Tarea 8 - Cómputo distribuido

Dalia Camacho, Gabriela Vargas

El objetivo del algoritmo *Total order broadcast* es garantizar que los mensajes enviados a un conjunto de procesos sean recibidos en el mismo orden.

1 Algoritmo *TO-broadcast-based universal construction*

Este algoritmo replica las características de una máquina de estado O que opera de forma secuencial.

O tiene las siguientes características:

- Conjunto de operaciones $op_x()$
- Función de transición determinista $\delta()$
- La variable de *state* que indica el estado en el que se encuentra el objeto.

La función $\delta(state, op_x(param_x)$ regresa la dupla $< state', res >$, donde $state'$ respresenta el nuevo estado del objeto y res es el resultado $op_x(param_x)$
.

Los procesos realizan dos operaciones básicas en este algoritmo: **TO_broadcast(m)**, para enviar un mensaje a todos los procesos y **TO_deliver()**, para cada proceso que reciba el mensaje. La idea es que cada proceso p_i tenga una copia $state_i$ del objeto y que todos los p_i apliquen la misma secuencia de operaciones a su copia local del objeto O.

El algoritmo se describe de la siguiente manera:

Cuando algún proceso p_i invoca $op_x(param_x)$ en su copia de O, $result_i$ se inicializa en vacío y la variable $sent\_msg$ toma el valor $< op_x(param_x), i >$. p_i entonces llama la función $TO\_broadcast()$ tomando como parámetro el valor de $sent\_msg$ y cuando el valor de $result_i$ sea diferente del vacío, p_i regresa $result_i$.

Simultáneamente, los demás procesos realizan la siguiente tarea: Repetir indefinidamente Asignar la variable rec_msg con el resultado de la función $TO\_deliver$. El conjunto de variables $< state\_i, res >$ se asigna con el resultado de la función de transición $\delta(state_i, rec_msg.op)$ y si el valor de $rec_msg.proc$ es i entonces $result_i$ toma el valor res.

La especificación formal del algoritmo es la siguiente:

- **Cuando** la operación $op_x(param_x)$ sea invocada por p_i **hacer**

$result_i \leftarrow \emptyset$; $sent\_msg =< op_x(param_x), i >$;

$TO\_broadcast(sent_msg)$;

esperar($result_i \neq \emptyset$); regresar($result_i$).

- Operación de fondo T
  Repetir por siempre

  $rec\_msg \leftarrow TO\_deliver()$;

  $< state\_i, res > \leftarrow \delta(state_i, rec\_msg.op)$;

  si $(rec\_msg.proc = i)$ **entonces** $result_i \leftarrow res$

2 Algoritmo: *Implementing TO-broadcast from consensus*

Este algoritmo muestra cómo hacer *deliver* de los mensanjes m dentro de un *broadcast* si hay un objeto de consenso que determine el siguiente elemento que entra en el conjunto $to\_deliverable_i$. Consideramos que existen p_i procesos, un conjunto de mensajes pendientes para cada proceso i llamado $pending_i$, un conjunto de mensaje ordenados listos para entregar llamado $to\_deliverable_i$ y un objeto de cosenso $CS[k]$ asociado a la iteración del proceso k. $CS[k]$ recibe como entrada una secuencia propuesta previamente ordenada y regresa una secuencia.

El algoritmo se puede dividir en cuatro tipos de acciones posibles.

- Ante el envío de m con un broadcast envía el mensaje a sí mismo.
- Ante la recepción de un mensaje m por vez primera hace un broadcast[1] con m y lo guarda en el conjunto $pending_i$.
- Si el conjunto $to\_deliverable_i$ es no vacío ejecución de la functión *TO_deliver()* del primer mensaje en el conjunto.
- Esperar hasta que haya algún mensaje en $pending_i$ que no esté en $to\_deliverable_i$, todos los mensajes que cumplan esta propiedad se guardan en una secuencia *seq* y se ordenan. Se incrementa el contador de iteraciones sn_i y al objeto de consenso $CS[sn_i]$ se le da como entrada la secuencia *seq*. $CS[sn_i]$ regresa una secuencia que será guardada en el conjunto $to\_deliverable_i$ en espera de que se ejecute la acción *TO_deliver()*.

