COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

Práctica 1. Sistemas Distribuidos

Elena Conderana Medem y Sergio Cuenca Núñez

Tecnologías de Datos Masivos
Big Data Technology

Índice

Introducción

1. Contexto

La computación distribuida es un paradigma informático, donde un conjunto de ordenadores organizados en clústeres trabaja de manera coordinada como si se tratara de un único ordenador. Esta arquitectura distribuye la carga de trabajo entre distintos nodos, permitiendo resolver problemas de mayor magnitud y complejidad en menores intervalos de tiempo.

Por este motivo su uso está muy expandido en áreas como el Big Data o cálculos computacionalmente intensos. En este ámbito se encuentra el método de Montecarlo. Un método computacional que por medio de muestreos aleatorios reiterados obtiene resultados numéricos para aproximar soluciones a problemas matemáticos.

2. Descripción del problema

La práctica se centra entorno a calcular el valor de π con el método de Montecarlo en un sistema distribuido. El método de Montecarlo permite aproximar el valor de π generando aleatoriamente puntos dentro de una superficie cuadrada y computando la cantidad de dichos puntos que se ubican en el interior de un círculo inscrito en el cuadrado.

La bondad de la predicción es proporcional al número de muestras que se utilicen. Para alcanzar precisiones razonables se requiere un gran número de muestras, que se puede traducir en un problema computacionalmente costoso. Motivo por el cual se propone el uso de un sistema distribuido que alivie la carga sobre un único nodo y permita paralelizar la ejecución.

3. Objetivos

Los objetivos principales de esta práctica comprenden:

- Implementar un sistema distribuido para realizar el cálculo de π usando el método de Montecarlo.
- Diseñar e implementar el sistema distribuido por medio de un modelo maestro-esclavo, con 1 maestro y 2 esclavos.
- Usar sockets para comunicar a lo nodo maestro con los nodos esclavo.
- Utilizar hilos para paralelizar tareas concurrentes en los distintos esclavos desde el nodo maestro.

4. Justificación

Los siguientes motivos justifican la relevancia de implementar el cálculo de π con un sistema distribuido:

- La precisión del método de Montecarlo es directamente proporcional a las simulaciones realizadas. Sin embargo, el incremento del número de muestras para alcanzar resultados precisos conlleva un incremento de la carga computacional. La computación distribuida permite ejecutar cada simulación independientemente en nodos diferentes.

- La reducción en el tiempo de cálculo puede conducir a soluciones más precisas, pues se pueden realizar más simulaciones en el mismo intervalo de tiempo. Mayor número de simulaciones implica incremento de la precisión del método.

- El sistema puede escalar fácilmente mediante la inclusión de más nodos, que permitan aumentar la capacidad de cómputo. Esta característica es interesante si se buscan soluciones muy precisas de π con coincidencias hasta el undécimo decimal, por ejemplo.

Metodología

En la siguiente sección se procede a describir en detalle cómo se ha implementado la solución para el cálculo de π mediante un sistema distribuido, abordando cuestiones técnicas, sobre el proceso de desarrollo y sobre las tecnologías utilizadas.

5. Descripción del entorno de desarrollo

La implementación de la práctica se ha llevado a cabo en el lenguaje de programación Python. Un lenguaje de alto nivel, que destaca por su sencillez y su versatilidad. Cuenta con una amplia extensión de librerías, que extienden la funcionalidad base del lenguaje con colecciones de funciones, clases y métodos predefinidos. En esta solución concretamente se han empleado 6 librerías.

- `random`: Módulo que implementa generadores de números pseudoaleatorios para distintas distribuciones. En concreto se ha empleado dentro de la función de Montecarlo para generar valores de x e y aleatorios.
- `threading`: Módulo que permite la paralelización mediante hilos la ejecución de un programa. Dichos hilos se han usado para paralelizar el trabajo en los distintos esclavos.
- `socket`: Módulo que proporciona acceso a la interfaz BSD socket, permitiendo establecer conexiones de red a través de distintos protocolos, como TCP o UDP, para transmitir y recibir datos. Esta funcionalidad permite al master conectarse con los distintos esclavos.
- `sys`: Módulo que proporciona parámetros y funciones específicos para interactuar con el interpretador del sistema.
- `time`: Módulo que proporciona funciones para acceder y realizar conversiones relacionadas con el tiempo. Se ha utilizado para introducir un retardo de 2 segundos entre la conexión máster-esclavos y la ejecución del problema de Montecarlo y para medir dicho tiempo de ejecución.
- `matplotlib`: Biblioteca completa para realizar visualizaciones estáticas, animadas e interactivas. Se han realizado dos gráficos estáticos para visualizar el tiempo de ejecución frente al número de muestras, y el tiempo de ejecución frente al número de esclavos.

