INFORME TÉCNICO SOBRE LA IMPLEMENTACIÓN DEL ALGORITMO GENÉTICO PARA LA ASIGNACIÓN DE TAREAS A SERVIDORES

Nombre y apellidos: Alfredo Mituy Okenve Obiang

Asignatura: Sistemas Inteligentes

Profesor: Francisco Chávez de la O.

Curso: 2024/2025

Fecha: martes, 5 de noviembre de 2024

Introducción

El objetivo de este proyecto es desarrollar un algoritmo genético para optimizar la asignación de un conjunto de tareas a varios servidores, minimizando el consumo energético y manteniendo un equilibrio en el uso de los recursos, así como respetando la prioridad de cada tarea. Este tipo de optimización es crucial en sistemas con recursos limitados, ya que permite mejorar la eficiencia del hardware y reducir el consumo energético.

Descripción del Problema

Cada tarea tiene unos requisitos específicos de CPU y memoria, y una prioridad (Baja, Media o Alta). Por otro lado, los servidores tienen características distintas en cuanto a:

- Capacidad de CPU (en núcleos).

- Capacidad de memoria (en GB).

- Consumo energético en función de la carga (bajo y alto consumo).

El algoritmo genético debe encontrar la asignación óptima de tareas a servidores, de manera que:

1. Minimice el consumo energético en función de la carga de cada servidor.

2. Evite la sobrecarga de recursos (CPU y memoria).

3. Respete las prioridades de las tareas, penalizando las asignaciones subóptimas en cuanto a la prioridad.

Estructura del Algoritmo

El algoritmo genético se implementa en varias fases, con diferentes métodos de selección, cruce, mutación y reemplazo. Esto permite comparar múltiples configuraciones y evaluar cuál es la más efectiva.

1. Representación de la Solución (Individuo)

Cada individuo representa una posible solución, que en este caso es una asignación de todas las tareas a los servidores. Se representa como una lista en la que cada elemento indica el servidor al que se asigna una tarea específica.

Por ejemplo, si tenemos 3 servidores y 5 tareas, un individuo podría verse así:

$$\boldsymbol{individuo} = [\mathbf{1}, \mathbf{3}, \mathbf{2}, \mathbf{1}, \mathbf{3}]$$

En este caso:

- La **Tarea 1** se asigna al **Servidor 1**.

- La **Tarea 2** se asigna al **Servidor 3**.

2. Inicialización de la Población

La función ***inicializar_poblacion*** genera una población inicial de tamaño ***pop_size***, donde cada individuo es una lista de asignaciones aleatorias de tareas a servidores.

3. Función de Fitness

La función de fitness es el núcleo de la evaluación de los individuos. Calcula el valor de aptitud basado en tres aspectos:

- **Consumo Energético:** Calcula el consumo de cada servidor en función de su carga de CPU.

  - Si el uso de CPU de un servidor está por debajo del 75% de su capacidad, se considera en bajo consumo.

  - Si el uso supera el 75%, el servidor pasa a alto consumo.

- **Sobrecarga de Recursos:** Se penalizan las asignaciones que sobrecarguen un servidor (superando su capacidad de CPU o memoria).
- **Prioridad de Tareas:** Las tareas de prioridad alta y media reciben penalizaciones adicionales si la asignación no cumple con sus requisitos.

Implementación de la Función de Fitness

```
def fitness(individuo, tareas, servers, penalizacion_alta, penalizacion_media):
    total_energia = 0
    total_carga = [{'cpu': 0, 'memoria': 0} for _ in servers]  Carga por servidor
    penalizacion_prioridad = 0  Penalización por prioridades no atendidas

    for i, tarea in enumerate(tareas):
        server_index = individuo[i] - 1  Ajustar el índice del servidor
        server = servers[server_index]

        Acumular carga en el servidor
        total_carga[server_index]['cpu'] += tarea['cpu']
```

```
total_carga[server_index]['memoria'] += tarea['memoria']

 Calcular consumo energético basado en la carga
if total_carga[server_index]['cpu'] <= server['cpu'] * 0.75:
  total_energia += server['bajo_consumo']
else:
  total_energia += server['alto_consumo']

 Penalización por prioridad
if tarea['prioridad'] == 'Alta':
  penalizacion_prioridad += penalizacion_alta
elif tarea['prioridad'] == 'Media':
  penalizacion_prioridad += penalizacion_media

 Penalización por sobrecarga de CPU o memoria
penalizacion_sobrecarga
        = sum(1 for carga, server in zip(total_carga, servers)
          if carga['cpu'] > server['cpu'] or carga['memoria']
        > server['memoria'])

 Fitness final
fitness_value
        = 1 / (1 + total_energia + penalizacion_sobrecarga
        + penalizacion_prioridad)
return fitness_value * 10000
```

- Resultado: El fitness se maximiza cuando el consumo energético es bajo, las sobrecargas son mínimas y las tareas de alta prioridad se cumplen adecuadamente.

