Algoritmos de sincronización para el control de acceso a secciones críticas en sistemas distribuidos

Bibliografía:

Frédéric Magoulés, “Fundamentals of Grid Computing. Theory, Algorithms and Technologies”, cap. 2, pp. 29-66 Edit. CRC Press, 2010.

Objetivos

- Describir técnicas de sincronización para resolver el problema de la sección crítica en entornos distribuidos dispersos (clústeres).
- Nos interesa:
  - Cómo operan.
  - Sus ventajas e inconvenientes.

Contenidos

1. Introducción
2. Algoritmos basados en testigo.
    1. Algoritmo de Martin
    2. Algoritmo de Naimi-Thréhel
    3. Algoritmo de Suzuki-Kasami
3. Algoritmos para grandes configuraciones
    1. Algoritmos basados en prioridad
    2. Estrategias combinadas
4. Rendimiento

1. Introducción

- Objetivo: garantizar el acceso correcto a una sección crítica (SC) en entornos distribuidos en los que los nodos están interconectados por red.
- Los protocolos se basan en mensajes de red que intercambian los nodos que quieren acceder a la SC.
- Por tanto, sus rendimientos dependen de:
  - Tráfico de mensajes que generan.
  - El tiempo de latencia de los mensajes.

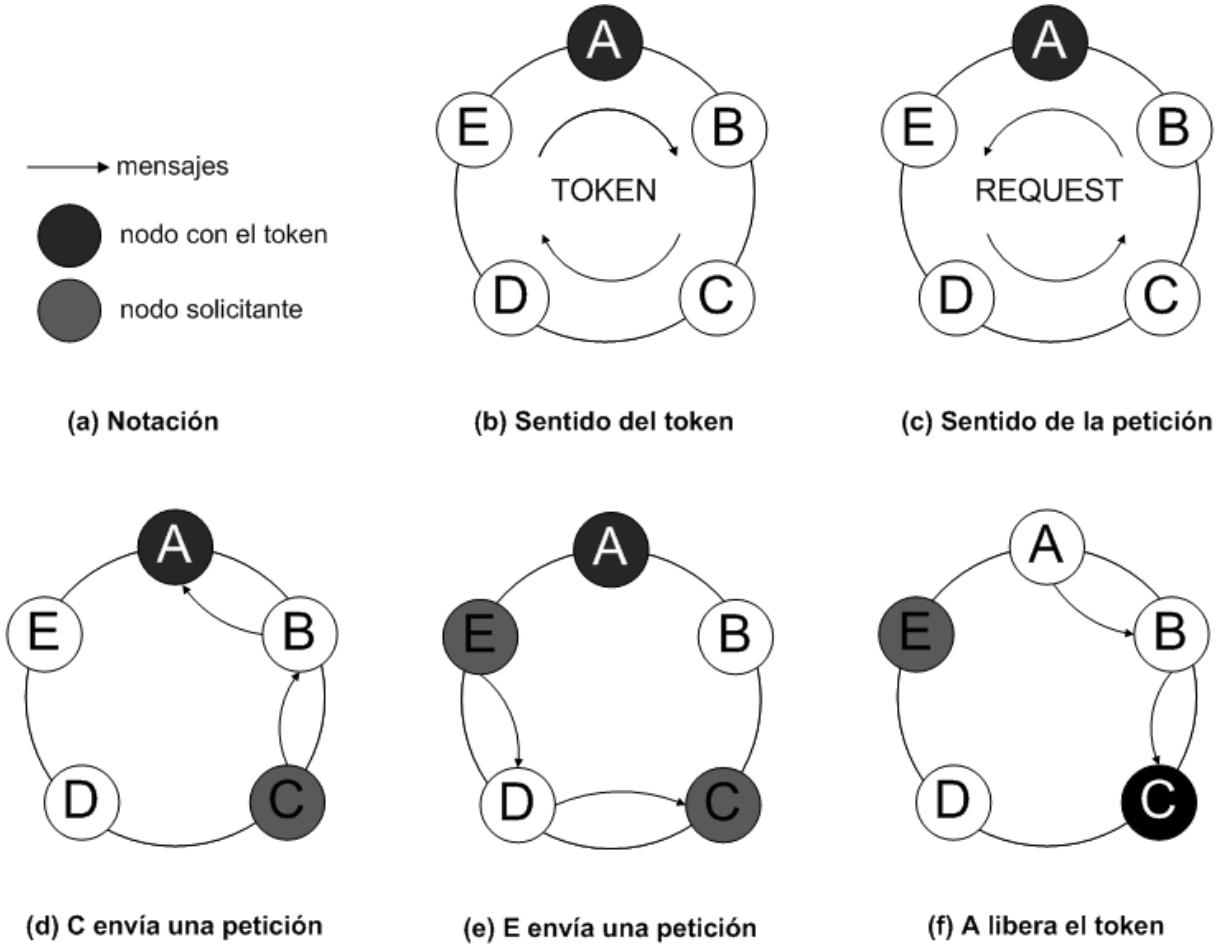
1. Introducción

- Dos tipos de algoritmos:
  - **Basados en permisos.** Un nodo accede a la SC si tiene permiso el permiso de los demás nodos o de la mayoría de ellos.
  - **Basados en testigos.** Un testigo circula entre los nodos. Sólo se puede acceder a la SC si se posee el testigo.
- En general, los protocolos basados en testigos producen un menor tráfico de mensajes que los basados en permisos.

2. Algoritmos basados en testigo: algoritmo de Martin

- Desarrollados en 1985 , se basa en dos mensajes:
  - Mensaje de petición de testigo.
  - Mensaje de testigo.
- Principios de funcionamiento:
  - Los nodos se organizan en un anillo.
  - El testigo recorre el anillo en un sentido.
  - Las peticiones de testigo recorren el anillo en el sentido opuesto.
  - Un nodo sólo puede entrar en la sección crítica (SC) si está en posesión del testigo.
  - Cuando un nodo i, que no posee el testigo, desea entrar en la SC, entonces envía una solicitud de testigo a su nodo sucesor, j, en el anillo.
