Técnicas de aprendizaje automático para sistemas operativos

Jose Ortiz Padilla, Diego Castañeda Ossa.

Resumen - Este proyecto analiza y optimiza la programación de procesos en sistemas operativos mediante la implementación de algoritmos tradicionales como FCFS, SJF, STCF y RR, comparándolos con un enfoque moderno basado en aprendizaje. Se evalúa su desempeño a través de métricas como tiempo de turnaround y tiempo de respuesta, con visualizaciones como diagramas de Gantt para entender su comportamiento.

Índice de Términos - Scheduling, sistemas operativos, optimización, turnaround, algoritmos, planificación.

I. Introducción

La programación de procesos es una tarea crítica en sistemas operativos, ya que permite gestionar la asignación de recursos, especialmente la CPU, entre múltiples procesos concurrentes. La eficiencia de esta asignación afecta directamente el rendimiento del sistema, la experiencia del usuario y la capacidad para manejar cargas variables.

Los algoritmos clásicos de programación de procesos, como First-Come, First-Served (FCFS), Shortest Job First (SJF), Shortest Time-to-Completion First (STCF) y Round Robin (RR), han sido fundamentales en la computación durante décadas. Sin embargo, presentan limitaciones al enfrentarse a entornos dinámicos con cargas no predecibles o prioridades cambiantes.

El propósito de este proyecto es diseñar un modelo que utilice aprendizaje automático para optimizar el desempeño de la programación de procesos. Este enfoque busca superar las limitaciones de los algoritmos tradicionales, mejorando las métricas clave, como los tiempos de turnaround y respuesta, especialmente en escenarios de alta concurrencia y cargas pesadas.

II. Marco teórico

Para el desarrollo del proyecto se es necesario tener en cuenta los siguientes conceptos:

- **Algoritmos de Scheduling clásicos [1]**

    1. FCFS (First-Come, First-Served):
       Este algoritmo atiende los procesos en el orden de llegada. Aunque es simple y fácil de implementar, puede generar problemas de ineficiencia en escenarios donde procesos largos bloquean a procesos más cortos.

    2. SJF (Shortest Job First):
       Prioriza procesos con menor duración de ejecución, reduciendo el tiempo promedio de espera. Sin embargo, su dependencia en la estimación previa de los tiempos de ejecución puede causar problemas en su implementación.

    3. STCF (Shortest Time-to-Completion First):
       Variante de SJF que permite interrupciones para atender procesos más cortos, optimizando el tiempo de espera en escenarios dinámicos.

    4. RR (Round Robin):
       Divide el tiempo de CPU en segmentos iguales, conocidos como quantums. Este algoritmo es ideal para sistemas interactivos, pero el rendimiento depende en gran medida de la elección del tamaño del quantum.

- **Métricas de evaluación [1]**

    1. Turnaround Time (TT): Tiempo total desde que un proceso entra al sistema hasta que se completa.

    2. Response Time (TR): Intervalo entre la llegada de un proceso y el inicio de su ejecución.

- **Optimización mediante Aprendizaje Automático [2]**

    El aprendizaje automático permite construir modelos capaces de analizar patrones y tomar decisiones basadas en datos históricos. Para la programación de procesos, estas técnicas pueden adaptarse a las condiciones del sistema en tiempo real, priorizando procesos con base en su impacto en las métricas clave.

III. Metodología

Se observó que los algoritmos clásicos de scheduling pueden experimentar limitaciones en escenarios con alta concurrencia y cargas cambiantes. Esto puede llevar a tiempos de espera ligeramente mayores, variaciones en la eficiencia de recursos y diferencias en la atención a los procesos según las condiciones del sistema.

- **Diseño del estudio**

El presente estudio se centra en la comparación y análisis de los principales algoritmos de planificación de procesos en sistemas operativos mediante simulaciones controladas. Se seleccionaron los algoritmos First Come First Served (FCFS), Shortest Job First (SJF), Shortest Time to Completion First (STCF) y Round Robin (RR) para evaluar su desempeño bajo diferentes métricas clave.

- **Definición del entorno de simulación**

  Lenguaje y Herramientas:

  1. Se utilizará un lenguaje de programación (como Python) para implementar los algoritmos.

  2. El entorno de simulación se diseñará para emular una cola de procesos con parámetros definidos como tiempo de llegada, tiempo de ejecución e identificador.

- **Implementación de algoritmos clásicos**:

  1. Se implementaron los algoritmos FCFS, SJF, STCF y RR como base de comparación.

- **Simulación de escenarios**:

  1. Generación de procesos con parámetros variables: tiempo de llegada, duración e identificador .

  2. Diseño de escenarios con diferentes niveles de carga y concurrencia para probar el desempeño.

- **Optimización del Scheduling**:

  1. Se utilizó un modelo de aprendizaje por refuerzo basado en Proximal Policy Optimization (PPO), implementado con la librería Stable Baselines3. Este modelo interactúa con un entorno de simulación personalizado (ProcessSchedulingEnv), diseñado específicamente para este proyecto.

     Durante el entrenamiento, el modelo aprende a optimizar el orden de ejecución de procesos, superando las limitaciones de los algoritmos clásicos en escenarios dinámicos y de alta concurrencia.

