Universidad Nacional Autónoma de México

Facultad de Ciencias

Computación Distribuida

Tarea 11

Johann Ramón Gordillo Guzmán
418046090

Tarea presentada como parte del curso de **Computación Distribuida** impartido por la profesora **M.C Karla Rocío Vargas Godoy**.

26 de Enero del 2021

Link al código fuente: https://github.com/JohannGordillo/

Índice

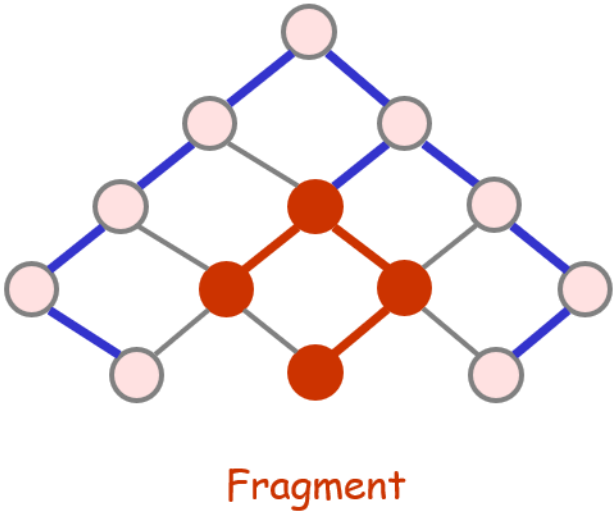
1 Árboles generadores de peso mínimo

1. ¿Qué es un fragmento?

   Respuesta.

   Un fragmento es un subárbol del árbol generador de peso mínimo (MST).

   Una propiedad interesante de los árboles generadores de peso mínimo es que, si nos tomamos un fragmento F del árbol generador de pesó mínimo T y la arista e de peso mínimo que sale del fragmento (conocida como *outgoing edge*), al unir mediante la arista e a su nodo adyacente con F se formará otro fragmento de T.



2. ¿Cuáles son los estados de un nodo?

   Respuesta.

   Un nodo puede tener los siguientes estados:

   - **Dormido.**

     Estado inicial (cuando inicia el algoritmo para el nodo).

   - **Encontrar.**

     Cuando el nodo participa en la búsqueda de la arista de peso mínimo que sale del fragmento al que pertenece.

   - **Encontrado.**

     En cualquier otro instante. Es decir, si no está en estado *dormido* ni en estado *encontrar*.

2 Leader Election Algorithm

3. ¿Por qué los algoritmos de elección del líder no están diseñados para redes anónimas?

Respuesta.

Es inmediato del siguiente resultado:

Teorema 1. *No hay algoritmos deterministas que resuelvan el Problema de Elección del Líder en un anillo uni/bi-direccional anónimo para más de un proceso.*

Demostración. Supogamos que existe un algoritmo distribuido A que resuelve el problema de elección del líder en una red anónima.
El algoritmo A está compuesto de n algoritmos locales deterministas $A_1, ..., A_n$, donde A_i es el algoritmo local ejecutado por el proceso p_i. Como la red es anónima, se tiene que:

$$A_1 = A_2 = \cdots = A_{n-1} = A_n$$

y también por el anonimato de la red se cumple que los estados iniciales σ_i^0 son iguales para todo i.

Luego, como todos los procesos ejecutan el mismo algoritmo local determinista A_i, existe una ejecución síncrona durante la cual todos los procesos ejecutan el mismo paso de su algoritmo local y cada proceso procede consecuentemente de σ_i^0 a σ_i^1.

En general, de un estado simétrico global (con todos los procesos en el mismo estado local) podemos pasar siempre a otro estado simétrico global, por lo que la ejecución síncrona nunca termina, ya que para terminar debe entrar en un estado asímetrico global.

Ya que el algoritmo A no cumple la propiedad de terminación para el problema de elección del líder, llegamos a una contradicción.

$\therefore$ No existe un algoritmo que resuelva el problema de elección del líder en una red anónima. ■

Así, suponer la existencia de un algoritmo distribuido que resuelva el problema de elección del líder en una red anónima nos lleva a una contradicción, pues no se cumple la propiedad de terminación para el problema de elección del líder.

