 | Slim Imag | 10500000 | 1.43 | 1.43 |

| Set Number | Number of Services | Avg build time per services(minutes) | strategy | total_time | speedup | efficiency |
|---|---|---|---|---|---|---|
| | | | e Build | .0 | | |
| 2 | 5000 | 8000 | Sequential Build | 40000000.0 | 1.00 | 1.00 |
| 2 | 5000 | 8000 | Parallel Build | 5000.0 | 8000.00 | 1.00 |
| 2 | 5000 | 8000 | Cached Build | 27500000.0 | 1.45 | 1.45 |

| Set Number | Number of Services | Avg build time per services(minutes) | strategy | total_time | speedup | efficiency |
|---|---|---|---|---|---|---|
| 2 | 5000 | 8000 | Slim Image Build | 28000000.0 | 1.43 | 1.43 |
| 3 | 400 | 12000 | Sequential Build | 4800000.0 | 1.00 | 1.00 |
| 3 | 400 | 12000 | Parallel Build | 400.0 | 12000.00 | 1.00 |
| 3 | 400 | 12000 | Cach | 4000 | 1.2 | 1.20 |

| Set Number | Number of Services | Avg build time per services(minutes) | strategy | total_time | speedup | efficiency |
|---|---|---|---|---|---|---|
| | | | ed Build | 000.0 | 0 | |
| 3 | 400 | 12000 | Slim Image Build | 3360000.0 | 1.43 | 1.43 |
| 4 | 50 | 20000 | Sequential Build | 1000000.0 | 1.00 | 1.00 |
| 4 | 50 | 20000 | Parallel | 50.0 | 20000. | 1.00 |

| Set Number | Number of Services | Avg build time per services(minutes) | strategy | total_time | speedup | efficiency |
|---|---|---|---|---|---|---|
| | | | Build | | 00 | |
| 4 | 50 | 20000 | Cached Build | 875000.0 | 1.14 | 1.14 |
| 4 | 50 | 20000 | Slim Image Build | 700000.0 | 1.43 | 1.43 |

## Load Balancing Phase(Output Data):

| Set Number | Number of Service Instances | Total Incoming Requests | algorithm | avg_load | max_load | min_load | variance | fairness_index | load_imbalance |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2000 | 1340 | Round Robin | 0.67 | 1.0 | 0.0 | 0.22 | 0.6667 | 1.0 |
| 1 | 2000 | 1340 | Least Connections | 0.67 | 1.0 | 0.0 | 0.22 | 0.6667 | 1.0 |
| 1 | 200 | 134 | Rand | 0 | 6 | 0 | 0.6 | 0.4 | 6.0 |

| Set Number | Number of Service Instances | Total Incoming Requests | algorithm | avg_load | max_load | min_load | variance | fairness_index | load_imbalance |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | 0 | 0 | om | .67 | .0 | .0 | 6 | 017 | |
| 1 | 2000 | 1340 | Genetic Algorithm (Load Balan | 0.67 | 5.0 | 0.0 | 0.63 | 0.4119 | 5.0 |

| Set Number | Number of Service Instances | Total Incoming Requests | algorithm | avg_load | max_load | min_load | variance | fairness_index | load_imbalance |
|---|---|---|---|---|---|---|---|---|---|
| | | | cing) | | | | | | |
| 2 | 8000 | 6240 | Round Robin | 0.78 | 1.0 | 0.0 | 0.17 | 0.7778 | 1.0 |
| 2 | 8000 | 6240 | Least Connections | 0.78 | 1.0 | 0.0 | 0.17 | 0.7778 | 1.0 |

| Set Number | Number of Service Instances | Total Incoming Requests | algorithm | avg_load | max_load | min_load | variance | fairness_index | load_imbalance |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 8000 | 6240 | Random | 0.78 | 6.0 | 0.0 | 0.77 | 0.4403 | 6.0 |
| 2 | 8000 | 6240 | Genetic Algorithm (Load | 0.78 | 6.0 | 0.0 | 0.75 | 0.4471 | 6.0 |

| Set Number | Number of Service Instances | Total Incoming Requests | algorithm | avg_load | max_load | min_load | variance | fairness_index | load_imbalance |
|---|---|---|---|---|---|---|---|---|---|
| | | | Balancing) | | | | | | |
| 3 | 12000 | 9600 | Round Robin | 0.80 | 1.0 | 0.0 | 0.16 | 0.8000 | 1.0 |
| 3 | 12000 | 9600 | Least Connectio | 0.8 | 1.0 | 0.0 | 0.16 | 0.8000 | 1.0 |

| Set Number | Number of Service Instances | Total Incoming Requests | algorithm | avg_load | max_load | min_load | variance | fairness_index | load_imbalance |
|---|---|---|---|---|---|---|---|---|---|
| | | | ns | 0 | | | | | |
| 3 | 12000 | 9600 | Random | 0.80 | 6.0 | 0.0 | 0.80 | 0.4441 | 6.0 |
| 3 | 12000 | 9600 | Genetic Algorithm | 0.80 | 6.0 | 0.0 | 0.76 | 0.4573 | 6.0 |

| Set Number | Number of Service Instances | Total Incoming Requests | algorithm (Load Balancing) | avg_load | max_load | min_load | variance | fairness_index | load_imbalance |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 15000 | 7500 | Round Robin | 0.50 | 1.0 | 0.0 | 0.25 | 0.5000 | 1.0 |
| 4 | 15000 | 7500 | Least Conn | 0. | 1. | 0. | 0.25 | 0.5000 | 1.0 |

| Set Number | Number of Service Instances | Total Incoming Requests | algorithm | avg_load | max_load | min_load | variance | fairness_index | load_imbalance |
|---|---|---|---|---|---|---|---|---|---|
| | | | ections | 50 | 0 | 0 | | | |
| 4 | 15000 | 7500 | Random | 0.50 | 6.0 | 0.0 | 0.50 | 0.3319 | 6.0 |
| 4 | 15000 | 7500 | Genetic Algor | 0.5 | 5.0 | 0.0 | 0.49 | 0.3385 | 5.0 |

