PCET's Pimpri Chinchwad University, Pune

Learn | Grow | Achieve

Name	
Checked By	Dr.Avinash Golande
Sign	
Date	
Grade	

EXPERIMENT NO. 02

- **Aim:** Develop a Python program to demonstrate the concept of elasticity by simulating workload increase and resource addition.

- **Objective:**
  1. Emulate increasing and decreasing workloads.
  2. Show how resources (threads/processes) can be dynamically added/removed (elasticity).
  3. Use a **queue** to represent incoming jobs.
  4. Use **threads** for resource handling (scale up/down).
  5. Monitor and **observe workload vs. resource usage**.
  6. **Conclude** how elasticity helps in efficient resource use.

- **Resources used :-** PC, Laptop**, VS** code, Jyputer

- **Theory:-**

Elasticity in Cloud Computing:

Introduction: -

In the modern era of cloud computing, elasticity is a fundamental concept that enables cloud systems to automatically adapt to changes in workload. Elasticity refers to the ability of a cloud infrastructure to dynamically allocate or deallocate computing resources (like CPU, memory, storage, etc.) to meet fluctuating demands.

Elasticity ensures that a cloud-based application or service can maintain performance and cost-efficiency without manual intervention. It is often confused with scalability, but while scalability refers to the capacity of the system to grow, elasticity emphasizes the automation of resource adjustments in real time.

1. **What is Elasticity?**
    1. Elasticity is the capability of a system to:
    2. Scale up: Automatically add more resources (like threads, virtual machines, containers, etc.) when the workload increases.
    3. Scale down: Automatically release unused resources when the workload decreases.
    4. This ensures that the application remains responsive under heavy loads and does not waste resources during idle times.

Example:

1. An e-commerce site may receive low traffic at night and high traffic during festive sales. An elastic system will automatically increase computing power during high traffic (scale up) and reduce it when demand drops (scale down).
2. Importance of Elasticity
3. Cost Efficiency: Resources are only used when needed, reducing operational costs.
4. Performance Optimization: Handles sudden workload spikes without delays.
5. Automation: Reduces the need for manual resource management.
6. Business Continuity: Ensures uptime and availability during traffic surges.

- **Concepts Used in the Simulation**
    - To simulate elasticity in Python, several **key computer science concepts** are used:

1. Queue

- Used to represent the workload (tasks/jobs). As workload increases, more items are added to the queue.

2. Threads and Processes

- **Threads** are used to simulate **workers** that process tasks.
- **Processes** (multiprocessing) simulate the external workload generation process.

3. Scale-Up Logic

- When the **queue length exceeds a threshold**, the system automatically creates new worker threads. This is analogous to scaling up in cloud systems.

4. Scale-Down Logic

- If a worker thread remains idle (no task to process) beyond a defined **timeout**, it shuts down. This mimics resource deallocation or scale-down.

5. Auto-Scaler

- A separate thread that continuously monitors the system and makes decisions to add or remove resources based on the current state.

