Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Cuadro de Mandos para Visualizar Algoritmos Distribuidos

Autor: Jan Cerezo Pomykol
Tutor: Fernando Pérez Costoya

Madrid, Abril - 2022

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Cuadro de Mandos para Visualizar Algoritmos Distribuidos

Abril - 2022

Autor: Jan Cerezo Pomykol
Tutor: Fernando Pérez Costoya
Departamento de Arquitectura y Tecnología de Sistemas Informáticos
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Cuando se habla de sistemas distribuidos, una de las propiedades fundamentales de los algoritmos de esta disciplina es que se ejecutan en múltiples máquinas (nodos) concurrentemente. Estos nodos interactúan de alguna manera entre ellos para lograr un objetivo determianado. Por lo general, la simultaneidad de los eventos causa dificultades a los estudiantes de la materia a la hora de entender el funcionamiento de estos algoritmos puesto que es complicado mantener una visión global del estado del sistema.

Este proyecto tiene como objetivo principal proporcionar una herramienta que facilite la comprensión del funcionamiento de los algoritmos de la asignatura de Sistemas Distribuidos. A diferencia de otras implementaciones que simplemente visualizan una simulación el algoritmo, en esta se pretende visualizar una ejecución real. De esta forma se podrá también ofrecer la posibilidad de interactuar con los nodos en tiempo de ejecución, pudiendo observar cómo el algoritmo reacciona ante distintas situaciones. Además de ofrecer una solución que facilita el aprendizaje en la asignatura de Sistemas Distribuidos, también se podría emplear como una herramienta de depuración de estos algoritmos.

Abstract

When it comes to distributed systems, one of the most important aspects of the algorithms is that they are executed simultaneously in various independent machines, commonly referred as nodes. These nodes interact with each other to achieve some objective. The simultaneousness of the events usually causes difficulties to the students when they try to understand the behavior of the algorithm. This is due to the fact that it is hard to maintain a global vision of the state of the distributed system.

The main goal of this project is to provide a tool that facilitates the understanding of the algorithms of the distributed system discipline. Contrary to other existing implementations that only simulate a visualization of the algorithm, this one will visualize a real execution, permitting real-time interaction with the nodes. This allows to visualize how the algorithm reacts in different situations. Additionally, this tool provides a debugging environment for this type of algorithms.

Tabla de contenidos

Capítulo 1

Introducción

Uno de los aspectos más dificultosos para los estudiantes de la disciplina de sistemas distribuidos es comprender el modo de operación de los algoritmos propuestos en esta materia. La simultaneidad de los eventos en los nodos dificulta mantener una comprensión del estado global del algoritmo. A esto se le suma complejidad que supone la caída de nodos o el comportamiento anómalo de estos. Este proyecto proporciona una herramienta que facilita la comprensión del estado del algoritmo mediante la visualización de los nodos del sistema y los mensajes entre estos. Esta herramienta no sólo puede ser empleada con fines docentes, si no también con fines de depuración de algoritmos distribuidos.

1.1. Trabajos previos

En la actualidad ya existen numerosas implementaciones que logran este tipo de visualizaciones. No obstante, se pueden clasificar en dos conjuntos:

1. Las que ejecutan una simulación. Los nodos del sistema distribuido no son "reales", si no que se simula el comportamiento con fines meramente visuales. Algunos ejemplos de este tipo son [1][2][3].

2. Las que visualizan una ejecución real. En estas implementaciones los nodos sí son procesos reales que se ejecutan localmente o en un servidor. El entorno de visualización captura, de alguna manera, los mensajes entre nodos para visualizar el estado de cada uno. Un ejemplo de este tipo es [4].

