Pràctica 1:
Distributed Systems

Pedro Mendoza Fernández
Gerard Panisello Gil
2023 **Parella P**

1-Introducción

En esta tarea hemos contruido un sistema de procesamiento de datos meteorológicos distribuido basado en el modelo cliente-servidor.
Los componentes de un sistema distribuido pueden comunicarse entre sí a través de diferentes patrones y tecnologías. Por lo tanto, hemos implementado dos versiones del mismo sistema: una basada en la comunicación directa entre procesos y otra basada en la comunicación indirecta. El objetivo es comparar los diferentes protocolos utilizados, analizar sus ventajas y desventajas, y discutir su idoneidad para cada tipo de comunicación.

2-Preguntas

2.1. Frame all communication steps of the system based on the four types of communication (synchronous/asynchronous, pull/push, transient/persistent, stateless/stateful). Also include their pattern and cardinality (one-to-one, one-to-all...).

Directa:

	Sinchronous/ asynchronous	Pull/ Push	Transient/ Persistent	Stateless/ Tatefull	Cardinality	Pattern
Sensors-LoadBal	Synchronous	Push	Transient	Stateless	All-to-One	In Only
LoadBal-Servers	Synchronous	Push	Transient	Stateless	One-to-All	Request-Reply
Proxy-Termina	Synchronous	Push	Transient	Stateless	One-to-All	Publisher-Subscriber

Indirecta:

	Sinchronous/ asynchronous	Pull/ Push	Transient/ Persistent	Stateless/ Tatefull	Cardinality	Pattern
Sensors-Servers	Asynchronous	Push	Transient	Stateless	All-to-One	In Only
Proxy-Terminal	Asynchronous	Push	Transient	Stateless	One-to-All	Publisher-Subscriber

2.2. Mention which communication type is more appropriate for each step and justify your decision in terms of scalability and fault tolerance.

Los tipos de comunicación que hemos usado para la parte directa:
Un sistema de comunicación síncrona, tenemos mayor escalabilidad que con un sistema asíncrono, lo que nos permite una conexión más fácil entre servidores y sensores, además con el push, hacemos que los servidores no soliciten constantemente los datos al cliente, sino que el servidor recopila constantemente los datos generados por los sensores.

Siendo el sistema es transitorio, evitamos tener que almacenar la información de los mensajes enviados. Usamos el Stateless aunque sería más apropiado usar el tatefull para mantener una conexión continua.

La parte indirecta funciona prácticamente igual, la diferencia es que tenemos un sistema asíncrono en el que los nodos funcionan de forma independiente al otro sin sincronización temporal

2.3. Are there single points of failure in the system? How could you resolve them?

Sí, hay varios puntos de fallida, por ejemplo, el más importante podría ser el LoadBalancer, ya que si hay algún error con el LoadBalancer o este falla, va a fallar consecutivamente todo el sistema que vaya por detrás. Pueden existir varios problemas y pueden ser solucionados revisando estos factores: Sobrecarga del LoadBalancer, fallo de hardware, problemas de conectividad, errores de configuración, falta de monitoreo...

2.4. Regarding system decoupling, what does a Message Oriented Middleware (MOM) such as RabbitMQ provide?

Un Middleware Orientado a Mensajes (MOM) como RabbitMQ ofrece diversas funcionalidades para el desacoplamiento del sistema.
Comunicación asíncrona: RabbitMQ permite la comunicación asíncrona entre los componentes del sistema, evitando bloqueos o esperas innecesarias.
Desacoplamiento de componentes: RabbitMQ actúa como intermediario entre productores y consumidores de mensajes, permitiendo que los componentes no necesiten conocerse directamente, sino solo la estructura del mensaje y la cola de envío o recepción.
Patrones de mensajería: RabbitMQ admite patrones como publicar-suscribir, colas de mensajes y enrutamiento de mensajes, brindando flexibilidad en la comunicación entre componentes y permitiendo escalabilidad y reutilización.
Fiabilidad: RabbitMQ garantiza la transferencia confiable de mensajes mediante técnicas como el acuse de recibo y la persistencia de mensajes.
Escalabilidad: RabbitMQ es escalable y puede manejar grandes volúmenes de mensajes, permitiendo configurar múltiples nodos o clústeres para gestionar cargas pesadas y garantizar el rendimiento y la disponibilidad del sistema.

2.5. Briefly describe Redis' utility as a storage system in this architecture.

Redis es una base de datos en memoria de clave-valor que se caracteriza por su rendimiento rápido, versatilidad en las estructuras de datos, opción de persistencia, operaciones atómicas y utilidad como sistema de caché distribuida.

En cuanto al rendimiento rápido, Redis almacena los datos en la memoria principal, lo que permite un acceso muy rápido a los valores almacenados.

En cuanto a la persistencia opcional, Redis ofrece opciones para la persistencia de datos en disco. Esto permite recuperar los datos en caso de reinicio del sistema o fallas.

Además, Redis se utiliza frecuentemente como sistema de caché distribuida. Esto implica almacenar datos en memoria para acelerar el acceso a contenido estático o datos consultados con frecuencia, reduciendo la carga en los sistemas de bases de datos.

3- Implementación

Hemos realizado dos implementaciones de modelos de comunicación en esta práctica. Hemos implementado la implementación directa utilizando gRPC y la implementación indirecta utilizando colas RabbitMQ. También hemos hecho uso del redis para almacenar datos en formato clave-valor y gestionarlos en nuestras implementaciones.

Parte directa: En la parte directa hemos decidido usar el gRPC y no el xmlRPC, ya que haciendo comparaciones vimos más ventajas al gRPC.
Los sensores envían información al Load Balancer el cual su función es repartir los datos recibidos entre todos los servidores, usamos un algoritmo que reparte la información en círculo, también llamado Round Robin.
Cuando los servidores tienen todos los datos, lo guardan en la base de datos redis.

Parte indirecta: En la parte indirecta, utilizamos colas RabbitMQ en lugar de utilizar el "load balancer" para distribuir los datos a los servidores. De esta manera, estas colas se encargan de realizar la distribución de datos.

4- Conclusiones

En esta práctica hemos aprendido muchas cosas sobre los sistemas de comunicación y los servidores. Aunque hemos adquirido la teoría sobre los dos modelos de comunicación, no hemos sido capaces de entregar prácticas que cumplan todos los requisitos. Sin embargo, hemos logrado completar las partes principales para hacerlo funcional.
En cuanto a la parte directa, podríamos haber hecho un mejor trabajo, ya que empezamos con mal pie, porque decidimos hacer una implementación con el xmlRPC, pero tuvimos demasiados problemas y perdimos mucho tiempo. Pero en general, una vez puestos con el gRPC la parte directa nos ha parecido muy curiosa y con utilidades en la vida real.
En cuanto a la parte indirecta, podríamos haber implementado la Tumbling Window, pero debido a una mala organización del trabajo no hemos podido hacerlo.
Dejando de lado las fallas, creemos que ha sido una práctica muy interesante con conceptos aplicables a la hora de pensar en cómo solucionar problemas futuros como ingenieros.