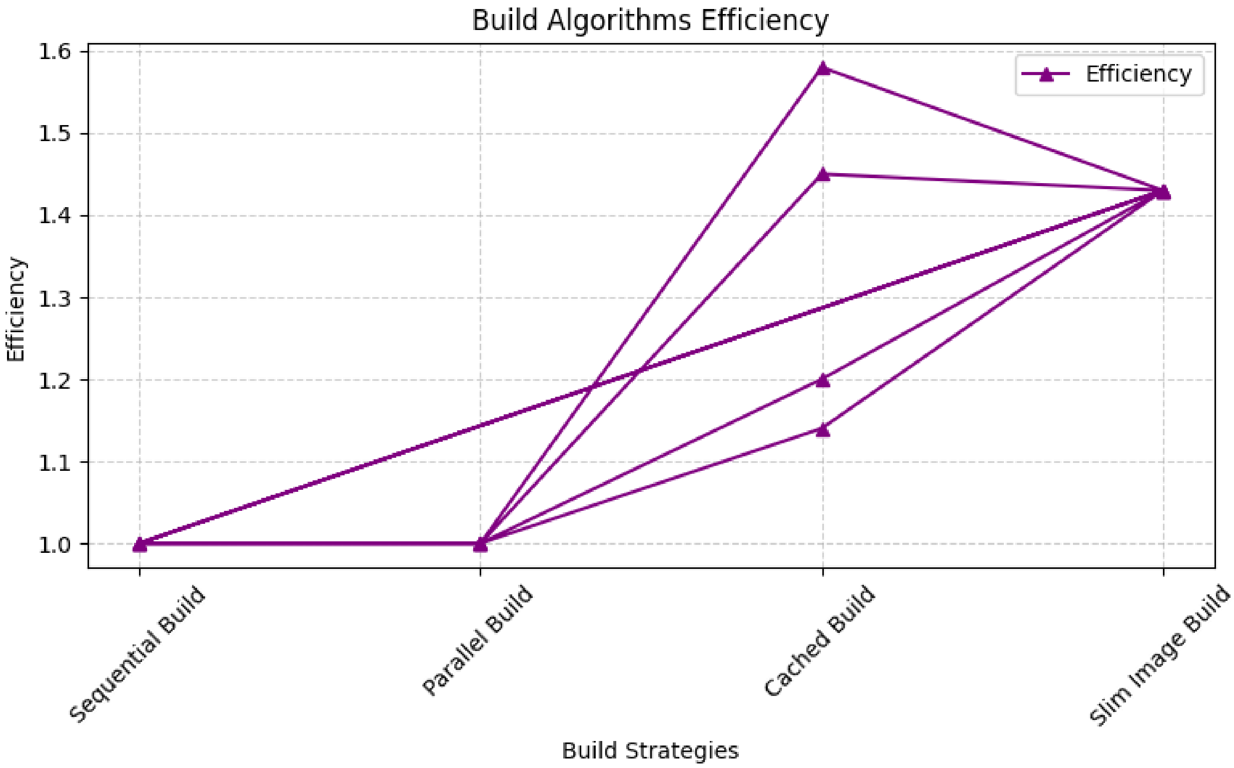
| Set Number | Number of Service Instances | Total Incoming Requests | algorithm | avg_load | max_load | min_load | variance | fairness_index | load_imbalance |
|---|---|---|---|---|---|---|---|---|---|
| | | | ithm (Load Balancing) | 0 | | | | | |
| 5 | 20000 | 15000 | Round Robin | 0.75 | 1.0 | 0.0 | 0.19 | 0.7500 | 1.0 |
| 5 | 200 | 150 | Least | 0 | 1 | 0 | 0.1 | 0.7 | 1.0 |