- **Métricas analizadas:**:

  1. TT: Comparar el tiempo promedio total de los procesos.

  2. TR: Evaluar la capacidad de respuesta rápida del sistema.

IV. Implementación

La implementación del proyecto se desarrolló íntegramente en Google Colab, utilizando el lenguaje de programación Python. Todo el código está contenido en un único archivo llamado Proyecto_Final.ipynb, que debe ejecutarse en orden para reproducir los resultados.

- **Ejecución del proyecto**

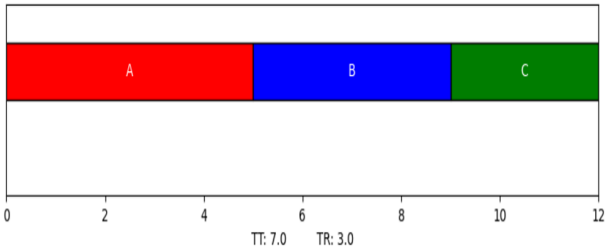
  1. Primero, se deben instalar las dependencias indicadas en la sección de configuración del modelo. Es posible que sea necesario reiniciar el entorno al instalar las dependencias.

  2. Posteriormente, el archivo se puede ejecutar de manera secuencial para generar los resultados esperados.

- **Visualización de resultados**

  1. Se implementaron diagramas de Gantt para visualizar el historial de ejecución de los procesos, facilitando el análisis y comparación de los algoritmos clásicos y el modelo basado en aprendizaje por refuerzo.

  2. Los diagramas muestran claramente cómo se distribuyen los tiempos de ejecución y las interrupciones entre los procesos, destacando las diferencias en eficiencia entre los enfoques.



V. Protocolo de experimentación

El protocolo de experimentación se diseñó para garantizar resultados confiables y reproducibles mediante las siguientes estrategias:

- **Recopilación de datos**

  1. La simulación de procesos generó datos estructurados como el historial de ejecución

de procesos, tiempos de inicio y finalización, y métricas clave (TT y TR).

2. Los datos se recopilaron automáticamente durante la ejecución de los algoritmos y el modelo en el entorno de simulación.

- **Procesamiento de datos**

1. **Python**: Se utilizó para implementar simulaciones y el modelo de aprendizaje automático, facilitando cálculos precisos y manipulación de datos.

2. Librerías

a. **NumPy**: Para cálculos matemáticos y estadísticos en las simulaciones.

b. **Matplotlib**: Para la visualización de resultados mediante diagramas de Gantt.

- **Validación de datos**

1. Los resultados de cada experimento se validaron comparando las métricas obtenidas (TT y TR) con los valores esperados basados en teorías y características de los algoritmos clásicos.

2. Para el modelo de aprendizaje automático, se monitorean las recompensas durante el entrenamiento para confirmar que el modelo estaba aprendiendo patrones coherentes y mejorando su desempeño.

VI. Resultados

Los resultados obtenidos se analizaron bajo dos métricas principales: Turnaround Time (TT) y Response Time (TR). Estas métricas se calcularon para cada algoritmo clásico (FCFS, SJF, STCF, RR) y el modelo basado en aprendizaje por refuerzo (PPO).

- **Conclusiones verificables y reproducibles**

1. El modelo basado en aprendizaje por refuerzo demostró un desempeño superior en escenarios dinámicos, donde los algoritmos clásicos enfrentan limitaciones.

2. La capacidad del modelo PPO de optimizar TT y TR simultáneamente lo convierte en una solución robusta y adaptable.

- **Fuentes potenciales de error**

1. Dado que el modelo PPO utiliza aprendizaje por refuerzo, puede haber variabilidad en los resultados dependiendo del estado inicial del modelo y el entorno.

2. La simulación utilizada asume un comportamiento ideal de los procesos (sin interrupciones externas ni fallos). En un sistema real, factores como latencia de hardware o eventos impredecibles podrían afectar el desempeño.

VII. Conclusiones

En general, este proyecto resalta el potencial del aprendizaje automático como una herramienta importante en el diseño y la optimización de sistemas operativos modernos.

- Los algoritmos tradicionales (FCFS, SJF, STCF y RR) son efectivos en escenarios controlados, donde las características de los procesos son predecibles y constantes. Sin embargo, presentan limitaciones significativas en situaciones dinámicas con alta concurrencia y cargas variables.

- El modelo basado en Proximal Policy Optimization (PPO) mostró una capacidad superior para adaptarse a estos escenarios, optimizando las métricas clave de turnaround time (TT) y response time (TR).

- La incorporación de técnicas de aprendizaje por refuerzo permite no solo mejorar el rendimiento en términos de TT y TR, sino también equilibrar estas métricas para obtener una planificación más eficiente y equitativa.

- Este proyecto demuestra que las técnicas modernas de aprendizaje automático pueden complementar y, en algunos casos, superar a los algoritmos clásicos en la programación de procesos. Esto abre nuevas posibilidades para la implementación de soluciones inteligentes en sistemas operativos.

VIII. Referencias

[1] R. H. Arpaci-Dusseau y A. Arpaci-Dusseau, *Operating System Three Easy Pieces*, versión 1.01, 2018. [En línea]. Disponible: www.ostep.org.

[2] OpenAI, "Proximal Policy Optimization," [En línea]. Disponible: https://spinningup.openai.com/en/latest/algorithms/ppo.html. [Último acceso: 5 de diciembre de 2024].