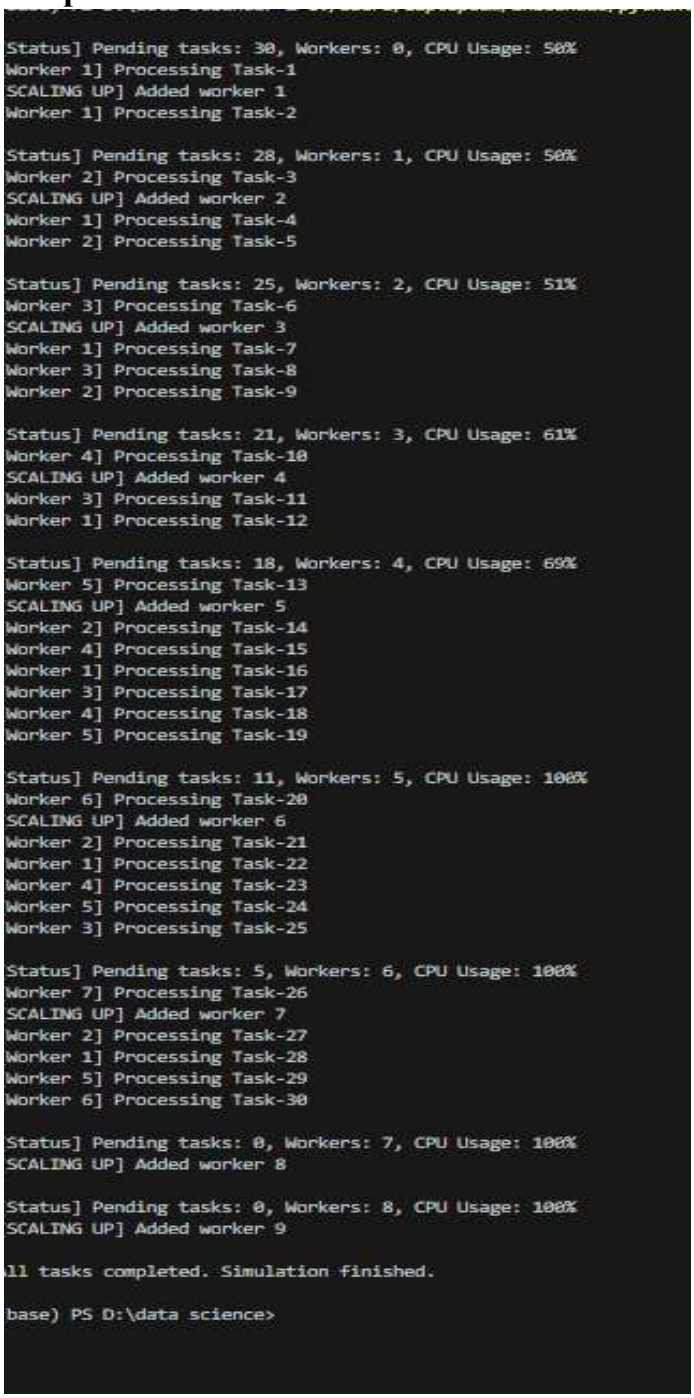
- **Observations from the Simulation**
  - When the queue grows due to an increase in job generation rate, the system **scales up** by adding more worker threads.
  - When the queue is empty and workers are idle, the system **scales down** by letting idle threads exit after a timeout.
  - This demonstrates **real-time elasticity** similar to cloud services like **AWS Auto Scaling**, **Azure Scale Sets**, or **Google Cloud Instance Groups**.

- **Code**

```
import time
import threading
import random
from queue import Queue

task_queue = Queue()
for i in range(30):
    task_queue.put(f"Task-{i+1}")

# Settings
workers = []
max_workers = 10
min_workers = 1
cpu_usage = 50  # starting CPU usage %

# Worker Function
def worker_function(worker_id):
    global cpu_usage
    while not task_queue.empty():
        task = task_queue.get()
        print(f"[Worker {worker_id}] Processing {task}")
        process_time = random.uniform(0.5, 1.5)
        time.sleep(process_time)
        task_queue.task_done()
        cpu_usage += int(process_time * 10)
```

```
        cpu_usage = min(cpu_usage, 100)

# Scaling Manager
def scale_manager():
    global workers, cpu_usage

    while True:
        pending_tasks = task_queue.qsize()
        current_worker_count = len(workers)
        print(f"\n[Status] Pending tasks: {pending_tasks}, Workers: {current_worker_count}, CPU Usage: {cpu_usage}%")

        # Scale UP
        if (pending_tasks > current_worker_count * 3 or cpu_usage > 70) and current_worker_count < max_workers:
            new_worker = threading.Thread(target=worker_function, args=(current_worker_count + 1,))
            workers.append(new_worker)
            new_worker.start()
            print("[SCALING UP] Added worker", current_worker_count + 1)

        # Scale DOWN
        elif (pending_tasks < current_worker_count * 2 and cpu_usage < 30 and current_worker_count > min_workers):
            removed_worker = workers.pop()
            print("[SCALING DOWN] Removed 1 worker (idle)")

        # CPU usage natural drop
        cpu_usage -= random.randint(5, 15)
        cpu_usage = max(cpu_usage, 10)

        # Stop condition
        if pending_tasks == 0 and not any(w.is_alive() for w in workers):
            print("\nAll tasks completed. Simulation finished.\n")
            break

        time.sleep(1)

# Start Scaling
scale_thread = threading.Thread(target=scale_manager)
scale_thread.start()
    scale_thread.join()
```

- **Output**



- **Conclusion**

Elasticity is one of the **cornerstone features** of cloud computing, enabling systems to respond effectively to changing demands. Through the use of **queues, threads, and process-based simulation**, the Python program demonstrates how real-world cloud platforms manage workloads dynamical