Tiempo, Estado y Relojes

30221 - Sistemas Distribuidos

Rafael Tolosana Calasanz

Dpto. Informática e Ing. de Sistemas

Lectura Recomendada

- G. Colouris, J. Dollimore, T. Kindberg and G. Blair. Distributed systems: Concepts and Design. 5th Edition. Addison-Wesley. May, 2011. ISBN: 978-0132143011. **Chapter 14**
- Raynal, M. (2013). Distributed Algorithms for Message-Passing Systems: **Chapters 6, 10, 14**
- Tanenbaum Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall. **Chapter 6**

Motivación

Motivación

Escalabilidad del Patrón Mutex Distribuido

P

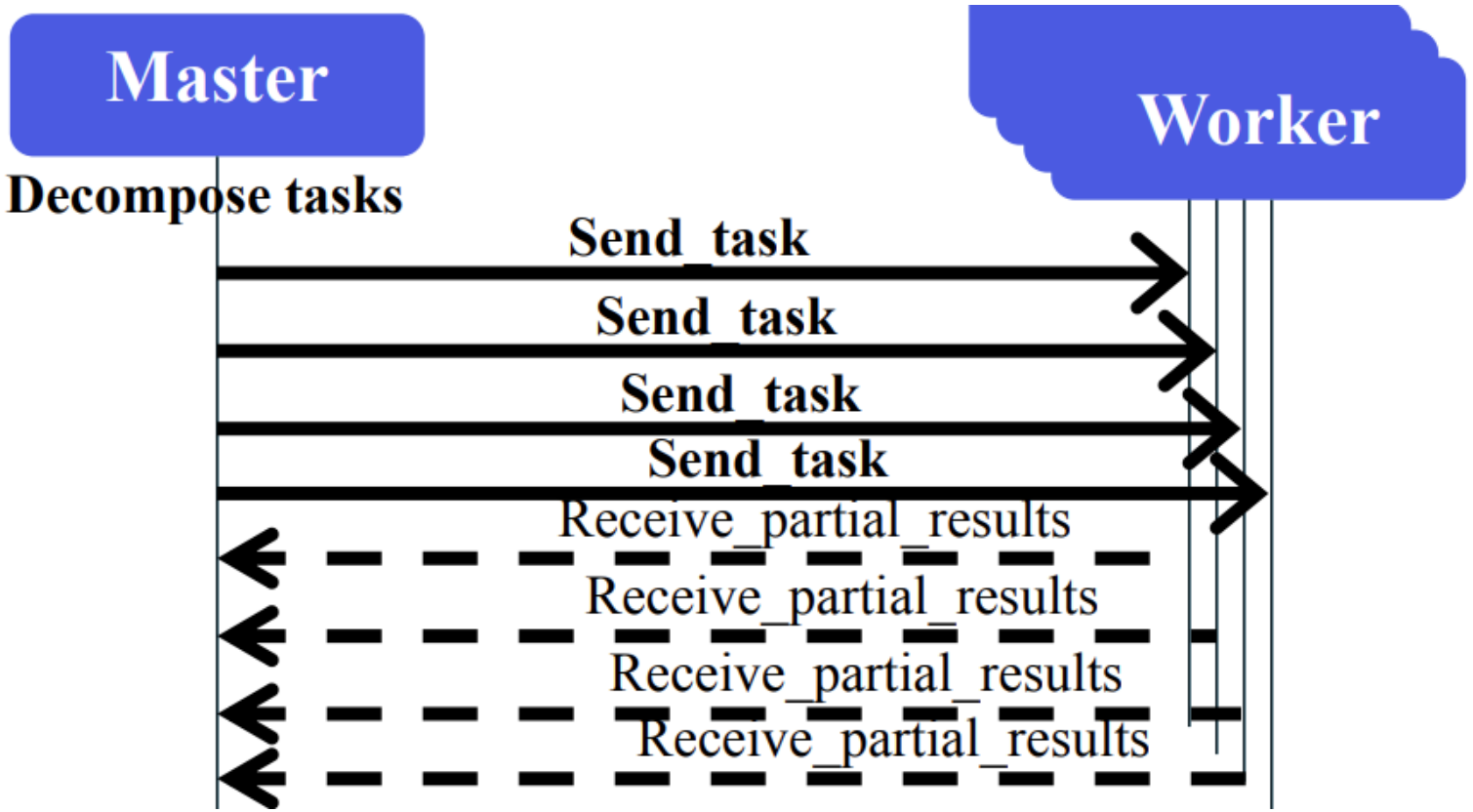
```
Integer action; PID pid; Integer counter = 0; FIFOQueue q;
While true
p1  receive(pid, x)
p2  if (x == wait && counter ==0)
p3      q.add(pid)
p4  else if(x == wait && counter == 1)
p5      send(pid, ok)
p6      counter--
p7  else // signal
p8      counter++
p9      if(!q.isEmpty())
p10             send(q.remove(), ok);
p11             counter--
```