    - Si éste no posee el testigo y no lo ha solicitado previamente, entonces reenvía la solicitud (en el sentido de la circulación de la petición del testigo) a su nodo sucesor k en el anillo.
    - Si no posee el testigo, pero lo ha solicitado previamente, entonces no la reenvía. La retiene hasta que satisface su petición.
  - Cuando la solicitud llega al nodo que posee el testigo, nodo k, entonces:
    - Si k está en la SC, entonces retiene la petición de testigo.
    - Si K no está en la SC , entonces envía el testigo a su sucesor (en el sentido de la circulación del testigo). Cada nodo entre k e i procederá de la misma manera.
    - Si durante el camino del testigo entre los nodos k e i, éste pasa por un nodo que ha solicitado el testigo para entrar en la SC, entonces se satisface su petición.

2. Algoritmos basados en testigo: algoritmo de Martin



(a) Notación

(b) Sentido del token

(c) Sentido de la petición

(d) C envía una petición

(e) E envía una petición

(f) A libera el token

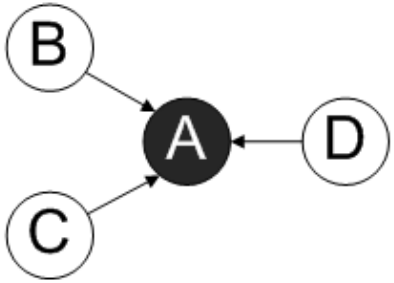
2. Algoritmos basados en testigo: algoritmo de Naimi-Tréhel

- Desarrollados en 1996 , se basa en dos estructuras de control que poseen cada nodo:
  - Puntero ÚLTIMO: da lugar a un árbol dinámico cuyo nodo raíz es el nodo del sistema que recibirá el testigo en último lugar entre los demandantes.
  - Cola PROXIMO: una cola distribuida que representa los nodos que están esperando por el testigo
  - Cada nodo posee dos punteros que apuntan a sus nodos ÚLTIMO y PRÓXIMO.

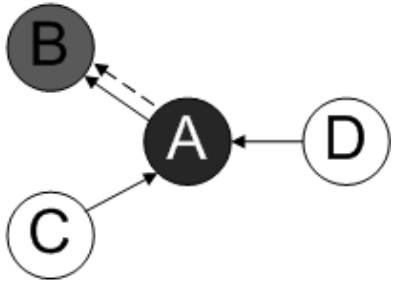
- Principios de funcionamiento:
  - Inicialmente el nodo raíz del árbol ÚLTIMO es el nodo que posee el testigo y el puntero ÜLTIMO de los demás apuntan al nodo *raíz del árbol.*
  - En todo momento el nodo raíz del árbol ÚLTIMO posee su puntero ÚLTIMO a nulo.
  - Una petición de testigo viaja a través de un camino indicado por los punteros ÜLTIMO hasta el nodo raíz.
  - Cuando un nodo recibe una petición , cada nodo del camino actualiza su puntero ÚLTIMO apuntando al nodo peticionario.
  - Cuando la petición de testigo llega al nodo raíz del árbol ÚLTIMO, éste actualiza su puntero PRÓXIMO para que apunte al nodo peticionario.
  - Cuando un nodo libera la sección crítica, entonces envía el testigo al nodo apuntado por su puntero PRÓXIMO.

