Rashtreeya Sikshana Samithi Trust
RV COLLEGE OF ENGINEERING®
(Autonomous Institution Affiliated to VTU, Belagavi)

Operating Systems
Experiential Learning Report

Topic: Predictive CPU Scheduling Using Machine Learning

By:
Noel Shaji Mathew (1RV23CD034)
Aditya Bhandari (1RV23CD003)
AN Keerthi Saagar (1RV23CD007)

Prof. Sneha M.

TABLE OF CONTENTS

INTRODUCTION

In modern computing systems, process scheduling plays a crucial role in managing resources like CPU, memory, and storage efficiently. Operating systems typically use scheduling algorithms, such as First-Come-First-Serve (FCFS) or Shortest Job Next (SJN), to allocate resources to processes. While these traditional methods have been effective, they often rely on static rules or immediate resource demands, which may not always lead to optimal outcomes in dynamic environments. For instance, scheduling a high-demand process during peak CPU utilization can result in bottlenecks, reduced system performance, and delayed execution of other critical processes.

To address these limitations, predictive techniques using machine learning offer a promising solution. By analyzing key process metrics, such as RAM, disk, and network usage, machine learning models can forecast CPU requirements before a process is executed. This project explores the development and implementation of a predictive CPU scheduling system that leverages a Random Forest model for CPU usage prediction. By integrating this predictive capability into the scheduling framework, we aim to enhance system performance, minimize resource contention, and demonstrate the potential of intelligent process management in computing systems.

PROBLEM STATEMENT

Modern computing systems face significant challenges in efficiently managing and scheduling processes, particularly under dynamic and high-demand environments. Traditional scheduling methods often lack the ability to anticipate resource requirements, leading to CPU overloading, inefficiencies, and suboptimal system performance. The problem lies in the inability to predict a process's CPU usage based on its resource metrics, such as RAM, disk, and network usage, before execution. This project addresses this gap by leveraging machine learning to forecast CPU usage, enabling predictive scheduling decisions that optimize resource utilization, reduce bottlenecks, and enhance overall system efficiency.

OBJECTIVES

1. Simulate realistic system processes with random resource metrics (RAM, disk, and network usage)

2. Train or use a pre-trained machine learning model to predict CPU usage based on the metrics

3. Develop a simulation framework for process scheduling using multiprocessing.

4. Optimize resource utilization by scheduling or deferring processes based on predicted CPU usage.

5. Demonstrate the feasibility of predictive scheduling in a simulated environment.

METHODOLOGY

Data Simulation

1. **Simulate Process Metrics:**
    - Create a function to generate random values for process metrics such as RAM, disk, and network usage.
    - Use pandas to organize these metrics into a DataFrame.
    - Include unique identifiers (e.g., pid) for each process.
2. **Simulate Workloads:**
    - Use multiprocessing to define a function that mimics process execution by consuming resources (e.g., performing CPU-intensive tasks).

Machine Learning Model Development

1. **Collect Training Data:**
    - Using data from a dataset named "System Resources", containing process metrics (RAM, disk, network) and actual CPU usage.
2. **Train the Model:**
    - Select the Random Forest Regressor as the machine learning model.
    - Split data into training and testing sets using train_test_split.
    - Train the model using the fit method with features (metrics) and the target (CPU usage).
3. **Evaluate the Model:**
    - Use metrics like Mean Squared Error (MSE) to evaluate prediction accuracy on the test data.
    - Optimize hyperparameters if necessary.
4. **Save the Model:**
    - Save the trained model using joblib for reuse during scheduling.

CPU usage prediction

In this project, the CPU usage prediction is a critical component for deciding which processes to schedule based on their predicted CPU consumption. The prediction part of the code primarily involves the following steps:

1. Data Input Preparation

Before making any predictions, the system first gathers and prepares the data for input into the model. The metrics DataFrame is generated for each process, which contains the following columns:

- **ram**: The amount of RAM used by the process.
- **disk**: The disk space or disk usage of the process.
- **network**: The network usage (bandwidth consumption) by the process.

Each process has these attributes, and the goal is to predict the CPU usage based on them. These metrics serve as features for the prediction model.

2. Loading the Pre-Trained Model

Before making predictions, the system loads the pre-trained Random Forest Regressor model that was previously trained on historical data containing similar features (RAM, disk, network) and actual CPU usage values.

- The model is loaded using the joblib.load() function, which is used for serializing Python objects (in this case, the trained model).

3. Prediction Process

3.1 Input Data Creation

For each process, the relevant features (RAM, disk, and network) are collected into a DataFrame that serves as the input to the model.

```
input_data = pd.DataFrame([metric], columns=['ram',
'disk', 'network'])
```

Here:

- **metric** is a dictionary containing the current process's resource usage metrics.
- This dictionary is converted into a single-row DataFrame (input_data), where each column corresponds to one of the features (RAM, disk, and network usage).

