José Javier Hurtarte Hernández 19707
Algoritmos y estructuras de datos

Estrategia reducción tiempo y análisis posterior

Lo que se observó al analizar los datos de estrategias para disminución de tiempo separadas por intervalo de tiempo de llegada fue, que en general la desviación estándar tiende a disminuir al momento de reducir el intervalo de llegada de los procesos. Se observó también que el tiempo promedio de la estrategia de aumentar la capacidad de memoria tiende a ser ligeramente mayor a la estrategia por defecto, pero luego la estrategia de aumento de memoria cuando se utilizan mas procesos posee un tiempo menor a la corrida de forma estándar, el incremento de tiempo se debe a que al inicio con pocos datos para lograr guardar y sacar un dato se debe de recorrer toda la memoria, la cual es de capacidad grande, para terminar la acción. Luego se vio que con mayor cantidad de procesos el tiempo disminuye debido a que esto evita como un tipo “Bottle neck”, porque los datos no se acumulan en cola para poder entrar a la memoria, causando así una reducción de tiempo. Finalmente se observa que las estrategias de doble procesador son las que poseen menor tiempo y desviación, aunque la estrategia de procesador rápido tendía a ser más rápida que la de doble procesador. Por lo tanto, al haber observado los comportamientos anteriores se llegó a dos estrategias para la reducción de tiempo basada en recursos.

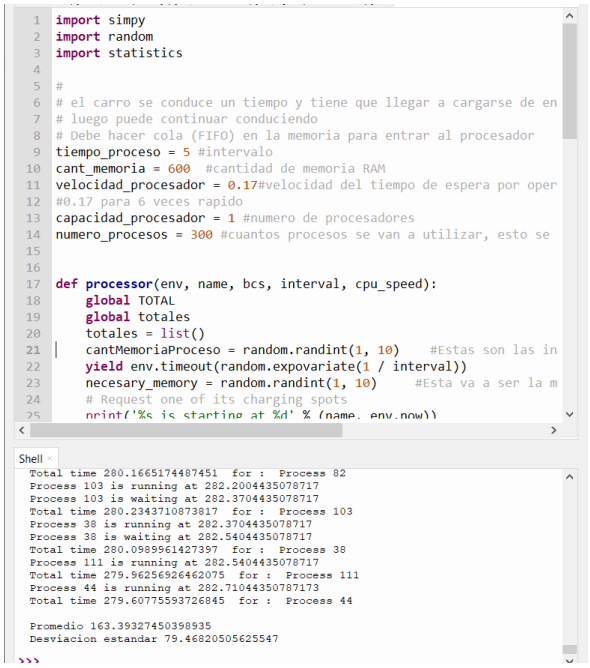
La primera estrategia esta diseñada para ser optima al inicio y al final sin tener que gastar demasiados recursos (dinero), esta involucra tener un doble procesador y una memoria grande ya que con esto se puede emplear hasta cierto punto un tipo de paralelismo el cual nos permitiría realizar múltiples operaciones al mismo tiempo, con esto se aumenta el rendimiento del tiempo globalmente, pero para balancear y disminuir aun mas el tiempo cuando se tienen muchos procesos se debe utilizar más memoria para poder trabajar de una manera eficiente grandes cantidades de procesos sin tener que estos deban de realizar cola para poder entrar a la memoria.

```python
import simpy
import random
import statistics


#
# el carro se conduce un tiempo y tiene que llegar a cargarse de en
# luego puede continuar conduciendo
# Debe hacer cola (FIFO) en la memoria para entrar al procesador
tiempo_proceso = 1 #intervalo
cant_memoria = 200  #cantidad de memoria RAM
velocidad_procesador = 1 #velocidad del tiempo de espera por operac
#0.17 para 6 veces rapido
capacidad_procesador = 2 #numero de procesadores
numero_procesos = 300 #cuantos procesos se van a utilizar, esto se


def processor(env, name, bcs, interval, cpu_speed):
    global TOTAL
    global totales
    totales = list()
    cantMemoriaProceso = random.randint(1, 10)    #Estas son las in
    yield env.timeout(random.expovariate(1 / interval))
    necesary_memory = random.randint(1, 10)     #Esta va a ser la m
    # Request one of its charging spots
    print('%s is starting at %d' % (name, env.now))
```

```
Shell
Process 53 is running at 803.0033306685398
Total time 800.3993765316808  for :  Process 202
Process 236 is running at 804.0000295460078
Total time 800.8886856170029  for :  Process 53
Process 44 is running at 804.0033306685398
Process 236 is running at 805.0000295460078
Process 44 is running at 805.0033306685398
Process 236 is running at 806.0000295460078
Total time 799.8295299264768  for :  Process 44
Process 236 is waiting at 807.0000295460078
Total time 802.3107155090026  for :  Process 236

Promedio 417.43153874836423
Desviacion estandar 235.44970009495992
```

(Ejemplo de corrida de primera estrategia)

La segunda estrategia esta diseñada suponiendo un limite inexistente de recursos. Aquí se toma la utilización de un procesador más rápido con una memoria grande proporcional a la velocidad del procesador para evitar el bottle neck, se dice que es suponiendo que no hay limite de recursos ya que para obtener un procesador 6 veces más rápido a lo convencional significa un gasto fuerte de dinero. El procesador rápido causa que los datos se procesen de forma secuencial 6 veces más rápido, y para disminuir la variación de esta combinación y el tiempo con grandes cantidades de procesos se usa una memoria proporcional grande. Debido a lo anterior explicado se concluye que esta sería la estrategia más rápida.



```
import simpy
import random
import statistics


#
# el carro se conduce un tiempo y tiene que llegar a cargarse de en
# luego puede continuar conduciendo
# Debe hacer cola (FIFO) en la memoria para entrar al procesador
tiempo_proceso = 5 #intervalo
cant_memoria = 600  #cantidad de memoria RAM
velocidad_procesador = 0.17#velocidad del tiempo de espera por oper
#0.17 para 6 veces rapido
capacidad_procesador = 1 #numero de procesadores
numero_procesos = 300 #cuantos procesos se van a utilizar, esto se


def processor(env, name, bcs, interval, cpu_speed):
    global TOTAL
    global totales
    totales = list()
    cantMemoriaProceso = random.randint(1, 10)    #Estas son las in
    yield env.timeout(random.expovariate(1 / interval))
    necesary_memory = random.randint(1, 10)     #Esta va a ser la m
    # Request one of its charging spots
    print('%s is starting at %d' % (name, env.now))
```

```
Shell
 Total time 280.1665174487451  for :  Process 82
 Process 103 is running at 282.2004435078717
 Process 103 is waiting at 282.3704435078717
 Total time 280.2343710873817  for :  Process 103
 Process 38 is running at 282.3704435078717
 Process 38 is waiting at 282.5404435078717
 Total time 280.0989961427397  for :  Process 38
 Process 111 is running at 282.5404435078717
 Total time 279.9625692646275  for :  Process 111
 Process 44 is running at 282.7104435078717
 Total time 279.60775593726845  for :  Process 44

 Promedio 163.39327450398935
 Desviacion estandar 79.46820505625547
>>>
```

(Ejemplo de corrida de segunda estrategia)

Debido a que es una simulación de algo existente, el análisis y estrategias se tomaron desde el punto de vista como que si yo fuese contratado para dar estrategias de reducción de tiempo