SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES, CHENNAI – 602105

CAPSTONE PROJECT REPORT

TITLE

Develop the Resource management based on Elastic cloud balancing and job shop scheduling

Submitted to

SAVEETHA SCHOOL OF ENGINEERING

By

M. Sumanth Kumar (192210052)

Guided by

Dr. J. Chenni Kumaran

PROBLEM STATEMET:

1)Problem Statement:

In modern computing environments, efficient resource management is crucial for optimizing performance and cost-effectiveness. Traditional approaches often struggle to dynamically allocate resources based on varying workloads and demand fluctuations. Elastic cloud balancing and job shop scheduling offer promising solutions to address these challenges by enabling adaptive resource allocation and efficient job scheduling.

2) Quality of Service (QoS) Requirements

The project aims to achieve high QoS by:

- Ensuring minimal job completion times.
- Optimizing resource utilization.
- Providing scalability to handle varying workloads.
- Guaranteeing reliability and fault tolerance.

3)Evaluation Cum Classification (EC2):

The EC2 calculation involves three key aspects:

- **Cloud Resource Evaluation and Ranking**: Assigning scores to resources based on their capabilities.
- **Cloud Evaluation Framework by Scheduler**: Efficiently scheduling tasks to appropriate resources.
- **Job Assessment and Grouping**: Grouping similar tasks for optimal execution.

4)CRPP-PSO (Cloud Resource Provisioning Problem - Particle Swarm Optimization)

- **Concept:** CRPP-PSO involves using Particle Swarm Optimization (PSO) to solve the resource provisioning problem in the cloud dynamically.
- **Benefits:** PSO helps in balancing resource utilization across multiple virtual machines (VMs) or instances based on workload predictions and historical data.

5)Significance of Job Shop Scheduling in Cloud Environments

In cloud environments, efficient job shop scheduling ensures optimal resource utilization and minimizes job completion times.

Proposed Design Work in Best cloud node prediction and matchmaking:

Dashboard Overview:

- Develop a predictive model to forecast resource demands based on historical data.
- Implement matchmaking algorithms to automatically select the best cloud nodes for deploying workloads.
- Improve efficiency, reduce costs, and enhance performance in cloud resource management.

Components

- Data Collection: Collect historical workload data, instance performance metrics, and cost information from cloud providers (e.g., AWS CloudWatch, Azure Monitor).
- Predictive Analytics: Develop machine learning models (e.g., regression, time series analysis) to predict future resource demands based on historical data.
- Matchmaking Engine: Implement algorithms (e.g., rule-based systems, genetic algorithms) to match predicted workload requirements with suitable cloud instances.

Workflow

Data Collection and Preprocessing

- Data Sources: Use APIs or monitoring tools to gather real-time and historical data on workload patterns, instance performance, and costs.
- Data Preprocessing: Clean, transform, and aggregate data to prepare it for training predictive models and input into matchmaking algorithms.

Predictive Analytics

- Model Training: Train machine learning models (e.g., Random Forest, LSTM neural networks) on historical data to predict future workload demands accurately.
- Prediction: Use trained models to forecast resource requirements for upcoming periods based on historical trends and current workload patterns.

Matchmaking Algorithms

- Selection Criteria: Define criteria (e.g., cost-efficiency, performance metrics, SLA requirements) for selecting the best cloud nodes.
- Algorithm Implementation: Develop matchmaking algorithms to evaluate available cloud instances and select the optimal node based on predicted workload demands.

Implementation Details

Technologies and Tools

- Programming Languages: Python for scripting and data analysis.

- Machine Learning Libraries: scikit-learn, TensorFlow, or PyTorch for building predictive models.
- Cloud APIs: AWS SDK (boto3), Azure SDK, or Google Cloud SDK for interacting with cloud providers.

Dashboard Overview:

- Start with a clean and intuitive dashboard.
- Provide an overview of the cloud nodes, their current status, and resource utilization.
- Include visualizations (such as graphs or heatmaps) to depict real-time performance metrics.
- Ensure that critical information is easily accessible at a glance.

Resource Selection:

- Allow users to select the type of workload or task they want to deploy.
- Based on the workload characteristics (e.g., compute-intensive, memory-intensive), recommend suitable cloud nodes.
- Provide clear labels and descriptions for each node to aid decision-making.

Predictive Insights:

- Incorporate predictive models to suggest the best node for a given workload.
- Display confidence scores or probabilities to indicate the reliability of predictions.
- Highlight any potential bottlenecks or resource constraints.

Customization Options:

- Let users customize prediction parameters (e.g., QoS thresholds, historical data window).
- Allow them to adjust prediction algorithms based on their specific requirements.

Alerts and Notifications:

- Implement alerts for resource overload, failures, or anomalies.
- Notify users when a node's performance deviates significantly from predictions.
- Provide actionable recommendations to mitigate issues.

User Feedback Loop:

- Gather feedback from users regarding prediction accuracy.
- Use this feedback to continuously improve the prediction models.
- Encourage users to report any discrepancies or unexpected behavior.

IMPLEMENTATION:

Connecting Components in Cloud:

System Architecture

- **Components:** Design a system architecture integrating ECB, CRPP-PSO, and job shop scheduling algorithms.
- **Technologies:** Use Python for scripting, AWS SDK (boto3) for EC2 instance management, and libraries/frameworks for PSO and job shop scheduling.

Workflow

- **Data Collection:** Gather workload data and performance metrics from AWS CloudWatch or similar tools.
- **Prediction:** Use machine learning models or statistical methods to predict future workload demands.
- **Resource Allocation:** Implement job shop scheduling algorithms (e.g., Genetic Algorithms) to optimize task scheduling on available instances.

