TP3: Desarrollando un sistema distribuido

Sistemas Distribuidos - Ingeniería Informática, Universidad Nacional de Mar del Plata

Versión: 1.1.1
Fecha: 15/11/2020

Revisión	Cambios realizados	Fecha
1.0.0	Creación del documento y estructura de secciones principales. Redacción de los primeros ejercicios.	23/10/2020
1.0.1	Se agregó que los suscriptores reciban todos los mensajes de la cola cuando se suscriben.	02/11/2020
1.1.0	Se agrega el ejercicio de promoción y el comportamiento de la cola de los brokers.	11/11/2020
1.1.1	Se quita la necesidad de cola de brokers en el ejercicio de promoción.	15/11/2020

Tabla de Contenidos

Introducción

El objetivo de este trabajo práctico es desarrollar un sistema distribuido simple utilizando las herramientas y conceptos aprendidos hasta el momento.

Marco teórico

Bibliografía

- Material de las clases (transparencias y libro utilizado en clase)

Enunciados

A partir de lo estudiado en clase y de la bibliografía de la materia, responda:

1. ¿Qué es un middleware en el contexto de sistemas distribuidos?
2. ¿Qué es un Middleware de Mensajes (MOM)?
3. ¿Qué patrones de intercambio de mensajes conoce?
4. ¿A qué se denomina la arquitectura de API REST? ¿Cuáles son sus características?
5. ¿Cómo compararía ZeroMQ con HTTP?
6. ¿Qué se entiende por “heartbeat” en el contexto de sistemas distribuidos?

Laboratorio: Sistema de Chat

El objetivo principal de este laboratorio es desarrollar un sistema de chat, cuyos requerimientos aparecen listados en el punto 5. Pero antes de llegar a eso, se recomienda realizar los ejercicios anteriores que le ayudarán a obtener las habilidades necesarias para programar el sistema.

1) Primeros pasos con ZeroMQ

ZeroMQ es una librería de comunicaciones orientada a mensajes, útil para el desarrollo de aplicaciones distribuidas. Opera entre la capa transporte (sobre distintos tipos de sockets) y la capa aplicación, soporta varios lenguajes, y tiene como objetivo facilitar la implementación de patrones de mensajes usados comúnmente, sin pretender ser un protocolo por sí mismo.

Una de las grandes ventajas que ofrece ZeroMQ es, en definitiva, evitar trabajar directamente con sockets cuando se quieren implementar interacciones populares por medio de mensajes como pub/sub, request/reply, etc. Ahorra una carga de trabajo significativa aportando una solución probada y que continuamente se está mejorando.

A diferencia de otras utilidades similares, ZeroMQ no provee un broker, sino que, o bien los procesos se comunican de manera directa sin intermediarios, o son los desarrolladores los encargados de programar dichos brokers.

1.1) Programando patrones de mensajes

Para entender cómo funciona, haremos unos ejercicios de programación utilizando el paquete `zeromq` de Node.js.

Se puede instalar el paquete de la siguiente forma (es importante instalar la versión 5 ya que la 6 se encuentra en *beta*):

```
$ npm install zeromq@5
```

ZeroMQ permite intercambiar mensajes de procesos utilizando diferentes protocolos. Los más comunes, y que veremos en esta práctica, son:

- REQREP: sigue el patrón request-reply
- PUBSUB: sigue el patrón publish-subscribe
- PIPELINE: sigue el patrón push-pull

En base al código proporcionado[1], ejecute y analice el comportamiento de cada uno de los protocolos. Como material para terminar de comprender dichos protocolos, debe revisar la guía de ZeroMQ[2].

1.2) Analizando un broker sencillo

Instale las dependencias del paquete “broker-sample-zmq” y ejecute el broker, el publicador y suscriptor. Estudie el código (puede ayudarse con la guía de ZeroMQ anteriormente mencionada).

Luego modifique el paquete de la siguiente manera:

- Agregue 3 publicadores y 3 suscriptores para el tópico miTopico1
- Ejecute los 4 publicadores y 4 suscriptores (y el broker) y verifique que los suscriptores reciben mensajes de los 4 publicadores
- Modifique 2 publicadores y 2 suscriptores para que funcionen con el miTopico2
- Vuelva a ejecutar todos los procesos, y verifique que los suscriptores sólo reciben del tópico correspondiente

Investigue sobre el problema del descubrimiento dinámico de un broker y cómo XSUB y XPUB resuelven esto con respecto a PUB y SUB (buscar ***dynamic discovery problem***).

