Algunos problemas algorítmicos y algunos algoritmos distribuidos

Pablo Galdámez Saiz
Universidad Politécnica de Valencia
DSIC

Algunos problemas algorítmicos y algunos algoritmos distribuidos

1. Estado global
2. Elección de líder
3. Exclusión mutua
4. Difusiones con orden total
5. Difusiones con orden causal
6. Compromiso distribuido

1.- Estado global

1. El problema
2. Algoritmo de Chandy-Lamport

1.- El problema

- Dado un algoritmo distribuido, calcular su estado global.
- Estado global: estado local de cada proceso + estado de cada canal.
- Estado de cada proceso: sólo las variables que nos interesen: variables del algoritmo que estudiamos.
- Estado de cada canal: mensajes enviados y todavía no entregados.
- Usos del estado global:
  - Detección de terminación.
  - Depuración distribuida.

2.- Algoritmo de Chandy-Lamport

Conceptos previos

- Una **instantánea distribuida** refleja el estado que **pudo haberse alcanzado** en el sistema.
- Las instantáneas reflejan sólo estados consistentes.
- Si el estado de P, dice que P ha recibido un mensaje de Q, en la instantánea, Q envió antes un mensaje a P.
- La noción de estado global se refleja por el concepto de **corte.**

2.- Algoritmo de Chandy-Lamport

- Cualquier proceso inicia el algoritmo.
- Sea P el proceso iniciador.
- P guarda su estado local, luego envía un mensaje de **marca** a todos los demás procesos.
- Cuando un proceso recibe el mensaje de marca:
  - Si no ha guardado su estado local: guarda su estado local y envía **marca** a todos los demás procesos. A partir de este momento almacenará lo que llegue por los canales entrantes.
  - Si ha guardado su estado local, guarda los mensajes recibidos por el canal (desde que el proceso guardó su estado local, hasta que recibió marca por este canal).
- Un proceso termina, cuando ha recibido marca por todos los canales.
- Cuando termina, envía su estado y el de sus canales al iniciador.

Algunos problemas algorítmicos y algunos algoritmos distribuidos

1. Estado global
2. Elección de líder
3. Exclusión mutua
4. Difusiones con orden total
5. Difusiones con orden causal
6. Compromiso distribuido

1- El problema

- Dado un conjunto de nodos, elegir a un líder.
- Suponemos que hay un conjunto de nodos preconfigurado: desde el 1 al N. De entre esos nodos, algunos están funcionado y otros no, pero no sabemos cuáles.
- Los nodos que no funcionan, no funcionarán durante todo el algoritmo. Los que funcionan, funcionarán durante todo el algoritmo.
- Si todos los nodos (procesos) son idénticos, **no hay forma determinista de elegir a uno de ellos,** por ello asumiremos que tienen identificadores únicos.

2.- Algoritmos de elección

1. Algoritmo Bully
2. Algoritmo para anillos

2.1.- Algoritmo Bully

1. Cuando un proceso quiere comenzar una elección (p.ej, porque el líder actual no contesta), envía ELECCION a todos los nodos con número mayor al suyo.
2. Si no responde nadie, él es el líder.
3. Si alguien responde, el nodo no hace nada más
4. Cuando un proceso recibe ELECCION, envía OK a quien se lo envió (para avisarle que participa y le ganará)
5. Será líder quien no obtenga respuesta de los nodos más altos, habiendo recibido ELECCION.

2.2.- Algoritmo para anillos

1. Suponiendo que tenemos a los nodos ordenados en una anillo lógico. (p.ej, en orden creciente de identificadores)
2. Cuando un nodo quiere elegir nuevo líder, envía ELECTION por el anillo (al siguiente del anillo que esté vivo: que responda con ACK).
3. Cada nodo que ve el mensaje incluye su propio ID en el mensaje.
4. Cuando el mensaje llega de vuelta al iniciador, contiene los IDs de todos los nodos participantes. En este momento, envía un mensaje LIDER por el anillo, avisando del nuevo líder (el nodo mayor).
5. Nótese que puede haber más de un mensaje simultáneo: pero los dos intentos de elección darán el mismo resultado.

Algunos problemas algorítmicos y algunos algoritmos distribuidos

1. Estado global
2. Elección de líder
3. Exclusión mutua
4. Difusiones con orden total
5. Difusiones con orden causal
6. Compromiso distribuido

1.- El problema

- Varios nodos pueden tratar de entrar a su sección crítica en cualquier momento
- En cada instante como máximo un nodo puede estar dentro de la sección crítica.
- Todo nodo que intente entrar, deberá lograrlo antes o después: para ello hemos de asumir que todo nodo que pide entrar en su sección crítica y lo consigue, antes o después pedirá salir de ella.

3.- Algoritmos de Exclusión mutua

1. Un algoritmo centralizado
2. Un algoritmo distribuido
3. Un algoritmo para anillos

3.1.- Un algoritmo centralizado

1. Se elige un nodo como coordinador (líder).
2. Cuando un nodo quiere entrar en la sección crítica, envía un mensaje al líder, pidiendo permiso.
3. Si ningún otro proceso está en la sección crítica, el líder responde CONCECER, respondiendo DENEGAR en caso contrario.
4. Cuando un proceso sale de su sección crítica, avisa al líder. Si el líder sabe de otro proceso que intentó entrar después, le envía CONCEDER.