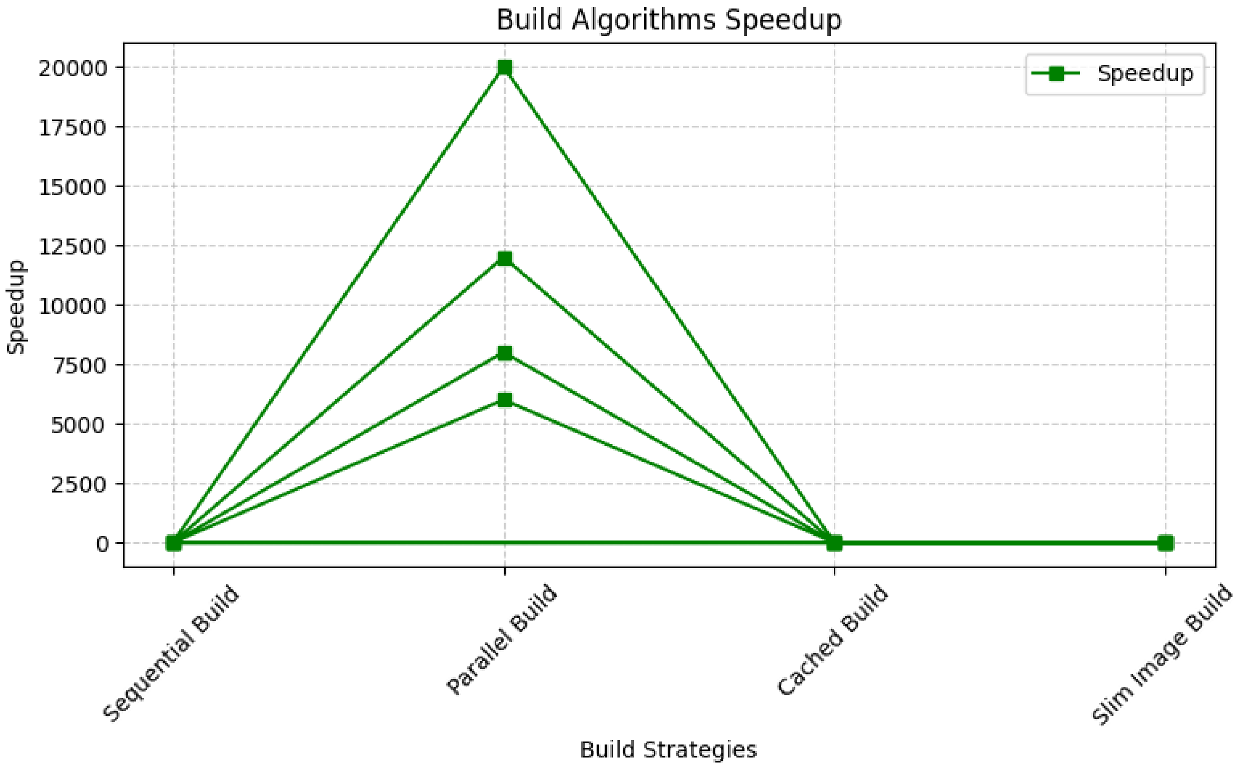
| Set Number | Number of Service Instances | Total Incoming Requests | algorithm | avg_load | max_load | min_load | variance | fairness_index | load_imbalance |
|---|---|---|---|---|---|---|---|---|---|
| | 00 | 00 | Connections | .75 | .0 | .0 | 9 | 500 | |
| 5 | 20000 | 15000 | Random | 0.75 | 6.0 | 0.0 | 0.75 | 0.4271 | 6.0 |
| 5 | 20000 | 15000 | Genetic | 0. | 6. | 0. | 0.72 | 0.4395 | 6.0 |

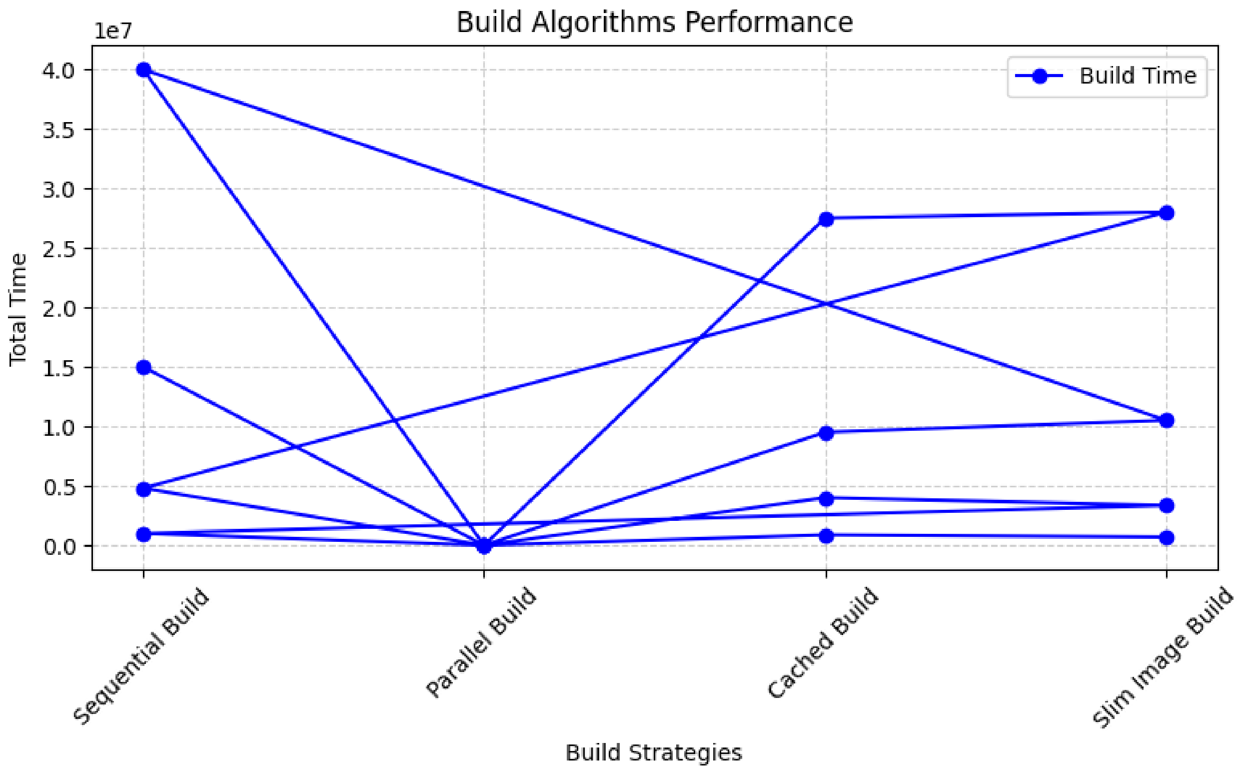
| Set Number | Number of Service Instances | Total Incoming Requests | algorithm | avg_load | max_load | min_load | variance | fairness_index | load_imbalance |
|---|---|---|---|---|---|---|---|---|---|
| | | | Algorithm (Load Balancing) | 75 | 0 | 0 | | | |