[1] Archivos anexos PIPELINE.txt, PUBSUB.txt y REQREP.txt.
[2] Link: http://zguide.zeromq.org/page:all | Archivo anexo ZeroMQ_Guia.pdf.

2) Desarrollando un chat simple

Para desarrollar el siguiente sistema de chat, se aplicarán algunos conceptos vistos en la materia: patrón de mensajes publicación/suscripción, brokers, arquitecturas p2p y cliente-servidor, HTTP, REST, etc.

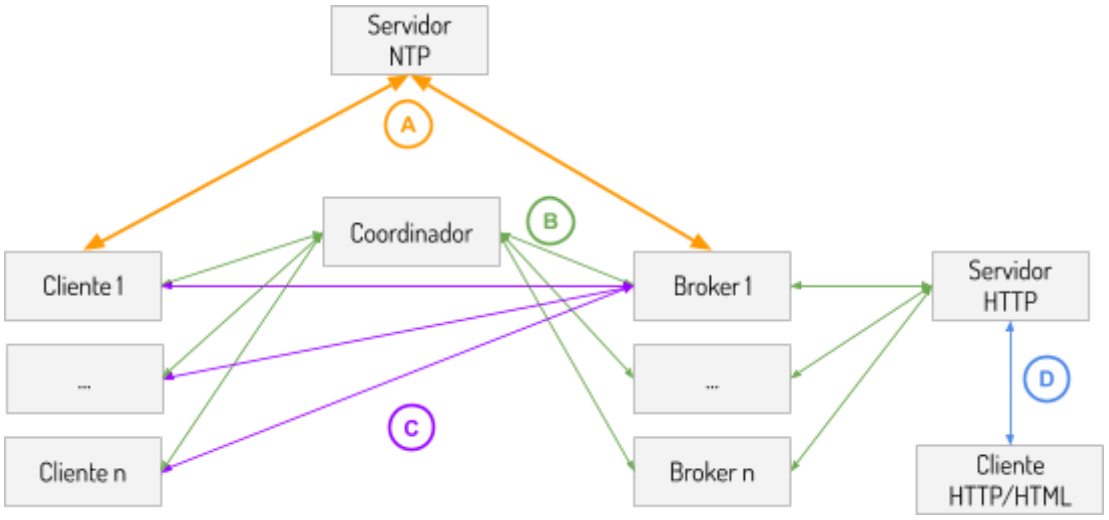
Requerimientos:

- Se debe implementar un sistema de chat utilizando Node.js, en el que cada cliente pueda escribir y recibir mensajes utilizando principalmente un patrón pub/sub.
- Los clientes podrán escribir mensajes a todos, o mensajes individuales a un cliente en particular.
- Debe permitir un número dinámico de clientes (es decir, el sistema debe funcionar con una cantidad de clientes que puede ir variando en el tiempo). Cada cliente debe tener un id único, que será pasado como argumento al iniciar el mismo.
- Los clientes enviarán un “heartbeat” periódico de cada 10 segundos, informando que están conectados. Esto servirá para que los clientes puedan llevar una lista local de quienes están conectados. Una vez que detecta un nuevo cliente, debe suscribirse al tópico correspondiente para poder recibir mensajes individuales.
- El sistema de mensajería está basado en el broker provisto en este TP pero con las siguientes modificaciones:
    - Debe permitir un número fijo de brokers a determinar por el/los alumnos (al menos 3 brokers). Cada broker tendrá asignado un id único (el id se debe pasar como argumento al iniciar el broker).
    - La primera vez que un cliente desea suscribirse o publicar en un tópico, debe solicitar a un proceso coordinador que le asigne un broker para dicho tópico.
    - Una vez asignado, el cliente utilizará los puertos de suscripción/publicación provistos por el coordinador, para suscribirse o publicar en dicho broker.
    - El proceso coordinador debe ir balanceando la cantidad de tópicos asignados a los brokers, a medida que se publican por primera vez, para que nunca haya más de 1 tópico de diferencia entre los brokers. Es decir, el coordinador debe siempre buscar que haya la misma cantidad de tópicos en cada broker.
    - El coordinador, previo a responderle al cliente la solicitud de asignación de tópico, debe informarle al broker que tiene que aceptar ese tópico. El broker deberá descartar mensajes de tópicos que le lleguen y que no tenga asignados.
    - Los tópicos disponibles puede encontrarlos en la sección de Mensajes de este documento.
    - Cada vez que haya una nueva suscripción, el cliente recibirá todos los mensajes que se encuentran en la cola de dicho tópico.
    - Las colas del broker deben retener hasta X mensajes (cantidad) y por un Y segundos (diferencia de tiempo entre el broker y el mensaje). Tanto X como Y deben ser parámetros pasados al broker como argumento al momento de ejecutarlo. Si las colas se llenan, se saca el mensaje que tenga la fecha de envío más antigua.

