PROJECT TITLE – 2

A Distributed Systems Cluster Simulation Framework

General Guidelines:

- Create a GitHub Repository.
- Weekly progress according to the assigned problem statement must be pushed to the repo, which will be considered during the evaluation.
- Each team's weekly progress update in the repository must be shown to the teachers in class.
- Name the GitHub repository with the four SRNs in order and the project title.

Problem Statement:

Develop a lightweight, simulation-based distributed system that mimics core Kubernetes cluster management functionalities. The system will create a simplified yet comprehensive platform for demonstrating key distributed computing concepts

High Level Architecture

- Implement an API server that integrates key functionalities such as node management, pod scheduling, and health monitoring.
- Nodes periodically send health signals for themselves
- To simplify the assignment, the system consists of a CLI/Web Interface that communicates with an API Server (Central Control Unit), which manages the Cluster Infrastructure composed of Nodes and Pods.

API Server (Central Control Unit)

Design and implement the API Server, which manages the overall cluster operation. It contains three key components:

- Node Manager: Tracks registered nodes and their statuses. Maintains information about node resources (CPU cores) and availability.
- Pod Scheduler: Assigns pods to available nodes based on scheduling policies and resource requirements.
- Health Monitor: Receives health signals from nodes and detects failures to ensure system reliability.

check for availability:

- Nodes periodically send heartbeat signals to the API server.
- API server tracks node status and can detect node failures.

Nodes

- Nodes are computing units that host pods.
- When adding a new node to the system, **you must launch a new container to simulate the physical node in the cluster**.
- Each node maintains an array list of pod IDs it hosts to simulate pod deployment.
- Periodically send health check signals for themselves to the API server.

Pods

- Smallest deployable units, scheduled on nodes.
- Require specific CPU resources.
- Pods are simulated through entries in the node's pod ID array list.

Organizing Resources in the Cluster

Resources in the form of CPU cores need to be allocated across nodes. Each node contributes resources to the cluster, and pods consume portions of these resources. The Pod Scheduler helps map resource requirements to available node capacity.

Note on Pod Scheduling

In a Kubernetes-like system, pod scheduling is used to efficiently allocate workloads across multiple nodes. Each pod has specific resource requirements CPU and the scheduler ensures pods are placed on nodes with sufficient available resources. The purpose of scheduling is to optimize resource utilization, ensure high availability, and maintain system performance.

For example, let's say you have a pod that requires 2 CPU cores, and your cluster has nodes with varying resource capacities. The scheduler would find a node with at least 2 CPU cores and place the pod there. Once scheduled, the pod ID is added to the node's array list of pods.

Fault Tolerance

Implement mechanisms to handle Node failures. This could involve strategies like detecting failed nodes through missed heartbeats and rescheduling pods from failed nodes to healthy ones.

Client Interaction and Features

Develop a command line interface or a web interface to interact with the Kubernetes-like system and perform the following actions:

Node Operations

These operations involve the Node Manager component of the API Server:

- Add nodes to the cluster, specifying CPU cores (**each addition requires launching a new container**).
- List all nodes and their health status.

Pod Operations

- Launch pods with specific CPU requirements.

Node Addition Process Flow

Here's an overview of how the node addition process works:

1. Resource Specification: When a user wants to add a node to the cluster, they specify the available CPU cores.
2. Container Launch: The system launches a new container to simulate the physical node in the cluster.
3. Node Registration: The CLI/Web Interface sends a request to the API Server with the node specifications.
4. Node Manager Processing: The API Server's Node Manager component registers the new node and assigns it a unique identifier.
5. Resource Allocation: The system adds the node's resources to the cluster's available resource pool.
6. Heartbeat Initialization: The node begins sending periodic heartbeat signals to confirm its availability.
7. Status Update: The API Server confirms successful node addition and updates the cluster state.
8. Client Acknowledgment: The user receives confirmation that the node has been successfully added to the cluster.

Pod Launch Process Flow

Below is a high-level process flow for launching a pod in the system:

1. Client Request:
    - A client application initiates a request to launch a pod with specific CPU requirements.
2. Resource Validation:

- The API Server verifies if sufficient resources are available in the cluster to accommodate the pod.

3. Node Selection:
   - The Pod Scheduler component evaluates available nodes and selects an appropriate node based on resource availability and scheduling policies.
4. Resource Reservation:
   - The selected node's available resources are updated to reflect the allocation for the new pod.
5. Pod Deployment:
   - The API Server instructs the selected node to add the pod to its array list of pods.
   - The node adds the pod ID to its internal array list to simulate deployment.
   - Store this information in API server to relaunch pods during node failure
6. Status Update:
   - The node confirms pod addition to its array
7. Client Notification:
   - The system notifies the client about successful pod deployment, including the pod's assigned node and unique identifier.

Health Monitoring Process Flow

1. Periodic Heartbeats
   - Nodes send regular heartbeat signals to the API Server, including:
2. Health Monitor Analysis
   - The Health Monitor component analyzes the heartbeat data and updates node health statuses.
3. Failure Detection
   - If a node fails to send heartbeats within a specified timeframe, the Health Monitor marks it as potentially failed.
   - If a pod is reported as unhealthy in a node's heartbeat, the system can take appropriate action.
4. Recovery Actions
   - For confirmed node failures, the system can initiate pod rescheduling by:
     - Adding these pod IDs to healthy nodes' array lists to maintain service availability
5. Status Updates
   - The system updates its status information, which can be queried by users.

Weekly Guidelines

1. **Week 1 (15 M)**
    1. Implement API Server base implementation and Node Manager functionality.
    2. Add a Node ⟶ Provide CPU cores to join the cluster (launch new container).

2. **Week 2 (15 M)**
    1. Implement Pod Scheduler and Health Monitor with node heartbeat mechanism.
    2. Launch a Pod ⟶ Request a pod with CPU system assigns it automatically (adds to node's pod ID array).

3. **Week 3 (10 M)**
    1. List Nodes ⟶ See all nodes and their health status.
    2. System testing and documentation.

Key Deliverables

1. Node addition to cluster
2. Pod scheduling
3. Node health monitoring and failure recovery
4. Listing of Nodes in cluster along with their health status

Technology Stack

1. Docker (For Simulating Nodes)
2. Flask, Node.js, or any other programming language for implementing an API server.
3. First-Fit, Best- Fit, Worst-Fit scheduling algorithm for pod Scheduling

Further Enhancements

1. Implement auto-scaling for nodes based on cluster load.
2. Add pod resource usage monitoring.
3. Add network policy simulation for pod communication control.