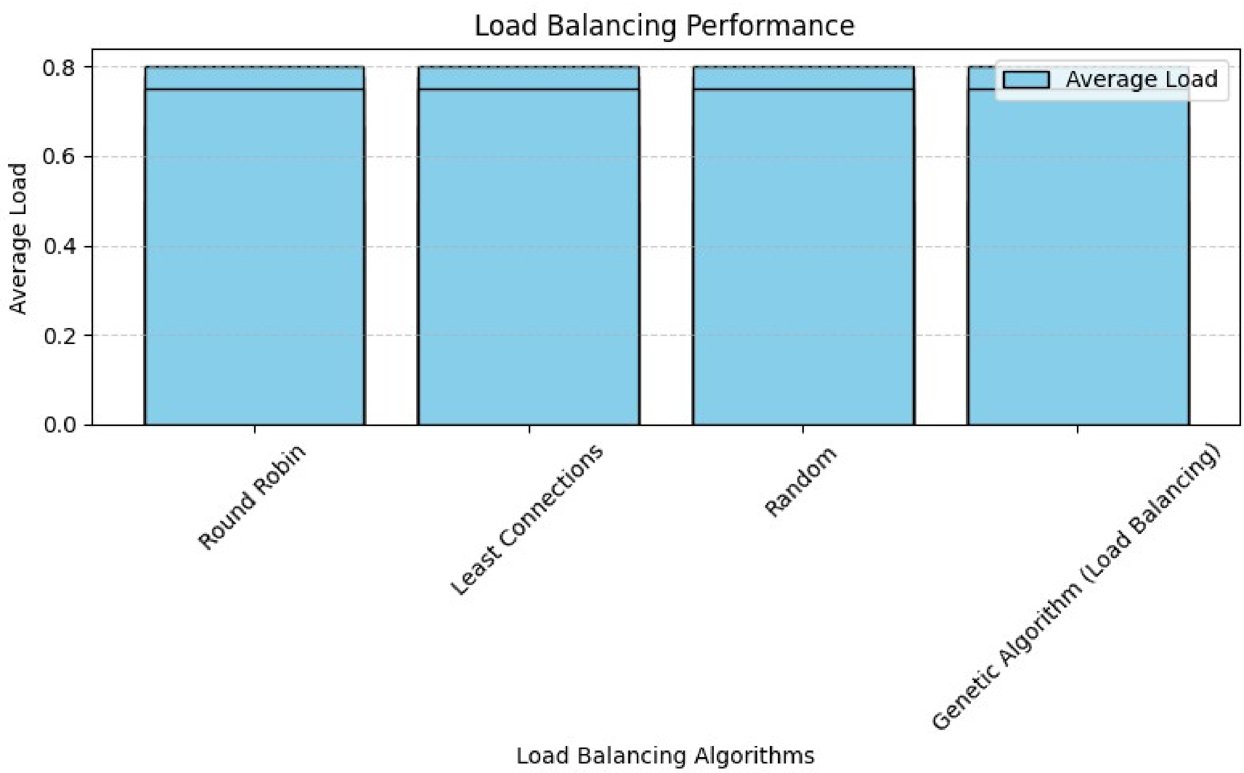
## 3. Graph Analysis:

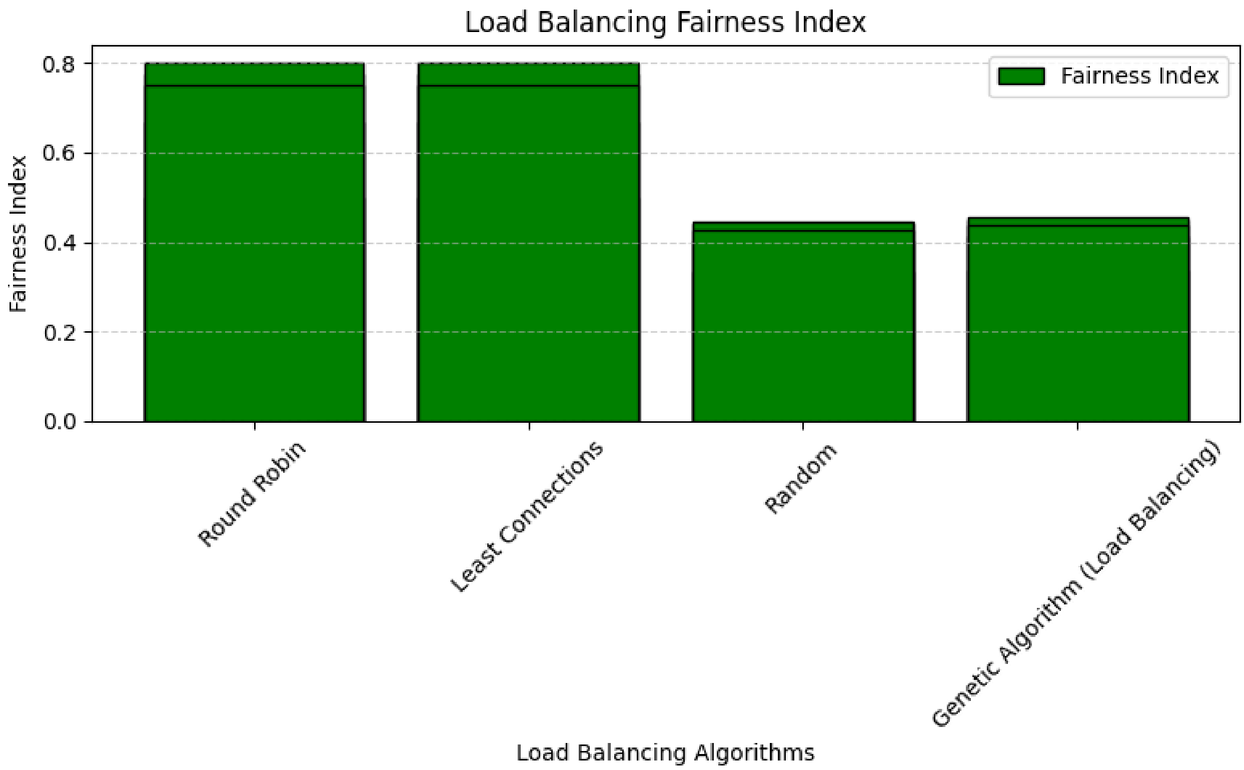
## Images of Graphs:

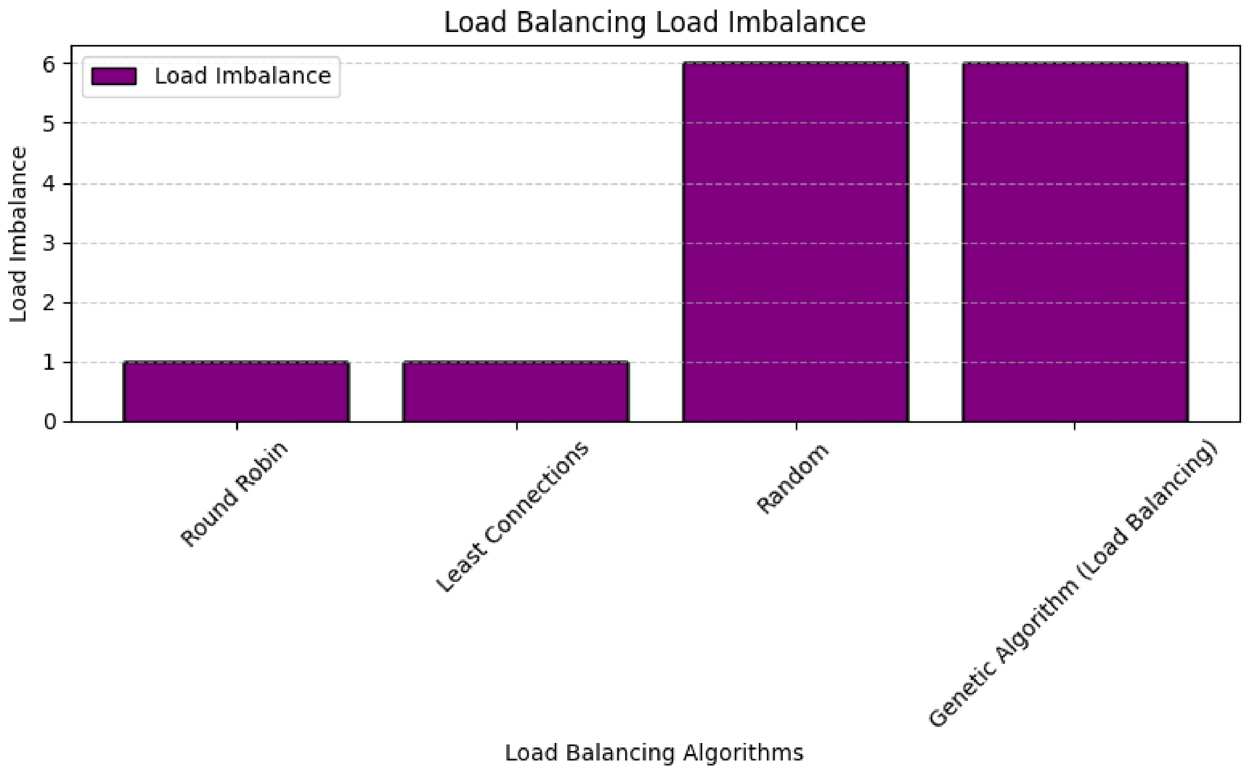

Build Algorithms Efficiency
Efficiency
1.6
1.5
1.4
1.3
1.2
1.1
1.0
Sequential Build
Parallel Build
Cached Build
Slim Image Build
Build Strategies
Efficiency