4. ¿Cuáles son algunas de las ventajas de trabajar con un líder único?

Respuesta.

Trabajar con un líder único trae consigo muchas ventajas, algunas de ellas son:

- Es más fácil para los humanos pensar en los sistemas distribuidos y darnos cuenta en dónde están los registros para evitar errores humanos.

- Se puede trabajar de manera más eficiente, ya que al no haber ambigüedad y saber dónde están los datos, podemos acceder directamente al líder y obtener lo que queramos.

- Se puede mejorar el rendimiento y reducir los costos computacionales del algoritmo.

3 Consenso con fallas bizantinas para $f < n/3$

5. ¿Cuál es el problema con la propiedad de acuerdo en sistemas con fallos que tenemos cuando estamos en un sistema con fallas bizantinas?

   Respuesta.

   Dada la naturaleza de los procesos bizantinos, si un proceso decide un valor, es imposible hacerlo decidir el mismo valor que los procesos correctos.

   De lo anterior se sigue que la propiedad de acuerdo uniforme, que nos dice que ningún par de procesos decide valores distintos, no tiene sentido cuando estamos en un sistema con fallas bizantinas.

   En su lugar, la propiedad de acuerdo que pediremos es que ningún par de procesos correctos decidan valores distintos.

6. ¿A qué se debe el nombre del algoritmo Exponential Information Gathering?

   Respuesta.

   El Exponential Information Gathering (EIG) es un algoritmo para resolver el consenso con fallas bizantinas con $f < n/3$ fallas. Tiene su origen en el algoritmo originalmente propuesto por *L.Lamport* y *R.Shostack* y debe su nombre a que la complejidad del algoritmo en cuanto al tamaño de los mensajes es proporcional a:

   $$(n)(n-1)\cdots(n-(f+1))$$

   Es decir, su complejidad es $O(n^f)$ (exponencial).

   Es óptimo desde el punto de vista del número de fallas, pues nos permite resolver el consenso con $f < n/3$ fallas, y también es óptimo desde el punto de vista de la complejidad en tiempo, ya que requiere $f + 1$ rondas. El único problema que presenta el algoritmo es que el tamaño de los mensajes incrementa de forma exponencial con el número de ronda.

   La ejecución consiste en dos partes. En la primera parte cada proceso va a generar un árbol local donde se va a almacenar toda la información recibida (a esta parte se le conoce como Information Gathering o Recopilación de Información) y en la segunda parte se va a procesar dicha información.

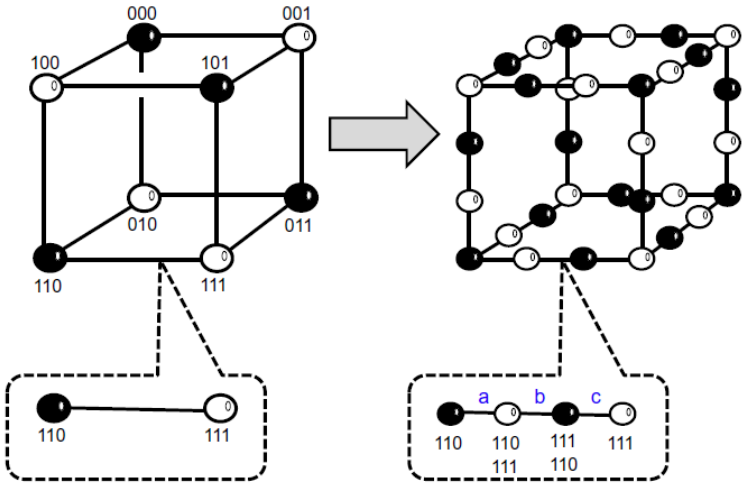
4 Topología en sistemas distribuidos

7. Explica con tus propias palabras qué es un complejo simplicial.

Respuesta.

Los complejos simpliciales son objetos geométricos combinatorios que representan todas las posibles vistas locales de los procesos del sistema en un momento dado.

Son un tipo particular de espacio topológico cuya estructura puede variar dependiendo del tipo de comunicación (por ejemplo, comunicación confiable o no confiable) que se esté utilizando.



Se obtienen congelando todas las posibles combinaciones diferentes de operaciones y escenarios de falla en algún punto del tiempo.

