PCS: Diseño Chat Algoritmo Berkeley (ZeroMQ)

Programación en sistemas Cloud

Alumno

José Javier Gutiérrez Gil

jogugi@posgrado.upv.es| jogugil@gmail.com

9 dicembre 2024

Profeso/i

Bataller Mascarell, Jordi (batallerdsic.upv.es)

Índice

CAPÍTULO 1

Algoritmo de Berkeley

El algoritmo de Berkeley es un método de sincronización de relojes en sistemas distribuidos. Se basa en la existencia de un líder (coordinador) que ajusta los relojes de los nodos en el sistema para mantener la coherencia temporal.

1.1 Objetivos

Refinar el diseño del "Algoritmo de Berkeley" que aparece en `"Apuntes_NotacionDiseño.pdf"` basándolo en sockets de ZeroMQ. A continuación, se explica paso a paso el algoritmo:

- **Inicio del proceso de sincronización:** El líder toma la iniciativa para sincronizar los relojes en el sistema y envía una petición a todos los nodos seguidores solicitando sus marcas temporales actuales.
- **Seguidores responden al líder:** Cada seguidor responde al líder con su marca temporal actual y, opcionalmente, con el tiempo que tardó en procesar la solicitud (latencia de ida y vuelta).
- **Líder calcula los ajustes:** Una vez que el líder recibe todas las marcas temporales, calcula el tiempo promedio ajustado del sistema (considerando posibles retrasos) y determina la diferencia de tiempo entre el promedio y el reloj de cada nodo.
- **Líder envía ajustes:** El líder comunica a cada nodo un ajuste temporal que especifica cuánto adelantar o retrasar su reloj.
- **Seguidores aplican ajustes:** Cada seguidor ajusta su reloj local según el valor recibido. En algunos casos, los ajustes se aplican de forma gradual para evitar cambios bruscos.
- **Finalización:** Los relojes de los nodos quedan sincronizados dentro de los límites de error permitidos por la red.

El algorimo de berkeley representado en IOAutomata se muestra en al figura 1.1. Adaptaremos eta representación usando zeroMQ para las comunicaciones.

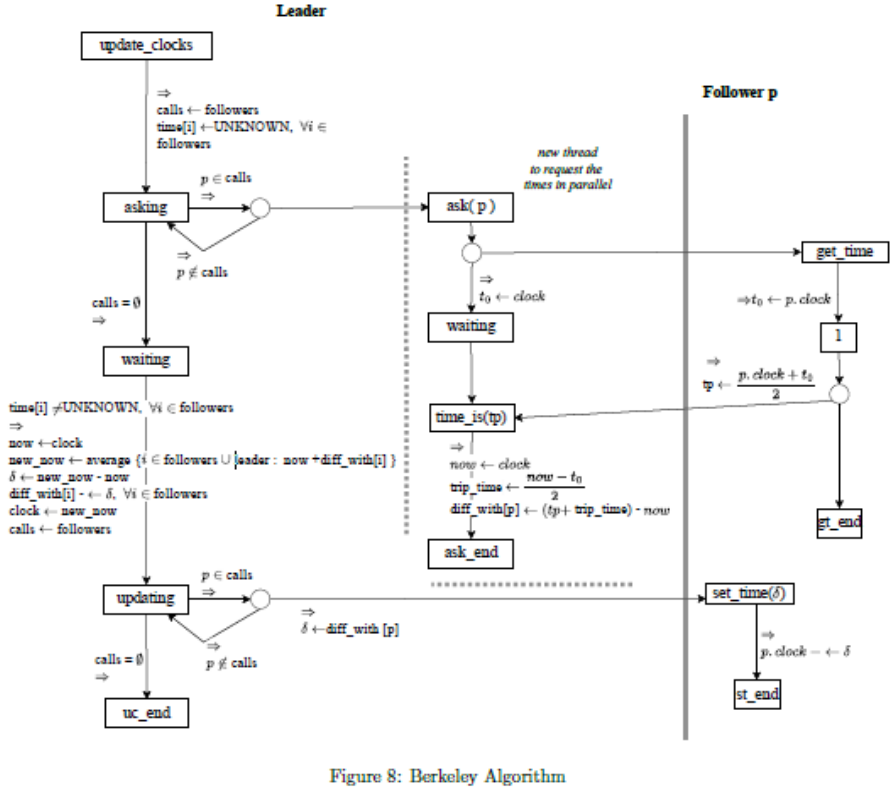

Figure 8: Berkeley Algorithm

Figura 1.1. Algoritmo Berkeley JAutomata (ref. Jordi Ballester «notacion.pdf»)

1.1.1 Cambios en el diagrama para representar ZeroMQ

Se proponen los siguientes cambios en el diagrama del algoritmo Berkeley para integrar sockets ZeroMQ:

- **Líder (Request/Reply):**

  El lider mantiene una nueva variable $req[i]$ que permite verificar que un nodo seguidor ha respondido correctamente a la solicitud de tiempo del líder. Dicha variable se inicializa a false para cada follower ($req[i] = false$ y $time[i] = UNKNOWN$)

  En el flujo para cada hilo donde el lider obtiene el tiempo de cada nodo p se realia lso sigueitnes cambios:

  - Cambiar el nodo *asking* para reflejar el envío de un mensaje ZeroMQ con el patrón REQ (Request) para pedir los tiempos de los seguidores. Notar que cada nodo p tiene un REP para esperar las peticiones del lider. (El líder abrirá un socket REQ para cada nodo y enviará un mensaje de solicitud de tiempo.)
  - Eliminar el nodo *waiting* ya que en el anterior ya espera la respuesta REP (Reply) del nodo p. Además, si se llega el tiempo t_p del nodo p actualizamos el valor de $req[p] = True$ para controlar que followers devuelven correctamente su tiempo.