Además de estas características Python es un lenguaje multiplataforma. En concreto se ha empleado el entorno Google Colab para desarrollar la práctica. Un servicio alojado de Jupyter Notebook que provee acceso gratuito a recursos incluyendo GPUs y TPUs.

6. Diseño de la solución

El sistema adopta una arquitectura maestro-esclavo para distribuir la carga de trabajo para el cómputo de π entre distintos servidores. La comunican entre los componentes principales del sistema, es decir, el maestro y los esclavos se realiza mediante sockets TCP.

El maestro se conecta a los puertos de escucha de cada esclavo y se encarga de distribuir la carga de cálculo entre ellos de manera uniforme, recopilar secuencialmente sus resultados parciales, gestionar las conexiones

con los mismos y realizar la estimación final de π. Por su parte el esclavo ejecuta el cálculo de π mediante el método Montecarlo para el número de muestras recibidas del maestro y envía el resultado a este de vuelta.

El funcionamiento de la arquitectura descrita tiene 4 partes constituyentes. En la inicialización se definen el total de muestras que se van a emplear para calcular π y el número de esclavos y sus respectivos puertos. Estos se inician quedando a la espera de conexiones por parte del maestro.

A continuación, el maestro divide el total de muestras entre los distintos esclavos y establece una conexión TCP con cada uno, enviando la cantidad de muestras asignadas al esclavo en concreto a través de dicha conexión.

Tras la recepción del número de muestras por parte de los esclavos, estos comienzan con la ejecución del método de Montecarlo para estimar la cantidad de puntos ubicados dentro del círculo unitario y enviar dichos resultados al maestro.

El maestro recopila los resultados parciales de los distintos esclavos. Una vez obtenidos cierra la conexión TCP con ellos y los esclavos cesan de escuchar en su puerto. Finalmente, el maestro calcula la estimación de π mediante la siguiente fórmula:

$$\pi_{estimate} = \left(\frac{\sum partial\ results}{N}\right) \times 4$$

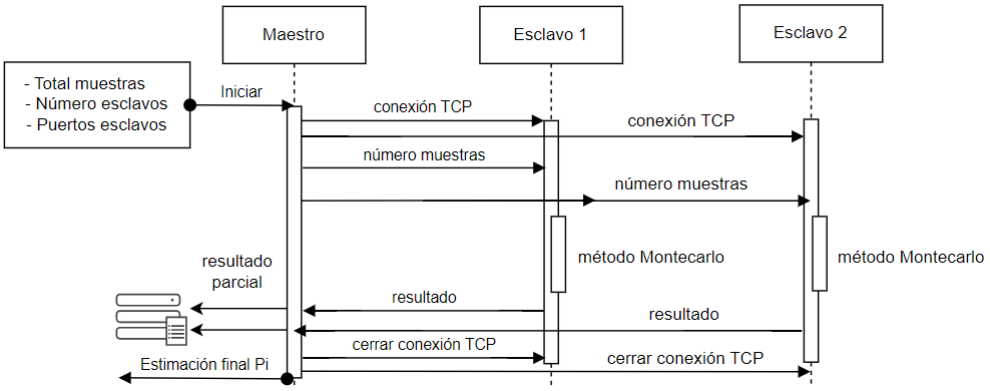

Figura 1. Diagrama de flujo de la arquitectura

7. Pruebas realizadas

Para comprobar la bondad del sistema distribuido desarrollado se propone un análisis de rendimiento para estudiar qué impacto tienen el número de muestras y la cantidad de esclavos en el tiempo de ejecución de la solución.

El primer análisis se realiza únicamente con 2 esclavos y se computa el tiempo de ejecución para 1.000.000, 1.250.000, 1.500.000 y 2.000.000 muestras. En el segundo análisis las muestras se limitan a 1.000.000 y se estudia el tiempo de ejecución con 1, 2, 3 y 4 esclavos. Idealmente, la precisión de las estimaciones debería aumentar conforme se incrementa el número de muestras, a pesar de que el tiempo de ejecución también se incremente.