Sin embargo, en lo que se refiere a las implementaciones del primer tipo, la ejecución *no real* del algoritmo no proporciona una representación realista del algoritmo, puesto que no se visualiza ningún proceso en tiempo de ejecución. Estas implementaciones por lo general tienen como objetivo único la visualización de la simulación de un algoritmo concreto, por lo que el algoritmo está integrado de alguna forma en la lógica de la aplicación. En estos casos se cumpliría el objetivo de facilitar la comprensión de los algoritmos, pero estos entornos presentan varias restricciones:

- Por lo general únicamente visualizan un algoritmo concreto, y este está fuertemente integrado con la interfaz. Por este motivo resulta complicado reemplazar el algoritmo en caso de que se quiera visualizar otro.
- La ejecución simulada de los eventos podría no representar los escenarios de comportamiento anómalo de nodos por el comportamiento impredecible de la red. La resistencia de fallos de los algoritmos distribuidos es una de las características más importantes de estos. Además suele ser de gran dificultad comprender este aspecto, por tanto es importante que el entorno sea capaz de visualizar cualquier tipo de situación que se pueda producir en una ejecución real.

Estas restricciones motivan, en parte, la creación de entornos del segundo tipo, que visualizan una ejecución real de un algoritmo. Estos tienen las siguientes ventajas comparados con los del primer tipo:

- La implementación de la interfaz de visualización del algoritmo suele ser independiente del propio algoritmo que se visualiza. Este desacoplamiento permite intercambiar el algoritmo a visualizar fácilmente sin tener que considerar muchos de los aspectos programáticos para visualizarlo.
- Dado que se muestra una ejecución real el entorno está implícitamente capacitado para representar cualquier situación que se pueda presentar.

1.2. Objetivos del proyecto

Teniendo en cuenta los puntos mencionados, se puede deducir que lo ideal es contar con un entorno de visualización con las siguientes características:

- Visualiza una ejecución real.
- La implementación del algoritmo es independiente de la implementación de la interfaz. En otras palabras, intercambiar el algoritmo a visualizar es sencillo.
- Se permite al usuario interactuar con los nodos, pausando o deteniendo su ejecución en tiempo real desde la interfaz.

Existen implementaciones que cumplen algunas de estas características, por ejemplo [4]. Esta permite visualizar una ejecución real, sin embargo no permite interactuar manualmente con cada nodo. Además, el diseño de la arquitectura de la implementación no permite un desacople completo del algoritmo con la interfaz. Para visualizar un algoritmo en este entorno es necesario implementar el algoritmo a visualizar en el lenguaje *Java*, y además una serie de métodos para visualizarlo correctamente.

Por lo general, ninguna implementación actual cumple estrictamente con los tres requisitos que se mencionan. El principal objetivo de este proyecto es proporcionar un entorno que los cumpla.

El contenido del resto del documento se organiza de la siguiente forma. En el capítulo de desarrollo se explica primero la arquitectura del entorno de visualización, seguida de la explicación detallada de la implementación y la justificación de las decisiones tomadas. Seguidamente se explica un caso de uso con una implementación concreta de un algoritmo (Raft). El funcionamiento detallado de este algoritmo se explica en el Anexo. Finalmente se describen los resultados obtenidos y el impacto, verificando que se han cumplido los objetivos del proyecto.

Capítulo 2

Desarrollo

En este capítulo se describe detalladamente tanto la arquitectura de la aplicación, justificando cómo se cumplen los requisitos mencionados en la introducción, como la implementación propiamente dicha de cada uno de los elementos. También se incluye un caso de uso con un algoritmo concreto: Raft.

2.1. Análisis del problema

Como se ha mencionado anteriormente, se desea visualizar una ejecución real del algoritmo, esto implica que debe existir algún medio de comunicación entre la interfaz y los nodos del mismo. Por tanto, es necesario que la aplicación que contiene la interfaz contenga además algún recurso capaz de interceptar o capturar los mensajes que se envían los nodos durante la ejecución del algoritmo. Este proceso *manager* tiene como objetivo modificar la visualización en pantalla en base a los eventos que detecta en los nodos. Existen diversas alternativas para conseguir que reciba informacion en tiempo real de los mensajes de los nodos.

