Instituto Tecnológico de Costa Rica

Programa de Maestría en Ciencia de la Computación

I Semestre 2017

Sistemas Operativos Avanzados

Estudiante: Oscar Rodríguez Arroyo

# Tarea 3

**1. ¿En qué consiste el “Problema de Consenso” en Sistemas Distribuidos? ¿Qué algoritmos o enfoques hay al respecto?**

En Sistemas Distribuidos el problema del consenso consiste en que se llegue a un acuerdo con respecto a un valor de datos simple entre varios procesos. Esto se logra mediante protocolos que deben ser tolerantes a fallos ya que los procesos involucrados pueden fallar o no ser del todo confiables. La idea es que cada proceso proponga un valor candidato, se comunique con los otros procesos y finalmente se llegue a un acuerdo seleccionando un único valor.

Por ejemplo, en una base de datos distribuida todas las bases de datos deben llegar a un consenso de si a una transacción se le hace *commit* o se le hace *rollback*. Es importante, en este caso, que todas realicen solamente una de esas dos operaciones y debe ser la misma en todas.

Se dice que un correcto consenso debe cumplir con las siguientes propiedades:

* **Terminación**: Todos los procesos correctos eventualmente toman una decisión.
* **Acuerdo:** Todos los procesos correctos seleccionan el mismo valor.
* **Integridad:** Todos los procesos que deciden seleccionan el valor correcto.

**Algoritmos de consenso:**

**Algoritmo 1:** Los procesos siguen varias rondas de manera incremental (1 … n):

* + En cada ronda, el proceso con el id correspondiente a la ronda es el líder.
  + El líder de una ronda decide un valor a proponer y lo envía a todos (broadcast).
  + Un proceso que no es líder en una ronda espera a recibir la propuesta enviada por el líder para aceptarla o para rechazar (sospechar) al líder.
  + Finalmente, se elige el valor propuesto por el líder no rechazado con el id más bajo.

**Algoritmo 2:** Los procesos siguen rondas incrementales (i =1 … n):

* + El proceso i envía su propuesta actual a todos los demás procesos.
  + Todos los procesos diferentes de i pueden adoptar como propuesta actual la que acaban de recibir.
  + Cada proceso toma una decisión basado en su propuesta actual al final de n rondas.

**Algoritmo 3:** Este algoritmo también se basa en rondas:

* Los procesos se mueven incrementalmente durante n rondas (i = 1 … n)
* El proceso i es el leader (coordinador) en cada ronda k tal que k mod n = i.
* En cada ronda, el proceso i trata de llegar a un acuerdo.
* El algoritmo termina cuando un proceso finalmente logra imponer una decisión (nadie lo rechaza)

**2. Investigue el algoritmo de Maekawa para resolver el problema de la región crítica en un ambiente distribuido. Explíquelo con un ejemplo y compárelo con el algoritmo de Ricart y Agrawalla**

El algoritmo de Maekawa para lograr exclusión mutua en un sistema distribuido consiste en el siguiente:

* Cada nodo del sistema distribuido tiene un identificador único y puede enviar diferentes tipos de peticiones las cuales tienen asignada una prioridad.
* Cuando un proceso recibe una petición (*request*) de un nodo i, la almacena en una cola ordenada por prioridad.
  + Si no ha concedido la exclusión a otro nodo, envía un mensaje para conceder(*grant*) la exclusión mutua.
  + Si ya la ha concedido, comprueba si i tiene menos prioridad que el proceso al que se la ha concedido,
    - Si es así, contesta a i con un mensaje de fallo (*failed*).
    - Si i tiene más prioridad, se envía un mensaje al proceso al que se le dio permiso preguntando (*inquire*) si ha obtenido permiso del resto de los procesos.
* Cuando un proceso recibe un mensaje de *inquire* contesta con un mensaje ceder (*yield*), si ha recibido algún mensaje de fallo.
* Cuando un proceso recibe un mensaje *yield*, almacena en la cola la petición a la que había concedido la exclusión mutua y se la concede al proceso de mayor prioridad de la cola (envía un mensaje *grant*).

Comparación con el algoritmo de Ricart y Agrawalla:

Algoritmo de Ricart y Agrawalla:

* El solicitante envía a todos los procesos un mensaje con su nombre, la sección crítica y una marca del tiempo.
* El receptor del mensaje de acceso a región crítica:
  + Si no está en la sección crítica, envía un mensaje de confirmación al solicitante.
  + Si está en la sección crítica, no responde (o deniega) y encola la petición.
  + Si está esperando a entrar a la sección crítica, compara la marca de tiempo del solicitante con la de su propia petición. Si la del solicitante es anterior, le envía la confirmación; si no, no responde (o deniega) y encola su petición.

|  |  |
| --- | --- |
| Algoritmo de Maekawa | Algoritmo de Ricart y Agrawalla |
| Para entrar en la región crítica hacen falta 2√n mensajes: √n para solicitar la exclusión mutua y el mismo número de respuestas | Para poder entrar a la sección crítica, un proceso tiene que enviar solicitudes a todos los procesos y recibir confirmaciones de todos los procesos (2n) |
| Para salir de la región crítica hacen falta √n mensajes. | Para salir de la zona crítica, un proceso envía confirmaciones a todos los procesos que tiene en su cola. |
| No es un algoritmo centralizado:   * Cada proceso es un cuello de botella * No tiene sentido para pocos procesos | No es un algoritmo centralizado:   * Cada proceso es un cuello de botella * No tiene sentido para pocos procesos |
| Complejidad computacional más baja por el número de mensajes que se envían. | Alta complejidad computacional (broadcast cada vez que se quiere entrar a región crítica). |
| Más complejo de entender por todos los tipos diferentes de solicitudes que viajan entre nodos | Fácil de entender. |

**3. Investigue el algoritmo de Chandy y Lamport para obtener un “distributed snapshot”. Explíquelo con un ejemplo. Mencione sus ventajas y desventajas.**

La idea es guardar un estado global consistente de un sistema distribuido asíncrono. El algoritmo es el siguiente:

1. Un proceso observador (el cual realiza el snapshot del Sistema) realiza lo siguiente:
   1. Guarda su propio estado local
   2. Envía un mensaje de solicitud de snapshot a todos los otros procesos. Este mensaje contiene un identificador (*token*) de snapshot.
2. Un proceso que recibe un *token* de snapshot por primera vez realiza lo siguiente:
   1. Le envía al proceso observador su propio snapshot.
   2. Adjunta el *token* de snapshot a todos los mensajes que lleguen posteriormente, de forma que este se propague.
3. Un proceso que ya recibió el *token* de snapshot y recibe un mensaje que no contiene dicho *token*, realiza lo siguiente:
   1. Reenvía el mensaje al proceso observador ya que obviamente este mensaje fue enviado antes del corte del snapshot (al no tener el *token* de snapshot adjunto significa que fue enviado antes de la creación de ese *token*).
   2. La idea es que el mensaje sin *token* sea incluido en el snapshot.
4. El observador finalmente construye un snapshot completo, que incluye el estado de cada uno de los procesos y todos los mensajes.

**Ventajas**

* Algoritmo simple y fácil de entender.
* Garantiza que todos los estados de todos los procesos al momento en que se inició la solicitud de snapshot sean almacenados (funciona).
* No depende de ningún nodo o proceso central (cualquiera puede iniciar la solicitud de snapshot)

**Desventajas**

* No considera los casos en los que ocurra algún fallo en la red.
* El algoritmo no está diseñado para sistemas distribuidos en los que los procesos comparten reloj y memoria.