# **Proyecto Final**

**Arquitectura de Software**

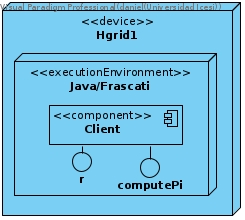
## Versión 1: Monolítico

Para esta versión nos enfocamos solo en el cálculo de PI por medio de dos métodos:.

El primero, era el contabilizador de los puntos dentro del círculo, que luego retornaba esos puntos como salida. El segundo, era el encargado de ejecutar la función del cálculo PI en esta ocasión como trabajamos en un círculo de radio 1, tal función se vería así:

Al ser esta solución monolítica se detectaban problemas al momento de ejecutar pruebas con entradas muy grandes, esto lo identificamos al ver que al asignarle una entrada de 10^9 ya no computarizada más o demoraba un tiempo demasiado grande. La razón por la que consideramos sucedió esto fue debido a que los métodos que se implementaron eran de una complejidad de . Por lo anterior, las capacidades computables no daban abasto o salía muy costoso resolverlo.

### Diagrama de deployment



## 

## Versión 2: Distribuido

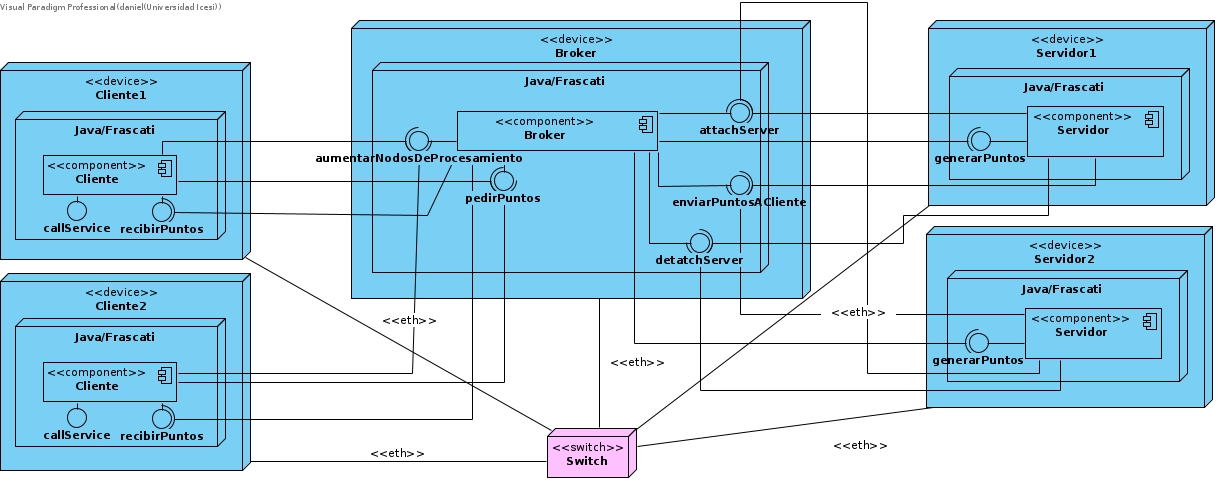
Para esta versión nos enfocamos en calcular pi con un mayor número de puntos, ya que la versión monolítica solo pudo con una petición de hasta 100.000.000 puntos, y esa petición era muy pequeña, la razón por la cual deducimos esto fue a través de los conocimientos obtenidos en el curso. Por eso decidimos una solución con un sistema distribuido, en donde usamos un cliente-servidor, en donde el tipo de procesamiento que usamos es *SIMD* porque el servidor solo usaba una instrucción (la de procesar los puntos aleatorios e incrementar la variable de aquellos que estuvieran dentro del círculo) pero los diferentes servidores usaban datos diferentes (multiple data). No se hizo elección de estructura de almacenamiento debido a que no realizamos un guardado de datos como tal, solo accedemos a entradas.

En la solución, el cliente no tiene ninguna dependencia del servidor para poder inicializarse, además tiene un blocksize por defecto que ayuda a la división de cuántos puntos va a recibir el servidor para contar. Por parte del servidor, si tiene una dependencia hacia el cliente (no se puede iniciar un servidor si no hay un cliente corriendo), además expone el servicio de generar puntos, en esta versión el servidor solo puede estar conectado a un solo cliente y un cliente puede tener muchos servidores conectados a él. El cálculo lo realiza el cliente cuando ya todos los servidores han acabado de contar los puntos y los han retornado.

Además esta solución cuenta con una interfaz gráfica. La cual permite agregar una entrada individual, compuesta de cantidad de puntos y semilla, como también con la carga de múltiples entradas por medio de un archivo CSV. Esta interfaz también muestra los resultados y los nodos que tiene conectado. Por otro lado, también crea un .txt con la salida, cantidad de nodos utilizados y tiempo implementado.

Esta solución a diferencia de la versión 1, es mejor debido a que permite hacer cálculos con entradas mucho más grandes, tales como 10^10, 10^12,etc. Debido a que ahora toda la responsabilidad no la tiene un solo nodo, sino que se reparte entre todos los posibles que estén conectados. Permitiendo un procesamiento más ágil y mejor distribuido en la cantidad de puntos.

### Diagrama de deployment



### Diseño de la interfaz gráfica