2.2. Arquitectura de la interacción

La solución más sencilla a este problema consiste en modificar la funcionalidad del algoritmo que se ejecuta en los nodos, de tal forma que durante los envíos de mensajes se envíe también una copia al proceso *manager*. En este caso el proceso *manager* sería un simple servidor que recibe mensajes de los clientes (nodos del algoritmo) y actualiza la interfaz para representar los cambios de estado. Con esto sería necesario modificar ligeramente el algoritmo que se quiere visualizar. La arquitectura de esta solución se muestra en la Figura 2.1

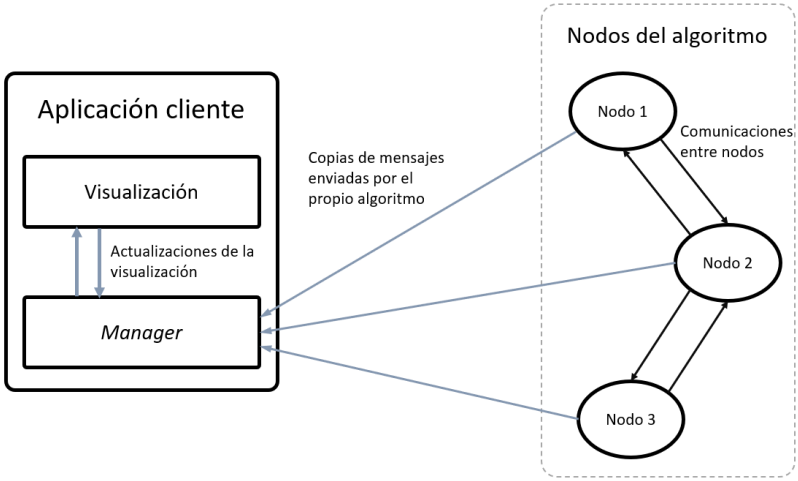

Figura 2.1: Diagrama de la primera arquitectura propuesta

En la Figura 2.1 se puede ver en la parte izquierda la aplicación cliente, que se compone de la interfaz, y del proceso *manager* que la actualiza. Ambas partes se ejecutan en la misma máquina. En la parte de la derecha se localizan los nodos del algoritmo, que pueden ejecutarse en un entorno distribuido. Las flechas entre nodos simbolizan las comunicaciones del algoritmo, y las flechas entre cada nodo y el proceso *manager* las copias de los mensajes.

Otra alternativa menos "intrusiva" en el algoritmo consiste en modificar de alguna manera las librerías que gestionan los envíos de paquetes de red. De tal forma que el algoritmo llama a la función de librería de envío de mensajes (`send`, `sendto`, `write`, etc) y esta envía una copia del mensaje al proceso *manager*. Esta solución permite no tener que modificar nada del algoritmo. La arquitectura (Figura 2.2) es muy parecida a la propuesta anteriormente, lo único que cambia es el origen del envío de las copias de mensajes.

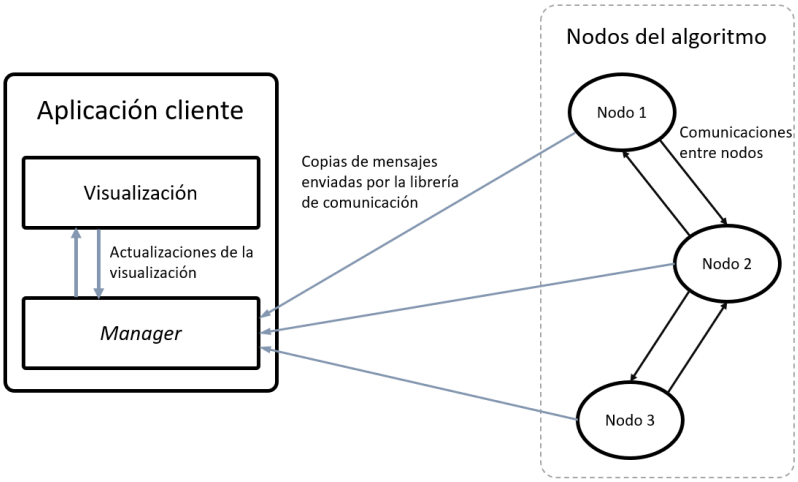

Figura 2.2: Diagrama de la segunda arquitectura propuesta

Sin embargo esta alternativa cuenta con el gran defecto de que no sería posible intercambiar el algoritmo por otro implementado en otro lenguaje de programación cualquiera. Sería necesario volver a modificar las librerías necesarias. Esta resticción, junto con el hecho de que puede ser complicado modificar la funcionalidad incluída en las librerías del lenguaje, motiva la siguiente alternativa.