8. ¿Por qué en el problema de los niños sucios en su representación combinatoria, cada niño pertenece a solamente dos triángulos?

Respuesta.

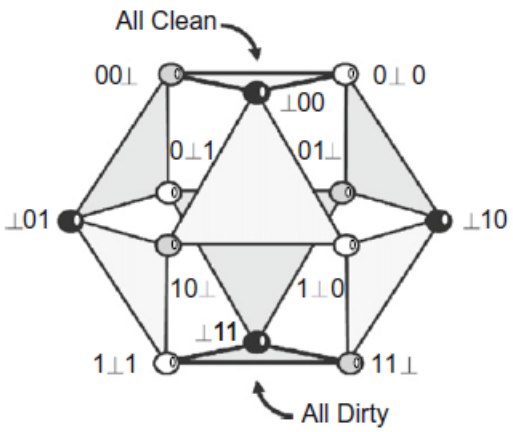
El enfoque combinatorio provee una representación geométrica de los valores de entrada del problema, así como de su evolución.

Suponemos que tenemos tres niños representados por los colores blanco, negro y gris.

En este enfoque cada vértice representa la entrada de un niño con un vector de las siguientes características: Las entradas en las que el vector tenga un 0 significan que los otros niños estan limpios y las entradas con 1 significa que estan sucios, además de que su vector de entrada tendrá un $\perp$, pues un niño no conoce su estado.

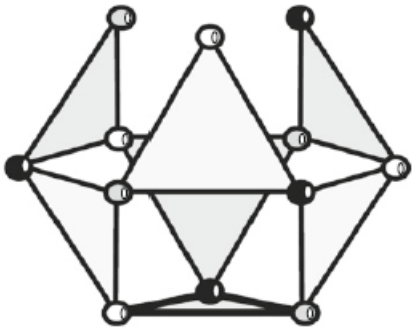
Usamos triángulos para representar cada posible configuración uniendo estados compatibles.

En el problema de los niños sucios en su representación combinatoria cada niño (cada vértice) pertenece exactamente a dos triángulos.



Esto se debe a la incertidumbre de cada niño sobre su estado. Su conocimiento es compatible con dos posibles situaciones: una en la que está sucio y otra en la que esta limpio. En un triángulo al que pertenece su estado es en el que está limpio y en el otro triángulo el niño está sucio.

Sin embargo, cuando el profesor anuncia que hay alguien sucio, tres nodos van a pertenecer a un solo triángulo, como se ve en la siguiente imagen:



Por lo tanto, en todo momento cada niño pertenece a lo más a dos triángulos.

5 Sincronizadores

9. ¿Cuáles tipos de sincronizadores existen y en qué difieren?

   Respuesta.

   Los sincronizadores son algoritmos distribuidos asíncronos que simulan un sistema síncrono sobre un sistema asíncrono. El rol de un sincronizador es generar pulsos para cada proceso de un sistema distribuido asíncrono y asegurar que un mensaje enviado al principio de un pulso r sea recibido por su proceso destinatario antes de que el pulso $r + 1$ comience.

   Se sigue que una secuencia de pulsos debe satisfacer la siguiente propiedad, denotada como ϕ: Un nuevo pulso $r + 1$ puede ser generado en un proceso solamente después de que este proceso ha recibido todos los mensajes enviados por sus vecinos en el pulso r.

   Decimos que un proceso p es seguro con respecto al pulso r si todos los mensajes que mandó a sus vecinos al comienzo del pulso ya han sido recibidos por su proceso destinatario.

   Hay cuatro tipos de sincronizadores:

   - Sincronizador α.

     El principio básico de este sincronizador es que cuando un proceso p se da cuenta de que es seguro con respecto a su pulso actual, se lo indica a sus vecinos por medio de un mensaje SAFE(). De esta manera, cuando un proceso a terminado sus acciones con respecto a un pulso r y conoce que sus vecinos son seguros respecto al pulso r, el sincronizador local puede generar el pulso $r+1$ en p.

   - Sincronizador β.

     El sincronizador β se basa en un árbol generador que tiene como raíz a un proceso p_a. Se transmiten mensajes SAFE() desde las hojas hacia la raíz y mensajes PULSE() desde la raíz hacia las hojas.