Así mismo, al aumentar el número de esclavos el tiempo de ejecución debería disminuir sin impacto perceptible en la precisión de las estimaciones de π.

Resultados

A continuación, se procede a estudiar de manera independiente los resultados obtenidos durante la ejecución de la práctica atendiendo a las dos casuísticas propuestas.

8. Descripción del entorno de desarrollo

La primera resolución realizada por el sistema corresponde a la estimación de π en base a un número cambiante de muestras. La Figura 2 muestra cómo el tiempo de ejecución aumenta linealmente conforme aumenta el número de muestras. En la Tabla 1 esta tendencia no es tan evidente. Si se obvia el error cometido con 1.500.000 muestras, este seguiría también una tendencia lineal, pero de carácter inverso al tiempo de ejecución. No obstante, este error no se considera insignificante y se analizará en la discusión de resultados.

Nº muestras	Predicción π	Error
1.000.000	3,143508	0,00192
1.250.000	3,143152	0,00156
1.500.000	3,143392	0,00180
2.000.000	3,140786	0,00081

Tabla 1. Error en estimación de π con 4 muestras distintas.

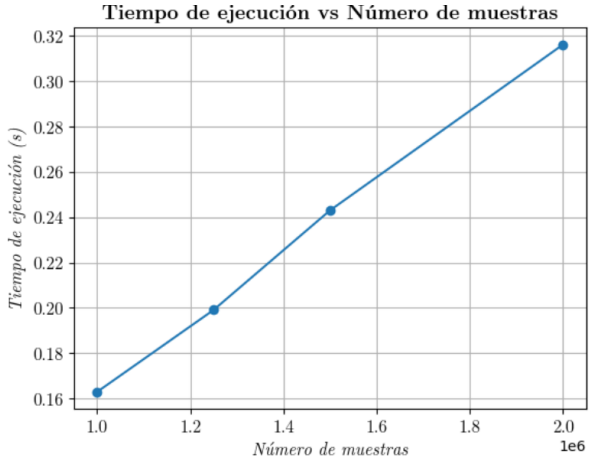

Figura 2. Tiempo de ejecución con 2 esclavos y cuatro muestras distintas.

En la segunda resolución se alcanza una estimación muy cercana a π con un error del 0,005% cuando la ejecución se realiza con dos esclavos. A diferencia del caso anterior, en la Figura 3 se aprecia un descenso del tiempo de ejecución conforme aumenta el número de esclavos con forma de sigmoide inversa. Cabe destacar que si se aumentara el número de muestras y de esclavos, probablemente la tendencia sería de carácter lineal. En la Tabla 2 se muestra la evolución de los errores en las predicciones, que se minimizan con 2 y 3 esclavos.

Nº esclavos	Predicción π	Error
1	3,142708	0,00112
2	3,141544	0,00005
3	3,142096	0,00050
4	3,139624	0,00197

Tabla 2. Error en estimación de π con 1.000.000 muestras y distinta cantidad de esclavos.

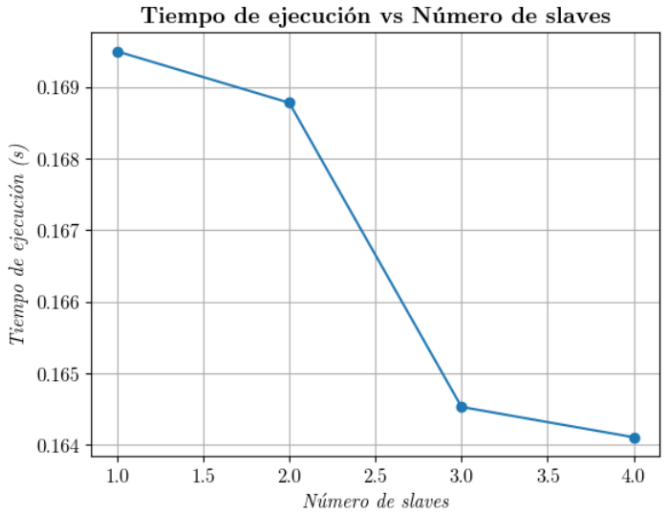

Figura 3. Tiempo de ejecución con 1.000.000 muestras y distinta cantidad de esclavos.

