Facultad de ingeniería
Departamento de TIC
Arquitectura de software

Tarea Final Arquitectura de Software
Cálculo de Pi con el Método de Simulación de Montecarlo

Christian David Flor Astudillo
Gabriel Cruz Libreros
Natalia Isabel Gonzalez Murillo

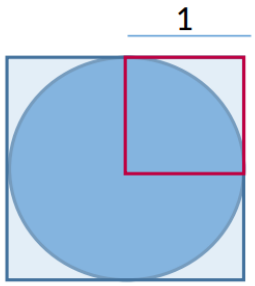
Universidad Icesi
Facultad de Ingeniería, Departamento de TIC
Cali, Colombia
2021

Introducción

Cálculo de π con el Método de Simulación de Montecarlo

Conociendo las fórmulas para el cálculo del área del cuadrado y el círculo, y usando un generador de datos aleatorios uniforme, es posible calcular el valor de pi(π).

Imagínense un cuadrado de lado 2 unidades de medida y un círculo inscrito, que tendría entonces un radio de 1 unidad de medida:



El área de ese cuadrado es entonces de 2x2 = 4 unidades cuadradas, y el del círculo de $\pi * 1^2 = \pi$.

Consideremos un sistema orto normal, cuyo origen coincide con el centro del círculo. Tenemos entonces la ecuación del círculo: $x^2 + y^2 <= 1$. Todo punto de coordenadas (x_p, y_p) del cuadrante superior derecho que se encuentre dentro del círculo tendrá como propiedad $x_p^2 + y_p^2 <= 1$.

Si solo tenemos en cuenta ese cuadrante, y obtenemos aleatoriamente puntos dentro de ese cuadrante siguiendo una distribución uniforme, todos los puntos estarán dentro del área de ese cuadrante del cuadrado, pero no todos estarán dentro del área del círculo. Pero al ser el cuadrado y el círculo simétricos, y al tener el mismo baricentro, las proporciones del área del círculo y del área del cuadrado se conservan en el cuadrante.

De manera general, se observa la siguiente proporción:

$\pi * l^2$ → $(2*l)^2$
puntos en el círculo → puntos totales

y despejando π, obtenemos:

$\pi = 4 * l^2 * ($ puntos en el círculo $) / (l^2 *$ puntos totales $)$

El objetivo de este trabajo es diseñar un sistema de software que calcule π a partir de tirajes aleatorios uniformes tal y como lo acabamos de describir. Este método se conoce como simulación de Montecarlo, y es una forma estática de simulación.

Estructura del documento

En este documento se encuentra la explicación de dos versiones para resolver la problemática planteada. La primera versión es la más sencilla, pues solo cuenta con un único nodo de procesamiento que realiza todo el trabajo. Mientras que la segunda versión es una solución distribuida en diferentes nodos y tipos de nodos, para garantizar una respuesta más rápida sin saturar los recursos físicos de las máquinas.

Cada versión tiene una serie de test, donde cada uno consta de dos números, uno que indica la semilla para realizar el aleatorio y otro que indica el número de puntos a ubicar en el círculo. Además, cada una cuenta con el link GitHub donde se encuentra el código de la solución junto con los pasos para ejecutarlo, y su respectivo diagrama de deployment.

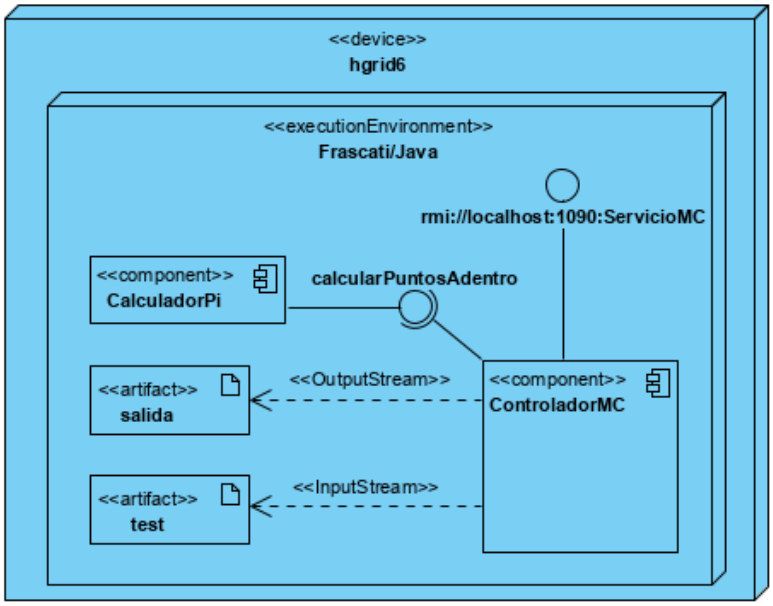
Primera versión

Link de github: https://github.com/ChristianFlor/MonteCarlo_Pi

Solución

El planteamiento básico de la solución consta de un único nodo de procesamiento, el cual se encarga de mostrar una interfaz de usuario, leer los casos de prueba, realizar la ubicación aleatoria de los puntos, calcular el valor de Pi y la escritura de un archivo, que contiene los resultados de Pi con el tiempo que tarda el procesamiento para el cálculo de Pi. Además de calcular el tiempo promedio de ejecución para cada prueba y de Pi.