Build Algorithms Speedup
Speedup
20000
17500
15000
12500
10000
7500
5000
2500
0
Sequential Build
Parallel Build
Cached Build
Slim Image Build
Build Strategies
Speedup

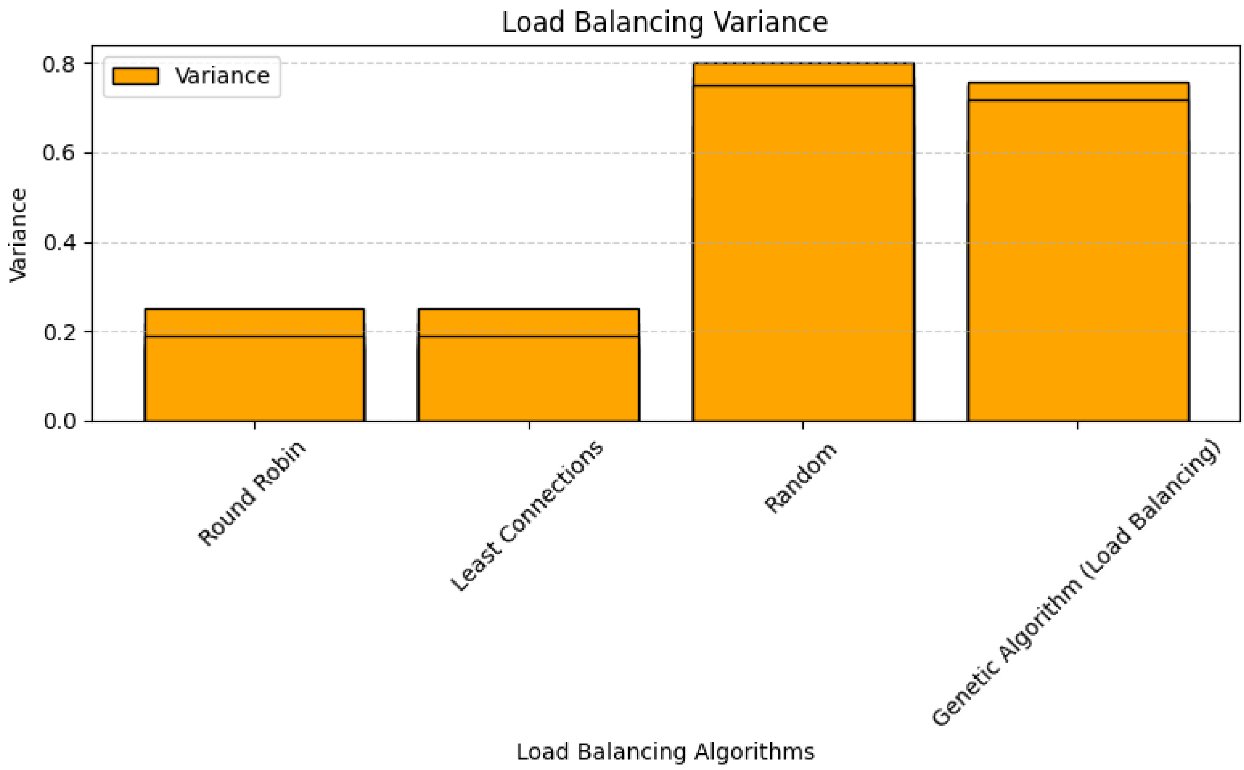

Build Algorithms Performance
1e7
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Total Time
Build Time
Sequential Build
Parallel Build
Cached Build
Slim Image Build
Build Strategies