9. Pantallazos de la Ejecución

En este apartado se muestra la ejecución de la solución propuesta. En la Figura 4 se aprecian las conexiones entre el par de esclavos y el maestro, y la estimación final del valor de π para un número ascendente de muestras. El primer resultado equivale a 1.000.000 muestras.

```
Slave esperando tareas en ('127.0.0.1', 9999)...
Slave esperando tareas en ('127.0.0.1', 10000)...
Master conectado a slave en ('localhost', 10000)
Master conectado a slave en ('localhost', 9999)
Slave en ('127.0.0.1', 10000) conectado con ('127.0.0.1', 50219)
Slave en ('127.0.0.1', 9999) conectado con ('127.0.0.1', 50220)
Estimación de Pi: 3.143508
Slave esperando tareas en ('127.0.0.1', 9999)...
Slave esperando tareas en ('127.0.0.1', 10000)...
Slave en ('127.0.0.1', 9999) conectado con ('127.0.0.1', 50223)
Master conectado a slave en ('localhost', 9999)
Master conectado a slave en ('localhost', 10000)
Slave en ('127.0.0.1', 10000) conectado con ('127.0.0.1', 50224)
Estimación de Pi: 3.143152
Slave esperando tareas en ('127.0.0.1', 9999)...
Slave esperando tareas en ('127.0.0.1', 10000)...
Master conectado a slave en ('localhost', 9999)
Master conectado a slave en ('localhost', 10000)
Slave en ('127.0.0.1', 9999) conectado con ('127.0.0.1', 50226)
Slave en ('127.0.0.1', 10000) conectado con ('127.0.0.1', 50225)
Estimación de Pi: 3.143392
Slave esperando tareas en ('127.0.0.1', 9999)...
Slave esperando tareas en ('127.0.0.1', 10000)...
Master conectado a slave en ('localhost', 9999)
Slave en ('127.0.0.1', 10000) conectado con ('127.0.0.1', 50229)
Master conectado a slave en ('localhost', 10000)
Slave en ('127.0.0.1', 9999) conectado con ('127.0.0.1', 50228)
Estimación de Pi: 3.140786
```

Figura 4. Ejecución con 2 esclavos y cuatro muestras distintas

La Figura 5 presenta una ejecución análoga a la indicada en la Figura 4. Sin embargo, el caso de uso en este caso es distinto. El número de muestras permanece invariable en 1.000.000 y la cantidad de esclavos sobre los que se apoya el maestro asciende secuencialmente de 1 a 4.

```
Slave esperando tareas en ('127.0.0.1', 9999)...
Master conectado a slave en ('localhost', 9999)
Slave en ('127.0.0.1', 9999) conectado con ('127.0.0.1', 50230)
Estimación de Pi: 3.142708
Slave esperando tareas en ('127.0.0.1', 9999)...
Slave esperando tareas en ('127.0.0.1', 10000)...
Master conectado a slave en ('localhost', 9999)
Slave en ('127.0.0.1', 9999) conectado con ('127.0.0.1', 50231)
Master conectado a slave en ('localhost', 10000)
Slave en ('127.0.0.1', 10000) conectado con ('127.0.0.1', 50232)
Estimación de Pi: 3.141544
Slave esperando tareas en ('127.0.0.1', 9999)...
Slave esperando tareas en ('127.0.0.1', 10000)...
Slave esperando tareas en ('127.0.0.1', 10001)...
Master conectado a slave en ('localhost', 9999)
Master conectado a slave en ('localhost', 10000)
Master conectado a slave en ('localhost', 10001)
Slave en ('127.0.0.1', 10001) conectado con ('127.0.0.1', 50236)
Slave en ('127.0.0.1', 9999) conectado con ('127.0.0.1', 50234)
Slave en ('127.0.0.1', 10000) conectado con ('127.0.0.1', 50235)
Estimación de Pi: 3.142096
Slave esperando tareas en ('127.0.0.1', 9999)...
Slave esperando tareas en ('127.0.0.1', 10000)...
Slave esperando tareas en ('127.0.0.1', 10001)...
Slave esperando tareas en ('127.0.0.1', 10002)...
Master conectado a slave en ('localhost', 10001)
Master conectado a slave en ('localhost', 9999)
Slave en ('127.0.0.1', 10001) conectado con ('127.0.0.1', 50237)
Master conectado a slave en ('localhost', 10002)
Slave en ('127.0.0.1', 9999) conectado con ('127.0.0.1', 50238)
Master conectado a slave en ('localhost', 10000)
Slave en ('127.0.0.1', 10002) conectado con ('127.0.0.1', 50239)
Slave en ('127.0.0.1', 10000) conectado con ('127.0.0.1', 50240)
Estimación de Pi: 3.139624
```

Figura 5. Ejecución con 1.000.0000 muestras y rango ascendente de esclavos (1-4).