Diagrama de deployment



Facultad de ingeniería
Departamento de TIC
Arquitectura de software

Segunda versión

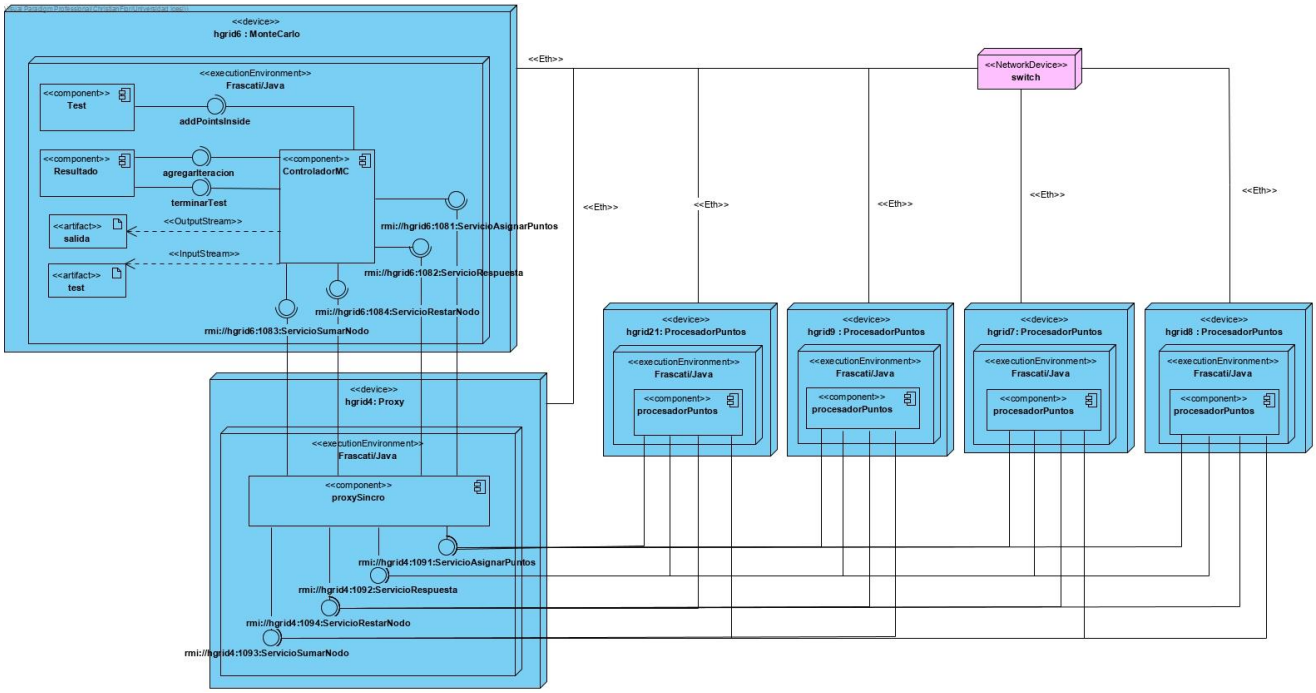
Link de github: https://github.com/ChristianFlor/MonteCarloPiDistribuido

Solución

Para esta versión se utilizó 3 tipos de nodos de procesamiento. El primero (MonteCarlo) encargado de realizar el cálculo de Pi, mostrar la interfaz de usuario, leer los casos de prueba y escribir los resultados. El segundo (ProcesadorPuntos) es el encargado de realizar el procesamiento de los puntos, esto quiere decir que es quien lanza los puntos de forma aleatoria y decide si está dentro o fuera del círculo. Además, este tipo de nodo se puede instanciar varias veces para reducir la carga de procesamiento y el tiempo de respuesta. Finalmente el tercer nodo (ProxySincro) es quien se encarga de sincronizar las llamadas realizadas por las diferentes instancias del nodo ProcesadorPuntos para evitar conflictos y pérdida de información.

Esta solución hace uso de la estructura de procesamiento SISD multi-nstanciado dado que todos los nodos de procesamiento están ejecutando la misma instrucción, además la cantidad de puntos que procesa cada nodo procesadorPuntos es la misma. Por otro lado, la estructura de procesamiento es UMA, debido a que el nodo MonteCarlo es el único con acceso a los datos y es el encargado de pasar la información a los nodos de tipo procesadorPuntos. Adicional a lo anterior, se hace uso del patrón de diseño Proxy con la variante de sincronización.

Diagrama de Deployment



Anexos

Realizamos este diagrama de clases como parte del proceso de diseño, para un mejor entendimiento de problema