  Una vez el lider obtiene todos los tiempos de cada follower, modificamos las sigueintes operaciones;

- Calculamos el tiempo promedio pero sólo con el número de followers que han contestado correctamente, es decir, aquellos que tienen true en la variable req. Por tanto, modificamos el cálculo:

$$\texttt{new\_now} \leftarrow \text{average}\,(i \in \text{followers} \wedge \text{req}[i] = \text{true} \cup \text{leader} : \text{now} + \text{diff\_with}[i])$$

- Modificamos la variable `diff_with` solo para los nodos que han contestado con su tiempo:

$$\texttt{diff\_with}[i] \leftarrow \delta, \forall i \in \text{followers}, \text{req}[i] = \text{true}$$

- Creamos el conjunto `call` sólo con estos nodos que tienen la variable `req` a true:

$$\texttt{call} \leftarrow \{i \in \text{followers} : \text{req}[i] = \text{true}\}$$

  De esta forma, el líder sólo enviará el delta para que modifique su tiempo local solo a los nodos que han contestado.

- **Seguidores (Request/Reply):**
  - Cambiar el nodo *get_time)* para representar la recepción de una solicitud REQ del líder.
  - Añadir un nodo para que cada seguidor envíe una respuesta REP con el tiempo calculado o el ajuste solicitado.
  - Modificar el nodo set_time añadiendo un nodo en cada seguidor que indique que está esperando la solicitud de ajuste y que recibirá el ajuste del líder mediante una respuesta REP.

Con esta corrección, el cálculo de `new_now` ahora solo toma en cuenta los seguidores que han respondido correctamente (aquellos donde `req[i] = true`) y el líder, asegurando que solo se promedien los tiempos válidos. Esto es crucial para sincronizar los relojes de manera correcta en un sistema distribuido. Esta nueva variable ayuda a filtrar los nodos que han respondido correctamente. Si el líder envía una solicitud de tiempo (`REQ`) y no recibe respuesta (por algún fallo en la red o en el nodo), entonces `req[i]` se mantiene como `false`, y ese nodo no se incluye en el cálculo del promedio de tiempo (`new_now`). Mejora la robustez del algoritmo y facilita la comprensión y el manejo de errores. Asegura que solo los nodos que han respondido exitosamente sean tomados en cuenta para el cálculo del nuevo tiempo ajustado. Esto ayuda a evitar errores de sincronización si algún nodo no puede ser alcanzado por el líder.

El esquema del algoritmo, adaptado a ZeroMQ utilizando REQ/REP y con la utilización de la variable $req[i]$ se refleja en la imagen 1.2.

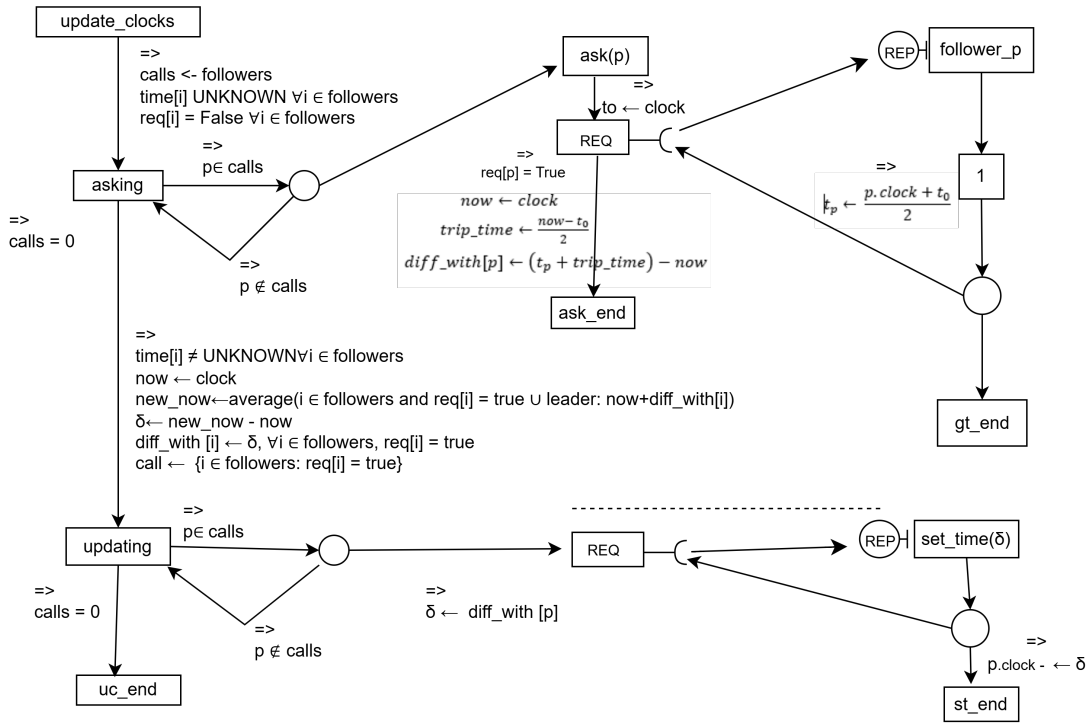

Figura 1.2. Algoritmo Berkeley con Sockets REQ/REP ZeroMQ