Qi

```
Integer y; PID pid;
q1 send(P, wait)
q2 receive(pid, ok)
q3 SC
q3 send(P, signal)
```

Motivación

Escalabilidad de la Arquitectura Master-Worker



Motivación

Algoritmos Centralizados

- **Ventajas**: simplicidad, fairness
- **Inconvenientes**: Escalabilidad, mucha responsabilidad en un punto

Algoritmos Descentralizados

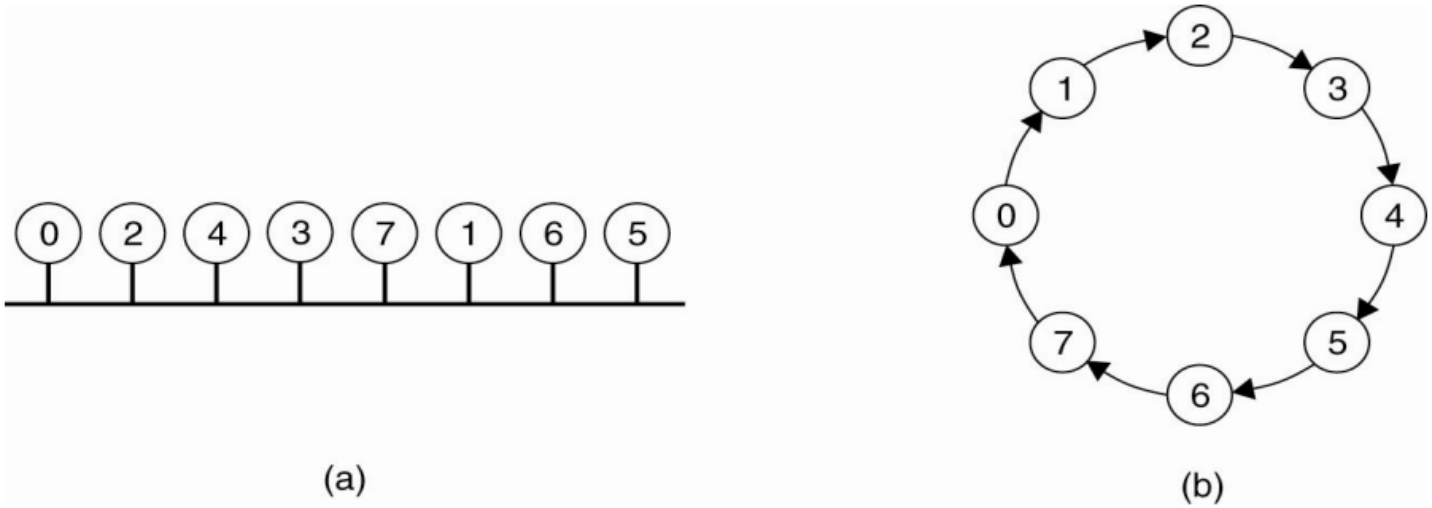
Algoritmos Descentralizados

Características

- No existe ningún proceso en el SD que tenga más responsabilidad que otro
- Ningún proceso tiene el conocimiento del estado global del SD
- Los procesos basan sus decisiones exclusivamente en información local y en la ordenación de eventos.
- Se asume alguna forma de IPC (comunicación) entre procesos.
  - directa, indirecta, asíncrona, síncrona, etc.

Token Ring

Algoritmo Mútex Distribuido en Anillo



(a) An unordered group of processes on a network.
(b) A logical ring constructed in software.

