CLOUD EDGE FOR TASK DEPLOYMENT AND LOAD BALANCING

Sathyanarayana S [1], Zikriya Ahmed[2], Srijan A Sanicum[3], Manoj P[4], Samarth G[5]

[1]Professor,[2]Final year Student,[3]Final year Student, [4]Final year Student,[5]Final year Student

Department of Computer Science and Engineering, Jawaharlal Nehru New College of Engineering, Shimoga

***_

Abstract – This report outlines the JCETD (Joint Cloud-Edge Task Deployment) strategy, aimed at optimizing task deployment and load balancing within joint cloud-edge datacenters. The methodology involves simulating the task deployment process as a deep reinforcement learning endeavor within the cloudedge model. Through continuous exploration and utilization of the system environment, tasks are strategically allocated to ensure optimal performance and load balancing. The ultimate goal is to achieve efficient computing capabilities and overall system equilibrium within the cloud-edge infrastructure.

***Key Words*:** Task Deployment, Cloud Scheduling, Load Balancing.

1. INTRODUCTION

Edge computing is slowly moving cloud computing applications, data and services from centralized nodes to edge computing is gradually shifting cloud computing applications, data, and services from centralized nodes to the network's edge. Positioned between terminal devices and traditional cloud computing data centers, edge computing is designed to handle low-latency and real-time tasks. This approach brings the cloud closer to endusers, offering computing and services with minimal latency. While edge computing significantly reduces latency, the improper assignment of tasks can lead to an uneven load distribution among nodes. Due to the diversity and heterogeneity of edge computing nodes, conventional load balancing algorithms are not directly applicable, making edge computing load balancing a critical research area in academia.

Load balancing strategies generally fall into two categories: static and dynamic. Static load balancing algorithms do not consider the prior state of the node while distributing the load, and they work well when nodes have minimal load variation. However, they are not suitable for the dynamic nature of the edge environment. A dynamic load balancing technique for edge computing, based on intermediary nodes, considers the previous state of a node when distributing the load. This report proposes a network architecture for edge computing based on intermediary nodes to enhance the acquisition of node state information. The intermediary node classifies and evaluates the node's status

using intrinsic attribute values and real-time attribute values. It then returns the node information with the relatively lightest load.

2. RELATED WORKS

In the study [1] proposed Efficient task deployment and load balancing are critical for the optimization of joint "cloud-edge" datacenters. Current research has predominantly focused on unilateral load balancing within either the cloud or edge computing centers, overlooking the broader challenge of balancing loads across the entire system. Addressing this gap, this paper proposes a novel approach. In [2] have In the contemporary technological landscape, the ascendancy of cloud computing has been indisputable, offering a robust infrastructure for delivering scalable and dependable services to a diverse array of users, spanning from individual consumers to large-scale enterprises. However, as the demands of computing continue to evolve and diversify, the limitations inherent in traditional cloud architectures have become increasingly apparent. It is within this context that emerging paradigms such as edge computing have garnered considerable attention and adoption.

3. METHODOLOGY

There are mainly 6 steps:

- **Requirement Analysis and System Design:** Identify system requirements and design the architecture, specifying cloud and edge roles, data flow, and communication protocols.
- **Algorithm Development:** Develop algorithms for task deployment and load balancing, considering task size, resource availability, latency, and cost.
- **Simulation Environment Setup:** Configure a simulation environment using appropriate tools to model the cloud-edge system and set up various testing scenarios.
- **Algorithm Integration and Testing:** Integrate the algorithms into the simulation environment and conduct initial testing to verify functionality and address issues.
- **Performance Evaluation:** Perform extensive simulation runs to evaluate performance using defined metrics and compare results with existing solutions.
- **Real-World Implementation and Validation:** Develop a prototype, implement it in a real-world setting, conduct a case study for validation, and refine the system based on feedback.

4. PROPOSED SYSTEM

The Joint Cloud-Edge Task Dispatching (JCETD) strategy implemented in the Python script optimizes task scheduling in a heterogeneous computing environment encompassing both cloud and edge resources. By considering factors such as task latency requirements, service times, and resource availability across

cloud and edge nodes, JCETD dynamically dispatches tasks to the most suitable computing resource. Tasks with stringent latency requirements are prioritized for execution on edge devices closer to the point of data generation, minimizing latency and enhancing responsiveness. Conversely, tasks with less stringent latency constraints or higher computational demands are offloaded to the cloud for processing, leveraging its abundant computational resources. This approach ensures efficient resource utilization while meeting the diverse latency requirements of tasks in distributed computing environments.