Estas acciones ocurren de manera concurrete entre cada proceso, sin embargo todo proceso correcto debe hacer el *deliver* en el mismo orden.

El algoritmo se presenta más formalmente como sigue:

- **Cuando** p_i invoca la función $TO\_broadcast(m)$ también envía el mensaje m a sí mismo.

[1] Al hacer el broadcast envía el mensaje m a todo proceso p_j.

- **Cuando** p_i recibe m por primera vez hace un $broadcast(m)$ y guarda m en $pending_i$. $pending_i \leftarrow pending_i \cup \{m\}$.
- **Cuando** $to\_deliverable_i$ **es no vacío** seam $m =$ primer mensaje $\in to\_deliverable_i$ que aún no se ha entregado. Se toma m y se hace *TO_deliver(m)*.
- *Tarea realizada de fondo T*

  Repetir por siempre

  esperar($pending_i - to\_deliverable_i \neq \emptyset$)

  Sea $seq = pending_i - to\_deliverable_i$

  Ordenar los mensajes en seq

  $sn_i \leftarrow sn_i + 1$

  $res_i \leftarrow CS[sn_i].propose(seq)$

  se agrega res_i como último elemento de $pending_i - to\_deliverable_i$

3 Resumen de la clase

En esta clase hicimos un repaso del problema de consenso bajo los siguientes rubros: la definición estricta, definiciones débiles, entradas posibles como valores propuestos, usos y aplicaciones fallas y temporalidad. El esquema general se encuentra en la Figura 1, los tipos de fallas se encuentran especificados en la Tabla 1.

Figure 1: Conceptos principales en el problema de consenso.

Temporalidad	Características
Asíncrono	No se puede resolver, esto se demostró con el teorema FLP. Ni siquiera se puede resolver un caso general del $(n-1)$-acuerdo. Bajo algunas condiciones de entrada es posible tener una solucion suficientemente buena para un ϵ-aproximado para algún ϵ dado.
Semisíncrono	Se puede resolver en algunos casos dependiendo de δ que es la cota sobre los retardos y de ϵ que es la cota de velocidad relativa.
Síncrono	Se puede resolver en $t+1$ rondas cuando hay t fallas de tipo *crash*. Ante fallas por omisión se puede resolver siempre y cuando $t < \frac{n}{2}$. Ante fallas bizantinas malignas se puede resolver siempre y cuando $3t + 1 \leq n$.

Table 1: Tipo de fallas

Más adelante vimos el problema de *broadcast* en que un proceso envía un mensaje a todos los otros procesos. Para que haya acuerdo en el *broadcast* todos los procesos correctos deben entregar (*deliver*) m.

Reliable broadcast es un caso particular del problema de broadcast cuyas propiedades son las siguientes:

- **Validity:** Si un proceso correcto difunde un mensaje m, entonces todos los procesos eventualmente recibirán m.
- **Agreement:** Si un proceso correcto recibe un mensaje m, entonces todos los procesos eventualmente recibirán m.
- **Integrity:** Para cada mensaje m, un proceso correcto recibe m por lo menos una vez si y solo si m fue previamente enviado por el proceso que lo difundió.

Dentro de la categoría *Reliable broadcast*, encontramos los siguientes algoritmos:

- **FIFO broadcast:** Si un proceso difunde m_1 antes de difundir m_2, entonces ningún proceso correcto recibirá m_2 antes de recibir m_1.
- **Causal broadcast:** Si la difusión del mensaje m_1 precede causalmente la difusión del mensaje m_2, entonces ningún proceso correcto recibirá m_2 antes de recibir m_1. La categoría de *FIFO broadcast* está incluida dentro de esta categoría.
- **Total order or atomic boradcast:** Para dos mensajes m_1 y m_2, si el proceso P_1 recibe m_1 sin haber recibido m_2, entonces ningún proceso P_2 recibirá m_2 antes que m_1.

  .

References