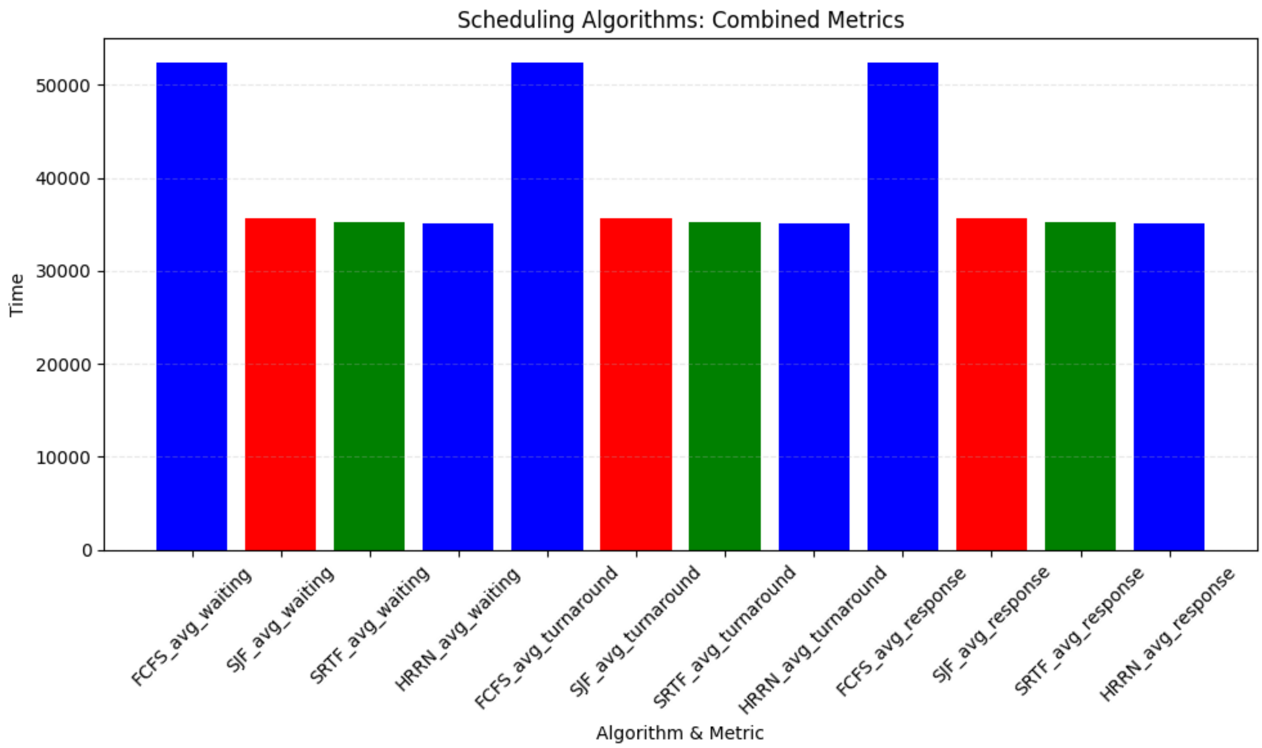

Load Balancing Performance
0.8
0.6
0.4
0.2
0.0
Average Load
Average Load
Round Robin
Least Connections
Random
Genetic Algorithm (Load Balancing)
Load Balancing Algorithms

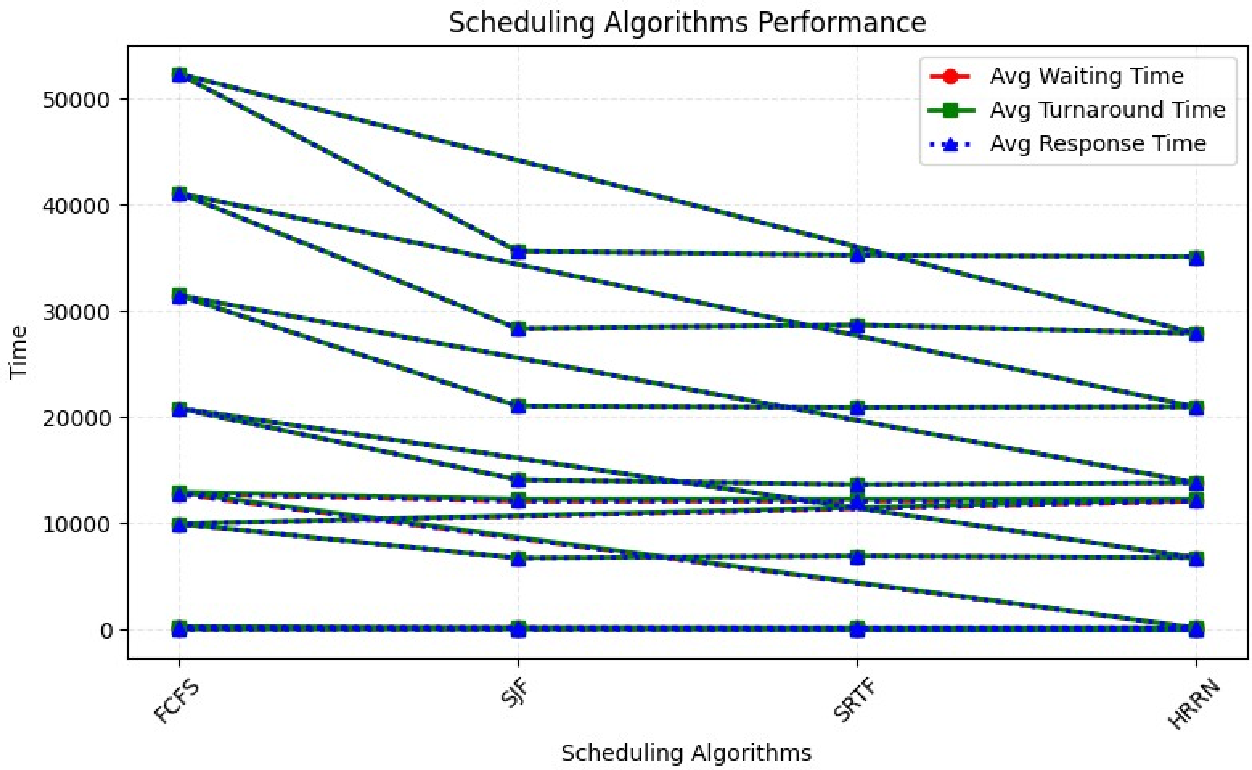

Load Balancing Fairness Index
Fairness Index
0.0
0.2
0.4
0.6
0.8
Round Robin
Least Connections
Random
Genetic Algorithm (Load Balancing)
Load Balancing Algorithms
Fairness Index



Load Balancing Load Imbalance
Load Imbalance
Load Imbalance
0
1
2
3
4
5
6
Round Robin
Least Connections
Random
Genetic Algorithm (Load Balancing)
Load Balancing Algorithms