3.2 CPU Usage Prediction

The model's .predict() method is then called with the input_data. This method takes the features (ram, disk, network) of the process as input and predicts the corresponding CPU usage. The prediction is stored in the variable predicted_cpu.

```
predicted_cpu = model.predict(input_data)[0]
```

- `.predict(input_data)`: The model makes a prediction for the given input data (the metrics of the current process).
- `[0]`: Since the model returns an array with one element (as there's only one process being predicted), the `[0]` extracts that value, which represents the predicted CPU usage.

3.3 Adjusting the Prediction

In some cases, raw predictions from models might need adjustment or scaling. In this code, the prediction is returned directly without any additional transformations:

```
adjusted_prediction = raw_prediction
```

This step could be expanded later if needed, such as by applying post-prediction normalization or correction based on system conditions.

4. Scheduling the Process Based on Prediction

Once the CPU usage is predicted for the process, the scheduling decision is made based on this prediction.

```
if predicted_cpu < 50:

    print(f"Process {metric['pid']}: Scheduled
(Predicted CPU: {predicted_cpu:.2f})")

else:

    print(f"Process {metric['pid']}: Deferred
(Predicted CPU: {predicted_cpu:.2f})")
```

Here:

- If the predicted CPU usage is below the threshold (e.g., 50%), the process is scheduled (started).
- If the predicted CPU usage exceeds the threshold, the process is deferred, meaning it won't be executed until resources become available.

5. Summary of CPU Usage Prediction Workflow:

- **Input Data:** The system gathers the process metrics (ram, disk, network) and prepares them for the model.
- **Prediction:** The trained Random Forest model predicts the CPU usage for the given process based on these features.
- **Scheduling Decision:** Based on the predicted CPU usage, the process is either scheduled or deferred, preventing system overload and optimizing resource usage.

6. Impact of CPU Usage Prediction

The primary purpose of this prediction is to optimize system resource management. By predicting CPU usage in advance, the system can make proactive decisions, reducing the chances of CPU bottlenecks or overloading by ensuring that only processes with manageable resource demands are executed concurrently. This leads to better overall system efficiency, minimized delays, and improved throughput.

Predictive CPU Scheduling Simulation

This Python script simulates predictive CPU scheduling, where processes are scheduled based on their predicted CPU usage and priority. It integrates machine learning (Random Forest Regressor) to predict CPU demands and employs a priority-based scheduling mechanism to ensure high-priority tasks are executed promptly. Here is how the code works:

1. **Prediction and Scheduling**:
    - A pre-trained **Random Forest Regressor** model predicts CPU usage for each process based on its resource usage, such as RAM, disk usage, and network usage.
    - The predicted CPU usage is compared against a defined threshold (e.g., 50%).
    - Processes with predicted CPU usage below the threshold are eligible

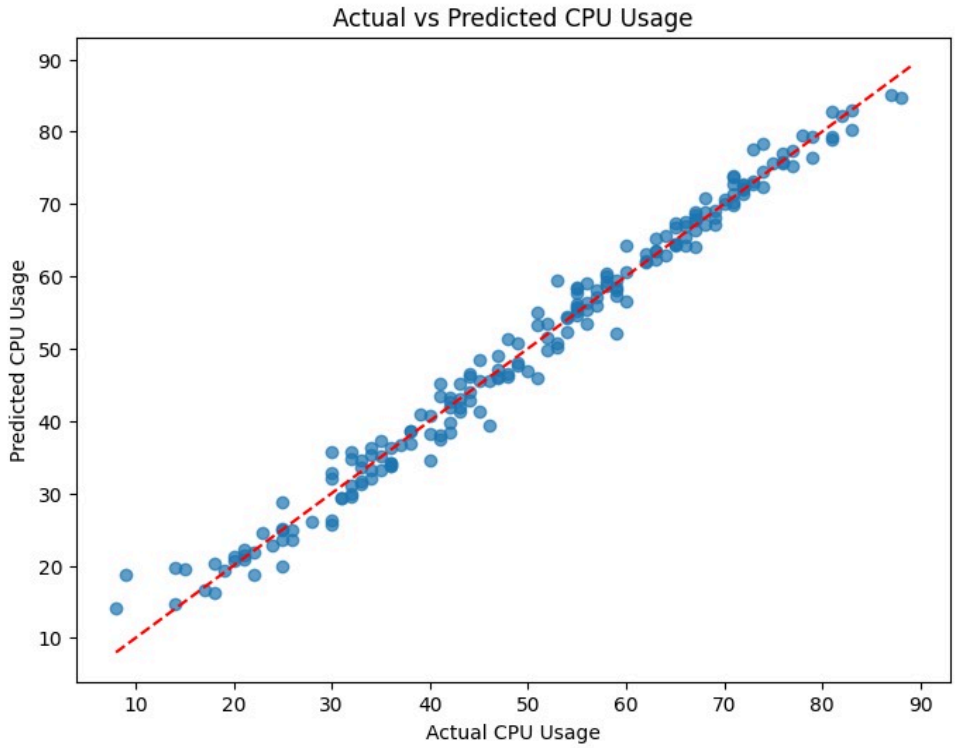
for execution, while others are deferred.