- Los brokers y los clientes de chat, tendrán sus relojes sincronizados por NTP utilizando como referencia el cliente-servidor realizado en el TP2.
- El sistema contará con un servidor web, que expondrá mediante una API REST para poder operar con las colas de mensajes de los tópicos. Debe permitir:
    - Debe permitir consultar la lista de tópicos asociado a un broker.
    - Debe permitir consultar la cola de mensajes actual de un tópico de un broker.
    - Debe permitir limpiar la cola de mensajes de un determinado broker.
- El sistema contará con un cliente web, hecho en HTML, al que se podrá ingresar con el navegador. Mediante un formulario interactuará con la API REST realizada en el punto anterior.
- Todos los mensajes que circulen en el sistema deberán estar formateados utilizando JSON. La estructura de cada mensaje deberá definirse entre todos en un documento compartido, para que los componentes de los distintos trabajos realizados sean compatibles e interoperables. Ver la sección Mensajes para más información.

Diagrama:

A continuación se muestra un diagrama de referencia que modela el sistema.



Mensajes:

Interfaz	Patrón/Protocolo	Tipos de mensajes
A	TCP	• Los utilizados en el práctico para sincronizar los relojes
B	REQ/REP (ZeroMQ)	• Solicitud y respuesta entre Clientes y Coordinador de dirección y puertos de broker por parte de un cliente ante la suscripción o la primera vez que se publica un tópica

		• Coordinador informa a Broker la asignación de un tópico • Servidor HTTP solicita los mensajes en la cola de cada broker
C	PUB/SUB (ZeroMQ)	• Cliente publica un mensaje al resto de los clientes utilizando el tópico message/all • Cliente publica un mensaje a otro cliente mediante message/<id cliente> • Cliente avisa su estado al resto de los clientes (“heartbeat”) mediante el tópico heartbeat
D	HTTP, REST	• Cliente solicita la lista de tópicos de un broker determinado mediante la petición GET /broker/123/topics, donde 123 es el id del broker. • Cliente solicita los mensajes en la cola de un tópico determinado mediante la petición GET /broker/123/topics/miTopico1, donde 123 es el id del broker y miTopico1 es el tópico. • Cliente solicita limpiar (remover todos los mensajes) de una cola de un tópico determinado mediante la petición DELETE /broker/123/topics/miTopico1, donde 123 es el id del broker y miTopico1 es el tópico.

El formato de cada uno de los mensajes se deberá acordar entre todos en el siguiente link: https://docs.google.com/document/d/1SVjSlVWl6HsMIcMkqV-GpjnMOzUJeQQzXNlHrz8PNps/edit?usp=sharing.

Ejercicio de promoción

Para obtener una promoción en la asignatura, el alumno deberá implementar un sistema de grupos al chat. El mismo, debe ser interoperable con el de otros grupos y a su vez, con otros sistemas que no implementen la funcionalidad de grupos. A continuación los detalles:

- Un grupo de chat es un canal de comunicación por el cual uno o más clientes pueden enviar y recibir mensajes. Si un cliente no pertenece al grupo, no puede ni enviar ni recibir mensajes a través de dicho canal.
- Un cliente podrá crear un grupo de chat ingresando el comando /group <id>, donde “id” es una cadena alfanumérica que identifica al grupo. Si dicho identificador ya existe, el cliente se unirá automáticamente al grupo luego ingresar este comando.
- Los clientes deben poder enviar y recibir mensajes en dicho grupo.
- Cada grupo utilizará un tópico, y la gestión del mismo será similar a los otros tópicos (a través de un coordinador) pero no habrá colas de mensajes.

Metodología de trabajo y evaluación

Cada grupo deberá presentar un plan de hitos de desarrollo (con fecha y alcance de cada hito). Cada lunes durante el horario de práctica, cada grupo deberá presentar en no más de 15 minutos, el avance y cumplimiento de los hitos. El primer hito debe incluir la presentación del plan y la definición de las interfaces (el documento de Mensajes compartido).

La implementación y defensa de este trabajo será evaluada, y es condición necesaria su aprobación para aprobar la cursada. La implementación puede ser grupal, pero la defensa será oral e individual. Consistirá en responder preguntas conceptuales y técnicas relacionadas con el desarrollo del mismo.

Fecha de entrega del sistema: 14/12 (antes de la clase de práctica)
Fecha de defensa: 14/12