First-In First-Out (FIFO) Scheduling: The First-In First-Out (FIFO) scheduling policy incorporated in the Python script prioritizes task execution on edge computing nodes based on their arrival order. In this strategy, tasks arriving at the edge are immediately dispatched to available edge devices for processing, without considering their latency requirements or computational complexity.

JCETD Task Deployment: The Joint Cloud-Edge Task Deployment (JCETD) model facilitates efficient task scheduling in a distributed computing environment comprising both cloud and edge resources.

Shortest Job First (SJF) Scheduling: The Shortest Job First (SJF) scheduling policy incorporated in the Python script prioritizes task execution on edge computing nodes based on the tasks' computational complexity. In this strategy, tasks arriving at the edge are immediately assessed, and those with the shortest expected processing time are dispatched to available edge devices first, regardless of their arrival order or latency requirements.

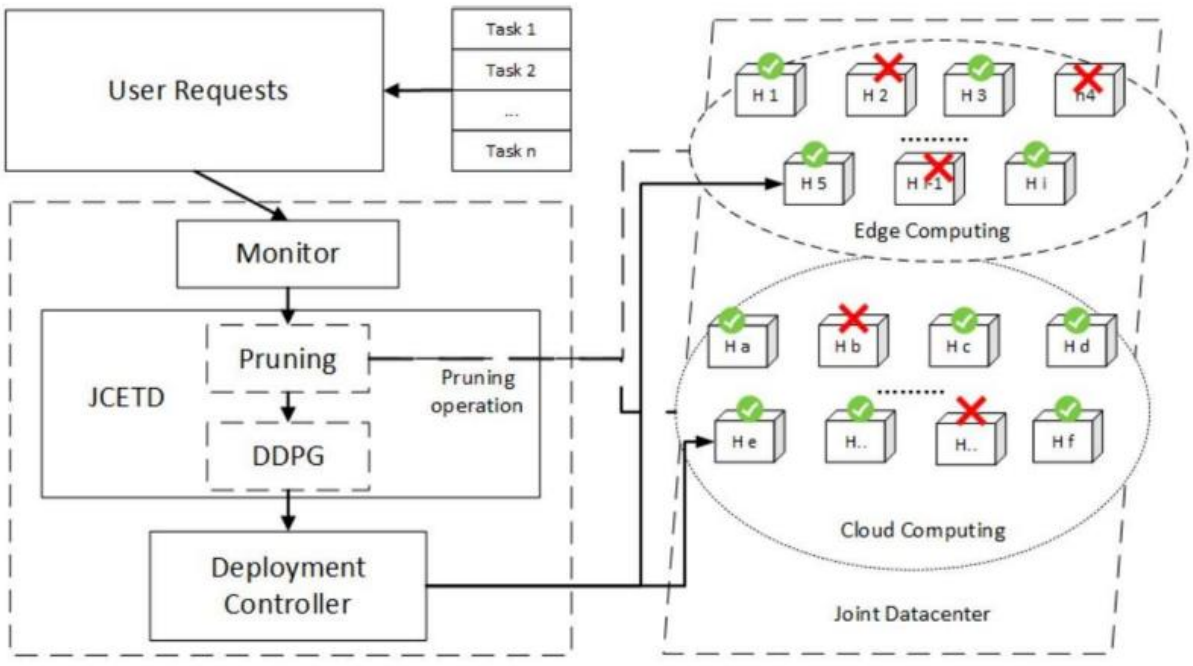

Fig. 3.1: Systems Architecture

4.1 JCETD Function:

```
def simulate(datacenters, tasks):
    load_balance_degrees = []
    for task in tasks:
        if task.location == 'cloud':
            heapq.heappush(datacenters[0].task_queue, task)
        else:
            heapq.heappush(datacenters[random.randint(1, len(datacenters) - 1)].task_queue, task)


        load_balance_degrees.append([datacenter.calculate_load_balance_degree() for datacenter in
    datacenters])

    schedule = []
    for datacenter in datacenters:
        while datacenter.task_queue:
            task = heapq.heappop(datacenter.task_queue)
            schedule.append(datacenter.schedule_task(task))

            load_balance_degrees.append([datacenter.calculate_load_balance_degree() for datacenter in
    datacenters])

        return schedule, load_balance_degree
```

5. TECHNOLOGY USED

- Python: Python is the primary programming language used in the code snippet.
- Visual Studio Code: VS Code is the integrated development environment (IDE) used for writing, debugging, and running the Python code.
- Python Extension for Visual Studio Code: Install the Python extension for VS Code, which provides features like IntelliSense, linting, debugging, and code navigation specific to Python.
- Matplotlib: Matplotlib is a plotting library for Python used for creating visualizations, such as line plots, histograms, and scatter plots. It's used in the code snippet to visualize the load balance degree over time

6. OUTPUT

The following graph illustrates the load balance degree over time for the Job Earliest Completion Time Deadline (JECTD) scheduling algorithm. The load balance degree is plotted against time steps, providing insights into the dynamic nature of workload distribution within the distributed computing network under the JECTD scheduling algorithm.