4. Operadores Genéticos

Selección

Se implementan cuatro métodos de selección que deciden qué individuos pasan a la siguiente generación:

1. Ruleta: Selección probabilística basada en el fitness.

2. Torneo: Selecciona el mejor individuo de un grupo aleatorio.

3. Ranking: Selecciona individuos en función de su posición en el ranking de fitness.

4. Truncamiento: Selecciona solo los mejores individuos.

Cruce

Se implementan dos métodos de cruce:

1. Cruce de un punto: Intercambia genes a partir de un punto aleatorio.

2. Cruce multipunto: Realiza el cruce entre dos puntos seleccionados aleatoriamente.

Mutación

Se implementan dos métodos de mutación:

1. Reasignación aleatoria: Cambia el servidor asignado a una tarea con cierta probabilidad.

2. Intercambio de genes: Intercambia las asignaciones de servidores entre dos tareas.

Reemplazo

Se implementan dos métodos de reemplazo:

1. Reemplazo Generacional: Reemplaza toda la población actual con la nueva generación.

2. Reemplazo Estacionario: Solo reemplaza un porcentaje de la población, manteniendo los mejores individuos.

Ejecución del Algoritmo

La función principal **algoritmo_*genetico*** implementa el ciclo de evolución del algoritmo genético durante un número de generaciones definido. En cada generación:

1. Evalúa el fitness de cada individuo.

2. Selecciona pares de padres y aplica los operadores de cruce y mutación para generar descendientes.

3. Reemplaza parte o toda la población según el método de reemplazo elegido.

4. Almacena el mejor fitness y el fitness promedio de cada generación para el análisis posterior.

Manejo de Restricciones

1. Penalización por Sobrecarga: La función de fitness penaliza cualquier asignación que sobrecargue los recursos de CPU o memoria de los servidores.

2. Penalización por Prioridad: Las tareas con prioridad alta y media reciben penalizaciones en la función de fitness si no se asignan de forma adecuada.

3. Consumo Energético: La función de fitness minimiza el consumo energético asignando tareas a servidores con baja carga.

Estas penalizaciones son claves para guiar el algoritmo hacia soluciones que no solo minimicen el consumo energético, sino que también respeten las restricciones de recursos.

Resultados Obtenidos

Configuraciones Probadas

Se probaron múltiples configuraciones variando los métodos de selección, cruce, mutación, reemplazo, así como las penalizaciones de prioridad y las probabilidades de cruce y mutación. Cada configuración fue evaluada en términos del mejor individuo y el fitness medio a lo largo de las generaciones.

Análisis de Resultados

1. Convergencia:

- La convergencia del mejor fitness fue más rápida en configuraciones con selección de torneo y reemplazo estacionario, que favorecen la preservación de los individuos más aptos.

- El cruce multipunto combinado con el método de mutación de intercambio produjo una mayor variabilidad, retrasando la convergencia, pero explorando más el espacio de soluciones.

2. Consumo Energético:

- Configuraciones con penalizaciones de prioridad más altas lograron reducir el consumo energético en las tareas de alta prioridad, al asignarlas de forma más eficiente a los servidores.

- La combinación de mutación por reasignación y reemplazo generacional resultó en configuraciones con menor consumo energético, debido a la variabilidad introducida.

3. Diversidad de Soluciones:

- La selección por ranking ayudó a mantener una mayor diversidad en las primeras generaciones, pero fue menos efectiva para encontrar soluciones óptimas en configuraciones con menos generaciones.

Visualización de Resultados

Los gráficos generados en ***grafica_comparativa.py*** muestran:

- Mejor Fitness por Configuración: Permite observar cómo cada configuración afecta la convergencia hacia la solución óptima.

- Fitness Medio por Configuración: Muestra la estabilidad y diversidad de la población a través de las generaciones.

Conclusiones

El algoritmo genético desarrollado ofrece una solución flexible y eficaz para la asignación de tareas en servidores heterogéneos, permitiendo probar diferentes configuraciones para optimizar el consumo energético y el uso de recursos. La implementación de métodos variados de selección, cruce, mutación y reemplazo proporciona un marco sólido para ajustar el equilibrio entre exploración y explotación en el algoritmo genético.

Este proyecto podría expandirse en el futuro para incluir otros factores, como el tiempo de ejecución o la carga de red, y optimizar aún más el sistema de asignación de tareas en centros de datos complejos.