3.2.- Un algoritmo distribuido

- Suponemos que todos los eventos están ordenados (p.ej, usando relojes de Lamport junto al número de nodo).

1. Cuando un proceso quiere entrar en la sección crítica, envía un mensaje TRY a todos.
2. Cuando un proceso recibe un mensaje TRY:
    1. Si no está intentando entrar en su sección crítica, envía OK.
    2. Si está en su sección crítica, no contesta y encola en mensaje.
    3. Si no está en su sección crítica, pero quiere entrar, compara el número de evento del mensaje entrante con el que él mismo envió al resto. Vence el número más bajo:
        1. Si el mensaje entrante es más bajo, responde OK.
        2. Si el más alto, no responde y encola el mensaje
3. Un proceso entra en la sección crítica, cuando recibe OK de todos.
4. Cuando sale, envía OK a todos los mensajes que retuvo en su cola.

3.3.- Un algoritmo para anillos

1. Un token circula por un anillo lógico.
2. Un proceso que quiere entrar en su sección crítica, espera a tener el token.
3. Sólo puede entrar en la sección crítica un proceso: el que tiene el token.
4. Si un nodo cae, hay que reconstruir el token.

Algunos problemas algorítmicos y algunos algoritmos distribuidos

1. Estado global
2. Elección de líder
3. Exclusión mutua
4. Difusiones con orden total
5. Difusiones con orden causal
6. Compromiso distribuido

4.1.- Difusiones con orden total

1. El problema:
   - Un nodo cualquiera envía un mensaje.
   - Todos los nodos entregan el mensaje.
   - Si dos nodos envían cada uno un mensaje, ambos mensajes deben ser entregados por todos los nodos en el mismo orden.
2. Solución sencilla: nodo especial secuenciador que impone el orden.

Algunos problemas algorítmicos y algunos algoritmos distribuidos

1. Estado global
2. Elección de líder
3. Exclusión mutua
4. Difusiones con orden total
5. Difusiones con orden causal
6. Compromiso distribuido

4.2.- Difusiones con orden causal

1. El problema:
   - Un nodo cualquiera envía un mensaje.
   - Todos los nodos entregan el mensaje.
   - Si dos nodos A y B envían respectivamente cada uno un mensaje a y b, y los mensajes tienen relación causal a—b, entonces todos los nodos deben entregar primero el mensaje a y luego el mensaje b.
2. Solución sencilla: utilizar relojes vectoriales para los eventos y cada nodo que deba entregar un mensaje, espera a saber que no existan eventos que precedan a tal evento de entrega.

Algunos problemas algorítmicos y algunos algoritmos distribuidos

1. Estado global
2. Algoritmos de elección
3. Exclusión mutua
4. Difusiones con orden total
5. Difusiones con orden causal
6. Compromiso distribuido

1.- Compromiso distribuido: el problema

1. Confirmar una transacción distribuida donde varios nodos han ejecutado acciones relacionadas con la transacción.
2. Podemos suponer que todos los nodos han ejecutado ya sus acciones asociadas a la transacción y saben si han de aceptarla o no.
3. Una vez todos los nodos han terminado, se debe confirmar la transacción en todos los nodos, si y sólo si todos los nodos están de acuerdo.

2.- Algoritmo de compromiso distribuido en 2 fases (2PC)

- Un nodo es el coordinador, los demás son participantes.
- El coordinador comienza tratando de confirmar la transacción: enviará un mensaje “commit” a todos los participantes. Para lograr “commit”, todos los participantes deben responder con “OK”.
- Los participantes responden “OK” o “NOK”, en función de su decisión local. (para simplificar, podemos suponer que responden aleatoriamente).
- Si algún nodo responde “NOK”, el coordinador debe cancelar la transacción con “rollback” y todos los participantes deben recibir tal mensaje.
- Si todos los nodos responden con “OK”, el coordinador debe confirmar con “confirm”. Todos los nodos deben recibir la confirmación.
- Una vez todos los nodos hayan recibido “confirm” o “rollback”, contestarán al coordinador con “FIN”. El nodo coordinador debe finalizar, indicando: transacción completada (commited/rolledback)

Algunos problemas algorítmicos y algunos algoritmos distribuidos

1. Estado global
2. Algoritmos de elección
3. Exclusión mutua
4. Difusiones con orden total
5. Difusiones con orden causal
6. Compromiso distribuido

Algoritmos de recolección de residuos

1.- Contar las referencias: Mantener un contador en el nodo servidor (nodo que creó el objeto). El contador puede tener un valor igual o superior a las referencias que están activas en cada momento--- Mensajes INC, DEC y REGISTER.

2.- Mantener en el servidor (el nodo que crea el objeto) una lista de nodos que referencian al objeto. Los nodos que tengan referencias, las deben “renovar”.

Recolección de residuos

- Un nodo crea un objeto y obtiene una referencia al objeto.
- Un nodo que tenga una referencia, la puede enviar a otro nodo.
- Un nodo que tenga una referencia, la puede descartar.
- Un objeto es un residuo, cuando no hay referencias al objeto