2. **Priority-Based Scheduling**:
    - Each process is assigned a priority level (higher priority indicated by a lower numerical value).
    - Among processes eligible for execution, the scheduler prioritizes those with higher priority levels, ensuring critical tasks are executed before less critical ones.
3. **Parallel Execution**:
    - The system employs Python's `multiprocessing` library to execute processes concurrently, mimicking real-world CPU behavior.
    - Each process executes a computational workload for a random duration while adhering to the scheduling order determined by prediction and priority.
4. **Real-Time Monitoring**:
    - The system continuously monitors and evaluates processes in real-time.
    - Predictions and scheduling decisions are dynamically updated based on the system state and process resource demands.

Key Insights and Usage

1. **Predictive Scheduling**:
    - Anticipates CPU demands, helping prevent overloading and optimizing resource utilization.
    - Ensures efficient decision-making by deferring processes likely to cause high CPU usage.
2. **Priority-Based Scheduling**:
    - Critical tasks are executed promptly, aligning with real-world requirements where certain processes (e.g., system updates, user-facing applications) are more urgent than others.
3. **Efficient Resource Management**:
    - By combining machine learning predictions with priority-based scheduling, the system balances resource usage while meeting priority constraints.
4. **Realistic Simulation**:
    - The use of multiprocessing reflects real-world CPU behavior, making the simulation more practical and insightful for systems-level design and testing.

RESULTS


Actual vs Predicted CPU Usage
Predicted CPU Usage
Actual CPU Usage
10
20
30
40
50
60
70
80
90

```
Starting Predictive CPU Scheduling Simulation...

Simulated Processes:
   pid  ram  disk  network
0    1   20    12        1
1    2   26    11        3
2    3   27    17        1
3    4   25    16        5
4    5   25    25        4

Monitoring and Scheduling Processes...
Process 1: Deferred (Predicted CPU: 86.79)
Process 2: Deferred (Predicted CPU: 86.79)
Process 3: Deferred (Predicted CPU: 86.80)
Process 4: Deferred (Predicted CPU: 85.73)
Process 5: Deferred (Predicted CPU: 76.26)
```

```
Starting Predictive CPU Scheduling Simulation...

Simulated Processes:
   pid  ram  disk  network
0    1   22    28        5
1    2   10    10        8
2    3   29    19        4
3    4   12    27       10
4    5   16    21        9

Monitoring and Scheduling Processes...
Process 1: Scheduled (Predicted CPU: 49.93)
Process 1 started with RAM: 22 MB, Disk: 28 MB, Network: 5 MB/s
Process 2: Scheduled (Predicted CPU: 38.75)
Process 2 started with RAM: 10 MB, Disk: 10 MB, Network: 8 MB/s
Process 3: Deferred (Predicted CPU: 53.34)
Process 4: Scheduled (Predicted CPU: 39.06)
Process 4 started with RAM: 12 MB, Disk: 27 MB, Network: 10 MB/s
Process 5: Scheduled (Predicted CPU: 42.58)
Process 5 started with RAM: 16 MB, Disk: 21 MB, Network: 9 MB/s
Process 1 finished.
Process 2 finished.
Process 4 finished.
Process 5 finished.

All processes completed.
```

- **Processes 1, 2, 4, and 5** are scheduled because their predicted CPU usage is below 50%.
- **Process 3** is deferred as its predicted CPU usage is above 50%.

After the scheduling decisions, the processes start executing based on the workload defined earlier. For each scheduled process, a computationally intensive task is performed for a random duration (between 2 and 5 seconds).

CONCLUSION

This project underscores the feasibility and importance of predicting CPU usage in operating systems. By leveraging machine learning models, it is possible to forecast CPU demands accurately and enhance resource allocation strategies. Simulations demonstrated the practical benefits of these predictions, showcasing their role in optimizing system performance.

Overall, this project highlights the potential of integrating machine learning with system resource management to improve the performance and efficiency of computer systems. It provides a solid foundation for exploring more advanced scheduling algorithms and resource management strategies that could be employed in larger-scale, real-time applications. Future improvements could involve incorporating more sophisticated models, handling more complex resource interdependencies, and testing the system in varied real-world scenarios.

Future Work

- **Real-Time Integration**: Develop models that can predict CPU usage in real time for dynamic system adaptation.
- **Advanced Algorithms**: Explore reinforcement learning or hybrid models to improve prediction accuracy and responsiveness.
- **Scalability**: Test the models on large-scale distributed systems to assess their effectiveness in diverse environments.