Project Testing:

Unit Testing:

- Test individual components (CRPP model, API endpoints) in isolation.
- Verify correctness and edge cases.

Integration Testing:

- Test the entire system end-to-end.
- Validate data flow, predictions, and resource allocation.

Stress Testing:

- Simulate high loads (concurrent requests, varying workloads).
- Assess system performance, scalability, and response times.

Real-World Scenarios:

- Deploy the system in a production-like environment.
- Evaluate accuracy, reliability, and user satisfaction.

PERFORMANCE EVALUTION:

Accuracy Metrics:

Prediction Accuracy: Measure how well the system predicts the most suitable cloud node for a given workload**.**
Common metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R-squared.

Matchmaking Accuracy:
- Evaluate how accurately the system matches resources to tasks based on predicted rankings.
- Assess precision, recall, and F1-score.

Resource Utilization:
- Analyze how efficiently cloud resources are utilized.
- Calculate resource utilization ratios (CPU, memory, storage) for each node.
- Compare actual utilization with predicted utilization.

Response Time and Latency:
- Measure the time taken to predict the best cloud node for a task.
- Evaluate system responsiveness during peak loads.

Scalability:
- Assess how well the system scales with increasing workloads.
- Test performance under various levels of concurrent requests.

Real-World Testing:
- Deploy the system in a production-like environment.
- Monitor performance during actual usage.
- Gather feedback from users and stakeholders**.**

Comparative Analysis:
- Compare the proposed CRPP-PSO approach with other prediction models.
- Benchmark against existing cloud resource allocation methods.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define NUM_JOBS 3
#define NUM_MACHINES 2
```

```
// Structure for representing a job
typedef struct {
    int id;
    int duration; // Time required to complete the job
} Job;

// Structure for representing a machine
typedef struct {
    int id;
    int current_job_id; // Id of the job currently running on the machine, -1 if idle
    int remaining_time; // Remaining time for the current job
} Machine;

// Function prototypes
void assignJobToMachine(Job jobs[], Machine machines[]);
void printMachineStatus(Machine machines[]);

int main() {
    Job jobs[NUM_JOBS] = {{1, 5}, {2, 3}, {3, 4}};
    Machine machines[NUM_MACHINES] = {{1, -1, 0}, {2, -1, 0}};

    int time = 0;

    // Simulation loop
    while (1) {
        printf("Time: %d\n", time);

        // Check if there are any idle machines and assign jobs
        assignJobToMachine(jobs, machines);

        // Simulate job execution on machines
        for (int i = 0; i < NUM_MACHINES; ++i) {
            if (machines[i].current_job_id != -1) {
                machines[i].remaining_time--;
                if (machines[i].remaining_time == 0) {
                    printf("Machine %d finished job %d.\n", machines[i].id,
machines[i].current_job_id);
                    machines[i].current_job_id = -1; // Machine becomes idle
                }
            }
        }

        // Print current machine status
        printMachineStatus(machines);

        // Check if all jobs are completed
        int all_jobs_completed = 1;
        for (int i = 0; i < NUM_JOBS; ++i) {
            if (jobs[i].duration > 0) {
                all_jobs_completed = 0;
                break;
            }
        }

        if (all_jobs_completed) {
            printf("All jobs completed.\n");
            break;
        }

        // Increment time for simulation
        time++;
    }

    return 0;
}

// Function to assign jobs to idle machines
void assignJobToMachine(Job jobs[], Machine machines[]) {
    for (int i = 0; i < NUM_MACHINES; ++i) {
        if (machines[i].current_job_id == -1) { // Machine is idle
            for (int j = 0; j < NUM_JOBS; ++j) {
                if (jobs[j].duration > 0) { // Job is not completed
                    machines[i].current_job_id = jobs[j].id;
                    machines[i].remaining_time = jobs[j].duration;
                    jobs[j].duration = 0; // Job assigned, mark as completed
                    printf("Job %d assigned to Machine %d.\n", jobs[j].id, machines[i].id);
                    break;
                }
            }
        }
    }
}

// Function to print current machine status
void printMachineStatus(Machine machines[]) {
    for (int i = 0; i < NUM_MACHINES; ++i) {
        printf("Machine %d: ", machines[i].id);
        if (machines[i].current_job_id == -1) {
            printf("Idle\n");
        } else {
            printf("Job %d (Time Remaining: %d)\n", machines[i].current_job_id, machines[i].remaining_time);
        }
    }
    printf("\n");
}
```

CONCLUSION:

Developing resource management based on elastic cloud balancing and job shop scheduling involves integrating advanced algorithms and infrastructure management techniques. Elastic cloud balancing aims to dynamically allocate and deallocate cloud resources based on current demand, ensuring optimal resource utilization and scalability. This involves monitoring resource usage metrics such as CPU utilization, memory usage, and network traffic, and then making decisions to scale resources up or down accordingly. Job shop scheduling, on the other hand, focuses on efficiently scheduling jobs on available machines, considering factors like job duration, machine capabilities, and minimizing overall completion time or meeting deadlines.

Implementing these concepts in a real-world scenario requires a robust software architecture capable of handling dynamic workload changes and resource scaling. Technologies such as containerization (e.g., Docker) and orchestration tools (e.g., Kubernetes) play crucial roles in managing elastic cloud resources. Additionally, advanced scheduling algorithms like Genetic Algorithms, Ant Colony Optimization, or even Machine Learning-based approaches can enhance job shop scheduling accuracy and efficiency. Successful implementation involves balancing resource availability, performance requirements, and cost-effectiveness, ultimately enabling organizations to achieve greater operational efficiency and responsiveness to fluctuating demands in cloud environments.