10. Discusión de los resultados

En la ejecución con un distinto número de muestras la conclusión no parece evidente en base a los datos recopilados, pues se esperaba que el error en la predicción descendiese paralelamente con el número de muestras. Si bien este fenómeno se ha cumplido en la mayoría de los casos, con 1.500.000 muestras se ha alcanzado un error superior al esperado. De manera tan aislada es complicado inferir el origen. Sería recomendable iterar el proceso un número determinado de veces y calcular la media, pues el error podría deberse a la secuencia de números aleatorios obtenidos para esa muestra.

El tiempo de ejecución, sin embargo, evoluciona linealmente con el aumento del número de muestras, coincidiendo con las hipótesis iniciales, pues un aumento en el número de muestras conlleva un incremento en las iteraciones del método de Montecarlo.

Para la estimación de π con distinto número de esclavos las mejores aproximaciones se alcanzan con un número intermedio de esclavos, 2 o 3. Al no tratarse de un cálculo que requiere mucho tiempo de ejecución el descenso

en tiempo de ejecución se mide en el orden de microsegundos. Si se extrapolara a ordenes de magnitudes superiores, donde en lugar de contemplar microsegundos se tratará de minutos u horas sería necesario realizar un análisis de coste. La decisión residiría en seleccionar 2 o 3 esclavos. Si bien 2 esclavos tienen un tiempo de ejecución superior a 3 y ligeramente inferior a 1, el error obtenido es del 0,005%. Con 3 esclavos se sacrifica precisión a favor de un mayor rendimiento.

Los resultados de esta ejecución son factibles dentro de lo esperado. Por un lado, el aumento en cantidad de esclavos no implica una mayor precisión en los resultados, sino un descenso en los tiempos de ejecución. Además, el descenso de los tiempos de ejecución no tiene que ser lineal, pues conforme crece el número de esclavos trabajando en paralelo, descienden en mayores órdenes de magnitud los tiempos de ejecución.

Conclusión

Por último, se van a resumir los principales hallazgos y aprendizajes obtenidos durante la práctica, y la relevancia de la solución implementada.

11. Resumen del Proceso

Para la implementación de un sistema distribuido para el cálculo de π utilizando el método de Montecarlo se han seguido los siguientes pasos.

1. Identificación del problema: Implementar el cálculo de π mediante el método de Montecarlo utilizando un sistema distribuido.
2. Diseño de la arquitectura e implementación: Desarrollo de la arquitectura basada en maestros y esclavos en el lenguaje de programación Python. Los códigos base de maestro y esclavo se han ampliado para que el maestro pueda gestionar una mayor cantidad de esclavos y estos implementen la funcionalidad del cálculo de π.
3. Pruebas y análisis: Ejecución del código de manera reiterada para comprobar su correcto funcionamiento e inclusión de análisis de rendimiento en base a muestras y esclavos.
4. Análisis de rendimiento: Visualización de los resultados para entender las relaciones entre la bondad de las estimaciones, los parámetros del método y las métricas de ejecución del sistema.

12. Principales Logros

La solución implementada demuestra la viabilidad del cálculo distribuido de π utilizando el método de Montecarlo. Para concluir se van a recorrer los logros más importantes alcanzados en base a los resultados que se han analizado en la sección anterior.

- Se ha logrado desarrollar una **solución funcional** que calcula el número π con el método de Montecarlo utilizando un sistema distribuido.
- Alcanzar una precisión de π con un error del 0,005%.
- Inferir que para extraer el rendimiento real es necesario realizar numerosas ejecuciones del método de Montecarlo y promediarlas, pues 2 esclavos y 1.000.000 muestras han dado errores muy dispares de valor 0,00005 y 0,00192.
- Conseguir reducciones no lineales del tiempo de ejecución con aumentos lineales del número de esclavos.
- Ampliar el conocimiento sobre conceptos de **programación de sockets y empleo de hilos** para gestionar ejecuciones concurrentes.