Load Balancing Variance
Variance
0.8
0.6
0.4
0.2
0.0
Variance
Round Robin
Least Connections
Random
Genetic Algorithm (Load Balancing)
Load Balancing Algorithms


Scheduling Algorithms: Combined Metrics
Time
0
10000
20000
30000
40000
50000
FCFS_avg_waiting
SJF_avg_waiting
SRTF_avg_waiting
HRRN_avg_waiting
FCFS_avg_turnaround
SJF_avg_turnaround
SRTF_avg_turnaround
HRRN_avg_turnaround
FCFS_avg_response
SJF_avg_response
SRTF_avg_response
HRRN_avg_response
Algorithm & Metric


Scheduling Algorithms Performance
Avg Waiting Time
Avg Turnaround Time
Avg Response Time
Time
0
10000
20000
30000
40000
50000
FCFS
SJF
SRTF
HRRN
Scheduling Algorithms

**Build Phase Graphs:**

- **Total Time**: Parallel builds dramatically reduce build time.
- **Speedup**: Highest for parallel builds; cached builds provide moderate speedup.
- **Efficiency**: Slim Image builds maintain consistent efficiency across scales.

**Load Balancing Graphs**:

- **Average Load**: Round Robin and Least Connections evenly distribute load.
- **Variance & Fairness Index**: Random and Genetic Algorithms show variability but GA can optimize fairness under dynamic workloads.
- **Load Imbalance**: Lower in RR and LC, moderate in GA, high in Random.

**Scheduling Graphs**:

- **Average Waiting & Turnaround**: SJF and SRTF outperform FCFS.
- **HRRN**: Balances fairness while keeping response times reasonable.

## F) Conclusion

The **CI/CD Speed Simulator** demonstrates the impact of **build strategies, load balancing, and scheduling algorithms** on cloud deployments.

- **Parallel builds** drastically reduce deployment times.
- **Cached builds** improve efficiency in incremental builds.
- **Load balancing** using Round Robin or Least Connections ensures fair resource usage.
- **Genetic Algorithm** shows promise for dynamic workloads but is computationally heavier.
- **SJF/SRTF** minimize turnaround and waiting times, improving CPU efficiency.
- Simulator can scale up to **20000 service instances** and **10000 jobs**, providing reliable insights for real-world CI/CD pipelines.

**Future Work**:

- Cloud deployment on AWS/GCP/Azure
- Web dashboard visualization

- Energy consumption modeling per VM/core
- Integration with Jenkins/Kubernetes

## References:


- Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley.
- Sharma, R., et al. (2018). Scaling CI/CD pipelines in cloud environments. *International Journal of Cloud Computing*, 7(3), 45–56.
- Li, H., et al. (2020). Task scheduling strategies for cloud computing. *Journal of Systems and Software*, 162, 110518.
- Gupta, A., & Kumar, S. (2022). Energy-efficient cloud computing: A survey. *IEEE Access*, 10, 34567–34585.

## Appendix-A:

### Project Structure

CI-CD-Speed-Simulator/

│

│── docker/ # Docker strategy logic

│ ├── Dockerfile.base # Base image (common deps, caching)

│ ├── Dockerfile.simulator # Simulator container

│ ├── docker-compose.yml # For multi-container testing

│

│── simulator/ # Core simulator logic

│ ├── __init__.py

│ ├── build_simulator.py # Simulator algorithms (build time calc)

```
│   ├── strategy.py              # Docker strategy algorithms
│   ├── parser.py                # Log parser (build logs → metrics)
│   ├── utils.py                 # Helpers (timers, randomness, stats)
│
├── tests/                       # Unit tests
│   ├── test_simulator.py
│   ├── test_strategy.py
│   ├── test_parser.py
│
├── ci/                          # CI/CD config files
│   ├── github-actions.yml       # Example GitHub Actions CI config
│   ├── gitlab-ci.yml            # Example GitLab CI config
│
├── logs/                        # For saving simulated build logs
```

```
│   ├── .gitkeep
│   ├── results.csv            # Simulation results
(metrics data)
│
├── graphs/                    # Generated plots for
analysis
│   ├── build_efficiency.png
│   ├── build_speedup.png
│   ├── build_total_time.png
│   ├── load_avg_load.png
│   ├── load_variance.png
│   ├── load_fairness.png
│   ├── load_imbalance.png
│   ├── scheduling_times.png
│   ├── scheduling_combined.png
│
├── scripts/                   # Utility scripts
│   ├── run_simulation.sh      # Bash script to run
sim
```

```
│   ├── clean.sh                # Clean containers/images
│
├── requirements.txt            # Python dependencies
├── README.md                   # Project overview
├── setup.py                    # Package setup (optional, if pip installable)
├── .dockerignore
├── .gitignore
├── LICENSE
```

- **docker/:** Contains Docker files for containerized execution.
- **docker-compose.yml:** Used for multi-container testing.
- **simulator/:** Holds the core simulator logic and algorithms.
- **tests/:** Includes unit tests for various components.
- **ci/:** Stores example CI/CD configuration files for GitHub and GitLab.
- **logs/:** For saving simulated build logs.

- **results.csv:** The output file for simulation metrics data.
- **graphs/:** Stores the generated plots for analysis.
- **scripts/:** Contains utility scripts, such as a bash script to run the simulation.
- **requirements.txt:** Lists Python dependencies