2. Algoritmos basados en testigo: algoritmo de Naimi-Tréhel

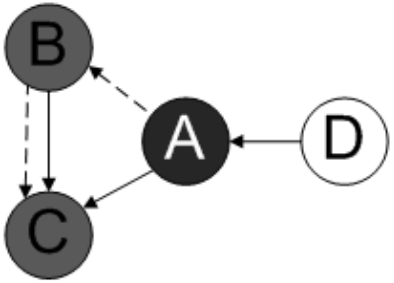
(a) Notación



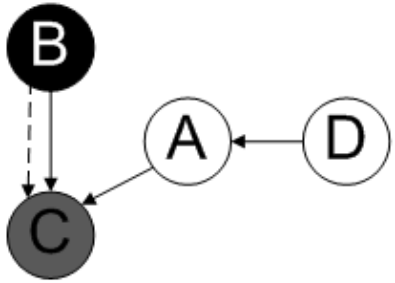
(b) Estado inicial



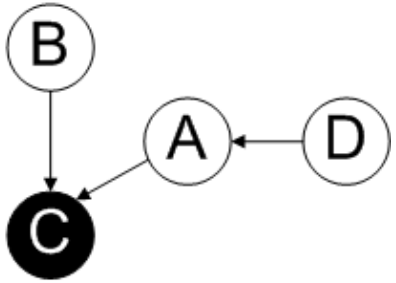
(c) B envía unapetición



(d) C envía una petición



(e) A libera el token

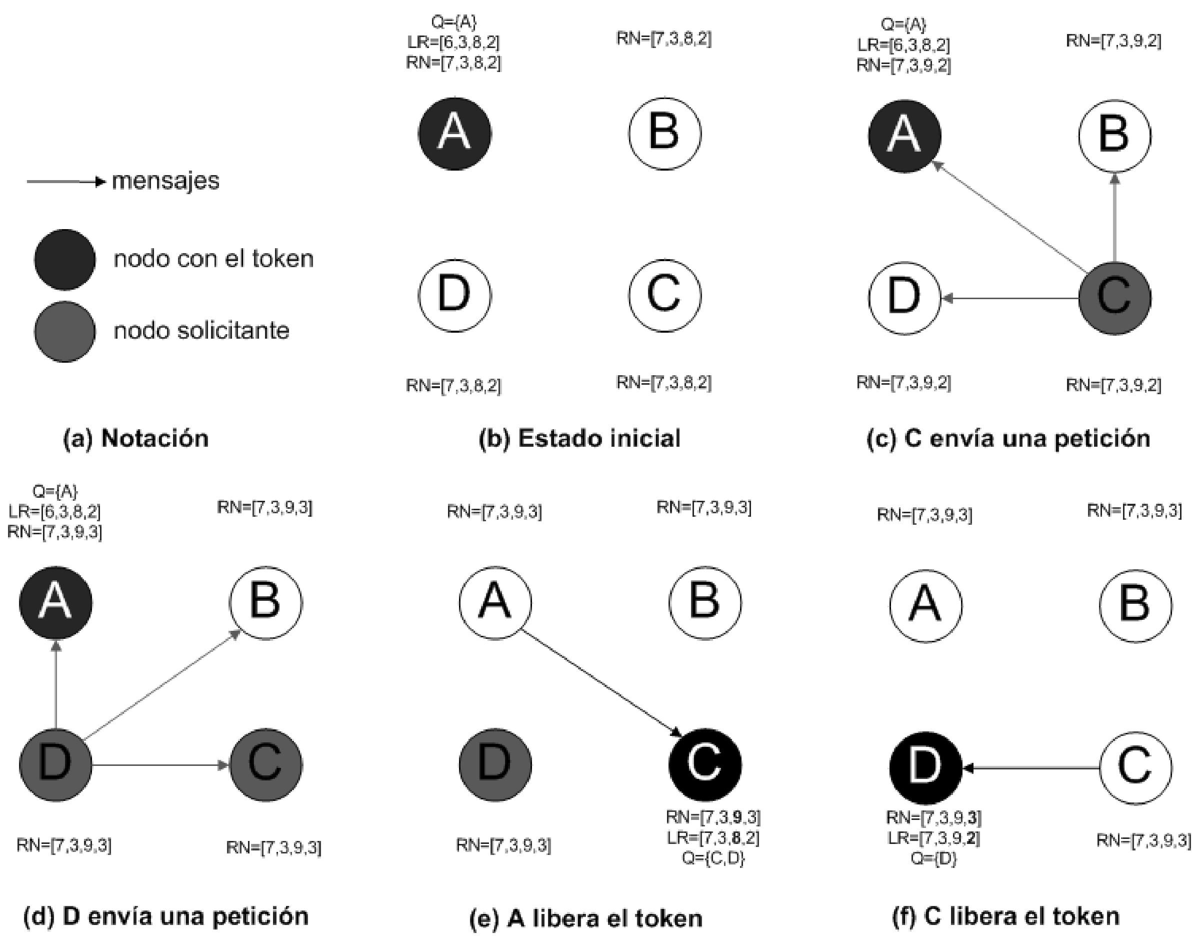


(f) B libera el token

2. Protocolos basados en testigo: algoritmo de Suzuki-Kasami

- Desarrollados en 1985, se basa en tres estructuras de control:
  - Cada nodo posee un array RN_i de dimensión N (número de nodos). Cada elemento j de $Rn_{i,.}$ o sea $Rn_i[j]$, almacena el número de veces que el nodo j ha solicitado entrar en la sección crítica (número de secuencia), según el conocimiento del nodo i.
  - Cola Q de nodos que están esperando por el testigo. Esta cola Q viaja con el testigo.
  - Array LN de dimensión N, que indica el número de secuencia más reciente satisfecha de cada nodo. LN viaja con el testigo.
- Principios de funcionamiento:
  - Cuando un nodo i, que no tiene el testigo, intenta entrar en la sección crítica, entonces envía un mensaje de petición de testigo al resto de los nodos. Este mensaje de petición posee dos campos:
    - Nodo peticionario. Este campo indica el nodo i que solicita el testigo.
    - Número de secuencia. Este campo informa del número de veces que el nodo peticionario i ha solicitado entrar en la sección crítica.
  - Cuando un nodo i abandona la sección crítica, entonces:
    - Actualiza LR[i] al valor de $Rn_i[i]$.
    - Actualiza la cola Q añadiendo todos los nodos que aún no estaban incluidos en Q y que él sabe que sus peticiones de testigo no han sido satisfechas.
    - Una vez realizada la actualización anterior de Q, si ésta no está vacía, entonces se extrae el primer nodo j de la cola y se le envía el testigo.

2. Algoritmos basados en testigo: algoritmo de Suzuki-Kasami



3. Algoritmos para grandes configuraciones: introducción

- Se aplican en grandes configuraciones de clúster:
  - Un gran número de nodos.
  - Varios clústeres
  - Dispersos en amplias áreas geográficas.
- Objetivos:
  - Reducir el número de mensajes que intervienen en el algoritmo de arbitraje de entrada en la SC.