     Cuando un proceso hoja es seguro, este se lo indicará a su padre por medio de un mensaje SAFE(). Cuando un proceso no es hoja, debe esperar que tanto él como sus hijos sean seguros para poder indicarselo a su padre.

   - Sincronizador γ.

     Este sincronizador combina los sincronizadores α y β para obtener una mejor complejidad en tiempo que β y una mejor complejidad en mensajes que α. Esta combinación se basa en una partición del sistema distribuido.

   - Sincronizador δ.

     El sincronizador δ asume que un subárbol t-generador ha sido construido sobre la gráfica de comunicación definida por el algoritmo síncrono. Cuando un proceso se vuelve seguro, éste ejecuta t fases de comunicación con sus vecinos en el subárbol t-generador, para al final saber que todos los procesos en la gráfica son seguros.

Como vimos, los sincronizadores difieren en la forma en que entregan a cada proceso la información.

Se dividen en dos: básicos y avanzados. Los sincronizadores básicos son α y β, mientras que los avanzados son γ y δ.

10. ¿Qué partes del sincronizador α y β son tomadas para hacer el sincronizador γ?

Respuesta.

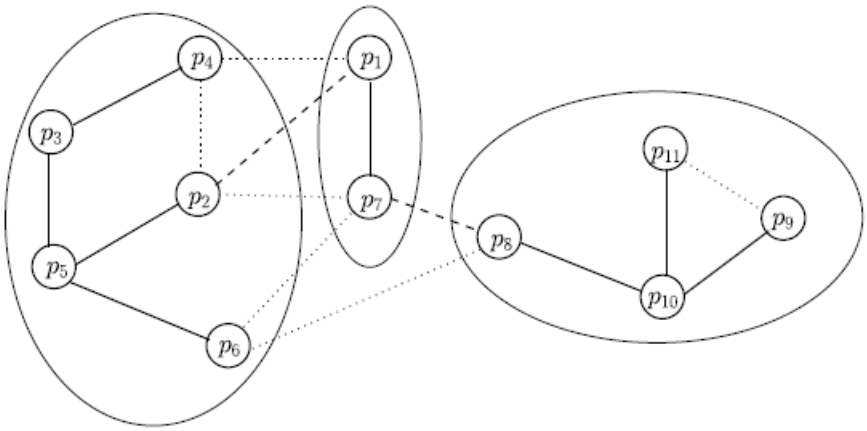
En el sincronizador γ dividimos la gráfica generada por el sistema en árboles generadores interconectados. Los árboles están conectados por medio de canales de comunicación. A los árboles se les llama *grupos*.

En cada grupo el árbol es usado (al igual que en el sincronizador β) para permitir a los procesos del grupo aprender que el grupo es seguro, es decir, que todos los procesos del grupo son seguros.

Posteriormente, cada grupo se comporta como si fuera un único proceso y el sincronizador α es usado para permitir a cada grupo saber si sus grupos vecinos son seguros. Cuando esto ocurre, un nuevo pulso puede ser generado dentro del grupo correspondiente.

Además, el sincronizador γ reutiliza algunos tipos de mensajes de los sincronizadores α y β, como SAFE() y ACK(). Los nuevos tipos de mensajes en este sincronizador son ALL_GROUP_SAFE() y GROUP_SAFE().

Notemos que este sincronizador combina los sincronizadores α y β para obtener una mejor complejidad en tiempo que β y una mejor complejidad en mensajes que α basándose en una partición del sistema distribuido.



6 Referencias

[1] Gallager, R., Humblet, P. & Spira, P. (1983). *A Distributed Algorithm for Minimum-Weight Spanning Trees*. ACM TOPLAS, Vol. 5, No. 1, Páginas 66-77.

[2] Raynal, M. (2013) *Distributed Algorithms for Message-Passing Systems*. Springer.

[3] Raynal, M. (2018) *Fault-Tolerant Message-Passing Distributed Systems: An Algorithmic Approach*. Springer.

[4] Herlihy, M., Kozlov, D. & Rajsbaum, S. (2014) *Distributed Computing Through Combinatorial Topology*. Elsevier.