The load balance degree remains relatively stable over time under the JECTD scheduling algorithm, indicating consistent workload distribution within the network. The prioritization of tasks based on

deadlines for earliest completion time ensures efficient allocation and execution of tasks, minimizing fluctuations in workload distribution.

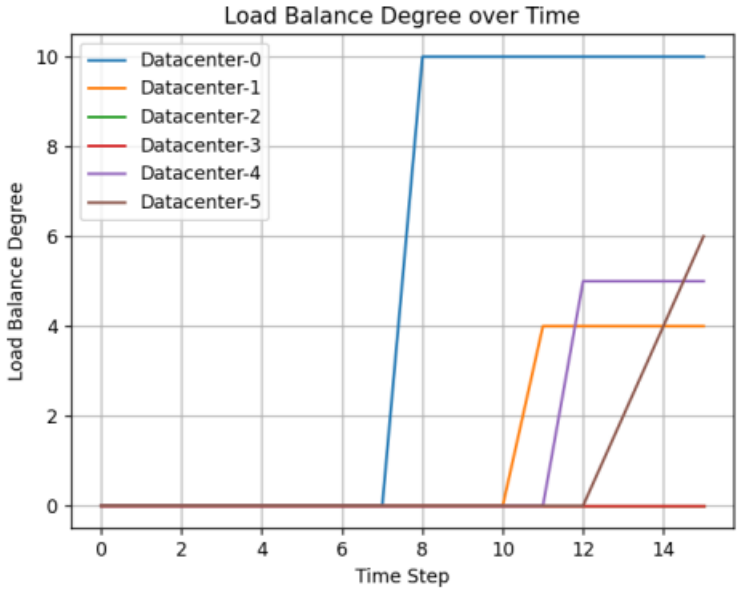

Fig 6.1: JCETD Scheduling Output

7. CONCLUSION

Hand Cloud-edge computing for task deployment and load balancing seamlessly merges cloud computing's scalability with edge computing's low-latency advantages, positioning computing resources nearer to data origins or end-users. This proximity significantly diminishes latency, enhancing system performance and user experiences. Load balancing strategies intelligently distribute computing tasks across edge nodes and cloud resources, optimizing resource utilization and bolstering system reliability. The immediate benefits include reduced latency for critical applications, adaptable resource allocation for varying workloads, heightened security with localized data processing, and real-time analytics capabilities for immediate insights, collectively driving innovation and efficiency in modern computing environments.

8. FUTURE SCOPE

- Optimized Resource Management: Developing more advanced load balancing algorithms and resource management techniques to ensure optimal utilization of edge and cloud resources based on dynamic workload patterns.
- Edge AI and Machine Learning: Integrating artificial intelligence (AI) and machine learning (ML) capabilities at the edge to enable intelligent decision-making, predictive analytics, and automation in edge computing environments.

- 5G Integration: Leveraging the capabilities of 5G networks to enhance connectivity, data transfer speeds, and support for a wide range of IoT devices and edge applications .

REFRENCES

[1] Yunmeng Dong , Gaochao Xu, Meng Zhang , and Xiangyu Meng "A HighEfficient Joint 'Cloud Edge' Aware Strategy for Task Deployment and Load Balancing".

[2] BIRJU Tanka and Dr.VAIBHAV GANDHI "A Comparative Study on Cloud Computing, Edge Computing and Fog Computing", 2023, IOS Press eBook

[3] He Sun, "Resource Deployment and Task Scheduling Based on Cloud Computing", Publisher: IEEE, 2022 IEEE 2nd International Conference on Computer Systems (ICCS)

[4] K. M. Aslam Uddin, "review of task scheduling in cloud computing based on nature-inspired optimization algorithm", June 2023Cluster Computing 26(5):1-31

[5] Tanzila Saba1 Amjad Rehman, "Cloud-edge load balancing distributed protocol for IoE services using swarm intelligence", Cluster Computing (2023) 26:2921–2931June 2021

[6] Zeinab Nezami , Kamran Zamanifar, "Decentralized Edge-to-Cloud Load Balancing: Service Placement for the Internet of Things", March 5, 2021

[7] Xiaolong Xu, Qingxiang Liu, Yun Luo, Kai Peng, , "A computation offloading method over big data for IoT-enabled cloud-edge computing", Published - Jun 2019 Future Generation Computer Systems