  - Disminuir los tiempos de espera antes de entrar en la SC.

3. Algoritmos para grandes configuraciones: Algoritmos basados en prioridades

- Algoritmo de *Bertier* (2004). Consiste en una adaptación del Algoritmo de *Naimi-Tréhel:*
  - Las peticiones de la cola NEXT se priorizan atendiendo primero las peticiones intra-clúster.
  - La inanición se evita estableciendo un umbral en cada clúster (U); mientras el número de peticiones sea menor que U, entonces éstas tienen prioridad.
  - Una petición de testigo viaja a través de un camino indicado por los punteros LASTs hasta el último nodo local de su clúster (*local-root*).
  - Si el puntero NEXT del *local-root* apunta a un nodo remoto y si U no se ha alcanzado, entonces el puntero NEXT del *local-root* se actualiza apuntando al nodo peticionario local.

3. Algoritmos para grandes configuraciones: Algoritmos basados en prioridades

- Algoritmo de *Muller* (1998). Consiste en una extensión del Algoritmo de *Naimi-Tréhel*:
  - A cada petición del testigo se le asocia una prioridad y se satisface aquella con mayor prioridad.
  - Se introduce una cola local en cada nodo. Todas las colas de los nodos forman una cola virtual y global:
    - Ordenada por la prioridad de las peticiones.
    - A igual prioridad se utiliza una política FIFO.
  - Cuando un nodo realiza una petición, ésta se propaga a través del árbol LAST hasta alcanzar un nodo que ha realizado una petición con igual o mayor prioridad, entonces el nodo almacena la petición en su cola.
  - Cuando un nodo cede el testigo, este añade al testigo su cola local.
  - El nodo que recibe el testigo añade a su cola la recibida con el testigo, reordenándola según las prioridades.
  - Aunque los autores no lo contemplaron, para configuraciones formadas por la interconexión de varios clústeres, entonces el algoritmo se extendería:
    - Distinguir prioridades intra-clúster y prioridades inter-clúster.
    - Primero se atienden a las peticiones intra-clúster (aquellas que provienen de nodos pertenecientes al nodo que posee el testigo) y después las peticiones de nodos de otros clústeres.