Algoritmo de Lamport - 1981

Algoritmo Mútex Distribuido de Lamport, 1978

- Evolución del Algoritmo de la Panadería
- **Idea**
  - Tengo que conocer a todos los procesos participantes
  - Los participantes tienen su reloj lógico propio
  - Envío a todo el mundo un *request* con mi reloj
  - Espero el permiso de todo el mundo
  - Entro en la SC
  - Concedo el permiso a todos aquellos que habían quedado esperándolo

Algoritmo de Lamport - 1981

Requisitos de Comunicación

- Se asume que no hay fallos de red (mensajes llegan)

Algoritmo de Lamport - 1981

Requisitos de Comunicación

- Se asume que no hay fallos de red (mensajes llegan)

Idea de Implementación

- Cada proceso mantiene un *heap*
  - Contiene solicitudes de acceso a la sección crítica
  - Ordenado por tiempo lógico
  - Cuando un proceso i recibe un mensaje request(j, T_j) de acceso a la sección crítica de un proceso j, lo añade al *heap*

Algoritmo de Lamport - 1981

Algoritmo de Lamport

- **Preprotocol de P_i**
  - P_i envía *request*(i, T_i) a todos los procesos (N – 1)
  - P_i introduce su propia petición en su heap
  - P_i espera las (N - 1) *respuestas ack*(i, T_i)
  - Cuando **recibe todas** las respuestas ack y su propia petición está la **primera** en el heap, accede a la SC
- **Postprotocol**
  - Quita su propia petición del heap
  - Envía un mensaje de *release*(i, T_i') con timestamp a todos los procesos (N – 1)

Algoritmo de Lamport - 1981

¿Cuánto cuesta?

- En términos de intercambio de mensajes
  - envío **N - 1** *request*(i, T_i)
  - espera **(N - 1)** *ack*(i, T_i)
  - envío **N - 1** *release*(i, T_i')

Algoritmo de Lamport - 1981

¿Cuánto cuesta?

- En términos de intercambio de mensajes
  - envío **N - 1** *request*(i, T_i)
  - espera **(N - 1)** *ack*(i, T_i)
  - envío **N - 1** *release*(i, T_i')

- **TOTAL** $3x(N-1)$

Algoritmo de Ricart-Agrawala - 1981

Requisitos y Características

- **N procesos** distribuidos que no comparten memoria y se comunican exclusivamente mediante el paso de **mensajes**
- La **red** de comunicación subyacente está **libre de errores**
- Los mensajes pueden llegar fuera de orden
  - orden de recepción distinto de envío
- No hay heap

Algoritmo de Ricart-Agrawala - 1981

Algoritmo de Ricart-Agrawala

- **Comportamiento** en general
  - Si Pi desea acceder a la SC y recibe una petición de acceso a la SC (request) con más timestamp, la posterga (y la almacena en una cola)
  - Si no desea acceder a la SC o recibe una petición de acceso a la SC (request) de menos timestamp, envía inmediatamente el ack

Algoritmo de Ricart-Agrawala - 1981

Algoritmo de Ricart-Agrawala

- **Preprotocol de P_i**
  - P_i envía *request*(i, T_i) a todos los procesos (N – 1)
  - P_i espera las (N - 1) *respuestas ack (sin estampilla)*
  - Cuando **recibe todas** las respuestas ack, accede a la SC
- **Postprotocol**
  - Envía un mensaje de ack para todos los mensaje postergados

Algoritmo de Ricart-Agrawala - 1981

¿Cuánto cuesta?

- En términos de intercambio de mensajes
  - envío **N - 1** *request*(i, T_i)
  - espera **(N - 1)** *ack*

Algoritmo de Ricart-Agrawala - 1981

¿Cuánto cuesta?

- En términos de intercambio de mensajes
  - envío **N - 1** *request(i, T_i)*
  - espera **(N - 1)** *ack*

- **TOTAL** $2x(N-1)$

Algoritmo de Ricart-Agrawala - 1981

Lamport vs. Ricart-Agrawala I

- En el Algoritmo de Lamport
  - Todos los procesos responden siempre, sin postergación
  - Un proceso decide acceder a la SC en función de si su request está la primera en el montículo

Algoritmo de Ricart-Agrawala - 1981

Lamport vs. Ricart-Agrawala I

- En el Algoritmo de Lamport
  - Todos los procesos responden siempre, sin postergación
  - Un proceso decide acceder a la SC en función de si su request está la primera en el montículo

- En el Algoritmo de Ricart-Agrawala
  - Los procesos solo responden si no desean acceder a la SC o si tienen mayor timestamp
  - Un proceso decide acceder a la SC si ha recibido todos los acks