La alternativa final cuenta con otro proceso capturador o *sniffer* que intercepta el tráfico de red a nivel de protocolo de un determinado proceso. Se ejecuta una instancia junto con cada nodo del algororitmo de tal forma que los mensajes capturados se reenvían al proceso *manager* para que este actualize la visualización. Esta captura de mensajes se lleva a cabo a nivel de protocolo, por lo que no es necesario modificar ningún aspecto del algoritmo que se quiere visualizar. En esta solución el cuadro de mandos se abstrae totalmente del algoritmo, de forma que permite visualizar implementaciones en distrintos lenguajes, siempre que el medio de comunicación sea un protocolo conocido, por ejemplo TCP. Esta modificación se puede observar en la figura 2.3.

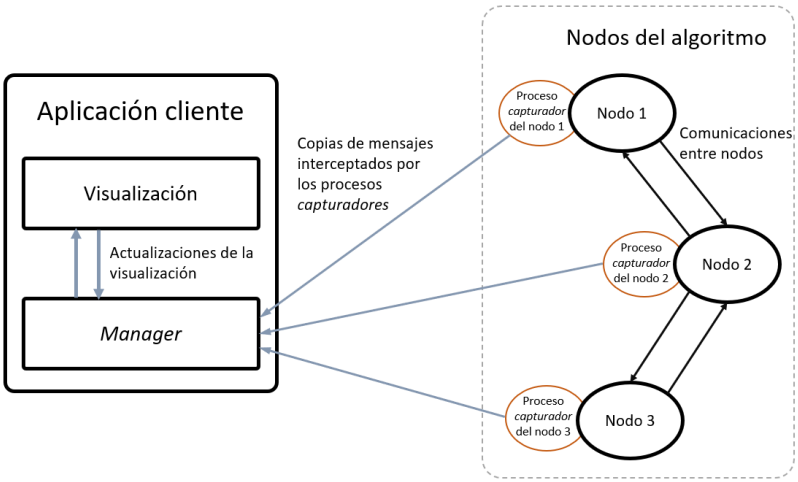

Figura 2.3: Diagrama de la tercera arquitectura propuesta

Por parte del proceso *manager*, el funcionamiento es el mismo que en las soluciones propuestas anteriormente. Otra ventaja de esta arquitectura es que soportaría también visualizar algoritmos cuyo medio de comunicación se basa en las llamadas a procedimientos remotos (RPC). En las soluciones propuestas anteriores únicamente se tienen en cuenta los algoritmos cuyo modelo de comunicación se basa en el envío de mensajes entre *sockets* (sería necesario modificar el proceso *manger* para admitir la comunicación mediante RPC). Sin embargo, dado que las librerías de llamadas a procedimientos remotos también se basan en la comunicación con algún protocolo (por lo general TCP), también se capturarían estos mensajes.

La ventaja principal de esta arquitectura comparada con las anterior, es que permite un desacoplamiento total del algoritmo. Además de esto, satisface los requisitos que se mencionan en la introducción. A continuación se explicará la implementación detallada de cada uno de los elementos de la arquitectura.

2.3. Implementación del proceso *capturador*

2.4. Implementación y descripción del cliente

2.5. Implementación del *manager*

2.6. Caso de uso

En este apartado se demuestra la funcionalidad completa de la aplicación con un algoritmo concreto: Raft. El funcionamiento de este algoritmo se puede encontrar en el Anexo de este documento. A modo de resumen, Raft es un algoritmo de consenso empleado para replicar información en conjunto de nodos.

Capítulo 3

Análisis de impacto

Capítulo 4

Conclusiones y trabajo futuro

Bibliografía

[1] D. Ongaro. The raft consensus algorithm. [Online]. Available: https://raft.github.io/

[2] B. Johnson. Raft - understandable distributed consensus. [Online]. Available: http://thesecretlivesofdata.com/raft/

[3] J. Martin. Raft distributed consensus algorithm visualization. [Online]. Available: http://kanaka.github.io/raft.js/

[4] Y. Moses, Z. Polunsky, A. Tal, and L. Ulitsky, "Algorithm visualization for distributed environments," *Journal of Visual Languages and Computing*, vol. 15, no. 1, pp. 97–123, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1045926X03000569

Anexos

Anexo I - Algoritmo Raft