3. Algoritmos para grandes configuraciones: Estrategias combinadas

- Se basan en distinguir dos niveles:
  - Nivel intra-clúster: peticiones pertenecientes a nodos pertenecientes a un mismo clúster.
  - Nivel inter-clúster: peticiones pertenecientes a nodos pertenecientes a diferentes clústeres.
- Se establece una prioridad entre los dos niveles.
- En cada nivel se utiliza un algoritmo que en la mayoría de las propuestas son diferentes.

4. Rendimiento

- Los resultados de las pruebas de rendimiento están condicionados a:
  - la heterogeneidad de las configuraciones:
    - Nodos.
    - Clúster.
    - Topologías de las redes.
  - Naturaleza de las aplicaciones que intervienen en las pruebas de rendimiento:
    - A: tiempo consumido por un nodo para ejecutar una SC.
    - B: intervalo de tiempo que transcurre entre que sale de una SS y solicita entrar en la siguiente CS.
    - P: ratio B/A, que expresa la frecuencia con la que un nodo solicita entrar en SC.
  - Grado de paralelismo de las aplicaciones que intervienen en las pruebas de rendimiento. Si N es el número total de procesos que interviene en la prueba, entonces:
    - Aplicaciones con un grado bajo de paralelismo: P <= N. Aplicaciones en las que la mayor parte de su ejecución se realiza en SC.
    - Aplicaciones con un nivel intermedio de paralelismo: N < p <= 3N. Aplicaciones en las que en algunas secciones de su código se ejecutan en SC.
    - Aplicaciones con un nivel alto de paralelismo: 3N <= P. Aplicaciones en las que rara vez se entra en SC.

4. Rendimiento

- Métricas:
  - Tiempo de espera en la obtención de la autorización para ejecutar la SC (T_{pendCS}). Su valor es la suma de:
    - Tiempo de transmisión de la solicitud del testigo (T_{req}).
    - Tiempo de procesamiento del algoritmo(s) de sincronización en los nodos.
    - Tiempo de transmisión del testigo (T_{token}).
  - Varianza del tiempo de espera.
  - Número de mensajes intercambiados por la técnica de sincronización empleada.

4. Rendimiento

- Un caso de análisis de rendimiento: French Large-Scale Grid'5000:
  - 17 clúster ubicados en 9 ciudades diferentes de Francia.
  - Cada nodo:
    - Procesador: Bi-Opteron (AMD X86).
    - Memoria: 2 Gbytes.
    - Conexión inter-clúster: enlaces dedicados de 10Gb/s.

4. Rendimiento

- **Tiempo promedio de obtención del testigo**

Intra-clúster/inter-clúster	Aplicaciones con un grado bajo de paralelismo	Aplicaciones con un grado medio de paralelismo	Aplicaciones con un grado alto de paralelismo
Naimi/Suzuki	**T_{pendCS} + T**	**T_{pendCS} + T**	**T + T**
Naimi/Martin	**T_{pendCS} + T**	**T_{pendCS} + K*T**	**N*T**
Naimi/Naimi	**T_{pendCS} + T**	**T_{pendCS} + T**	**Log(N) * T + T**

  - **T_{pendCS}** : Tiempo promedio requerido para satisfacer todas las peticiones inter-clúster del testigo, antes de satisfacer la petición inster-clúster analizada.
  - **T:** Tiempo promedio de transmisión de un mensaje entre dos nodos coordinadores.
  - **N**: Número de procesos.
  - **K:** número variable que depende del número de nodos coordinadores que intervienen para satisfacer la petición inter-clúster analizada.

4. Rendimiento

- **Número promedio de mensajes inter-clúster**

Intra-clúster/inter-clúster	Aplicaciones con un grado bajo de paralelismo	Aplicaciones con un grado medio de paralelismo	Aplicaciones con un grado alto de paralelismo
Naimi/Suzuki	**N+1**	**N+1**	**T + T**
Naimi/Martin	**2**	**2 < *K* < *N***	**(N/2) + (N/2)**
Naimi/Naimi	**log(N) + 1**	**Log(N) + 1**	**log(N) + 1**

  - **N:** número de procesos
  - **K:** variable que depende del número de nodos coordinadores que intervienen