Algoritmo de Ricart-Agrawala - 1981

Lamport vs. Ricart-Agrawala II

- En el Algoritmo de Lamport
  - Envío y Recepción de mensajes son eventos
  - Se siguen las reglas de incremento de relojes de Lamport

Algoritmo de Ricart-Agrawala - 1981

Lamport vs. Ricart-Agrawala II

- En el Algoritmo de Lamport
  - Envío y Recepción de mensajes son eventos
  - Se siguen las reglas de incremento de relojes de Lamport

- En el Algoritmo de Ricart-Agrawala
  - Solo el envío y la recepción de requests es un evento
  - El reloj está implementado en dos variables
    - Eso permite accesos consecutivos a la SC por un mismo proceso con el mismo reloj

Algoritmo de Ricart-Agrawala - 1981

Implementación Original en Algol I

```
SHARED DATABASE
CONSTANT
        me, ! This node's unique number
        N;  ! The number of nodes in the network
INTEGER Our_Sequence_Number,
            ! The sequence number chosen by a request
            ! originating at this node
        Highest_Sequence_Number initial (0),
            ! The highest sequence number seen in any
            ! REQUEST message sent or received
        Outstanding_Reply_Count;
            ! The number of REPLY messages still
            ! expected
BOOLEAN Requesting_Critical_Section initial (FALSE),
            ! TRUE when this node is requesting access
            ! to its critical section
        Reply_Deferred [1:N] initial (FALSE);
            ! Reply_Deferred [j] is TRUE when this node
            ! is deferring a REPLY to j's REQUEST message
BINARY SEMAPHORE
        Shared_vars initial (1);
            ! Interlock access to the above shared
            ! variables when necessary
```

Algoritmo de Ricart-Agrawala - 1981

Implementación Original en Algol II

```
PROCESS WHICH INVOKES MUTUAL EXCLUSION FOR
THIS NODE
Comment Request Entry to our Critical Section;
   P (Shared_vars)
      Comment Choose a sequence number;
      Requesting_Critical_Section := TRUE;
      Our_Sequence_Number := Highest_Sequence_Number + 1;
   V (Shared_vars);
   Outstanding_Reply_Count := N − 1;
   FOR j := 1 STEP 1 UNTIL N DO IF j ≠ me THEN
         Send_Message(REQUEST(Our_Sequence_Number,me),j);
   Comment sent a REQUEST message containing our sequence num-
   ber and our node number to all other nodes;
Comment Now wait for a REPLY from each of the other nodes;
   WAITFOR (Outstanding_Reply_Count = 0);
Comment Critical Section Processing can be performed at this point;
Comment Release the Critical Section;
   Requesting_Critical_Section := FALSE;
   FOR j := 1 STEP 1 UNTIL N DO
      IF Reply_Deferred[j] THEN
      BEGIN
         Reply_Deferred[j] := FALSE;
         Send_Message (REPLY, j);
         Comment send a REPLY to node j;
      END;
```

Pre-protocol

Post-protocol

Algoritmo de Ricart-Agrawala - 1981

Implementación Original en Algol III

```
PROCESS WHICH RECEIVES REQUEST (k, j) MESSAGES
Comment  k is the sequence number begin requested,
         j is the node number making the request;

BOOLEAN Defer_it    ;
                    ! TRUE when we cannot reply immediately
Highest_Sequence_Number :=
    Maximum (Highest_Sequence_Number, k);
P (Shared_vars);
    Defer_it :=
        Requesting_Critical_Section
        AND ((k > Our_sequence_Number)
                OR (k = Our_Sequence_Number AND j > me));
V (Shared_vars);
Comment Defer_it will be TRUE if we have priority over
  node j's request;
IF Defer_it THEN Reply_Deferred[j] := TRUE ELSE
    Send_Message (REPLY, j);

PROCESS WHICH RECEIVES REPLY MESSAGES
Outstanding_Reply_Count := Outstanding_Reply_Count − 1;
```

Algoritmo de Ricart-Agrawala - 1981

Abstracción de la Implementación en Algol

operation acquire_mutex() **is**
(1) $cs\_state_i \leftarrow trying$;
(2) $\ell rd_i \leftarrow clock_i + 1$;
(3) $waiting\_from_i \leftarrow R_i$; % $R_i = \{1, \ldots, n\} \setminus \{i\}$
(4) **for each** $j \in R_i$ **do** send REQUEST$(\ell rd_i, i)$ to p_j **end for**;
(5) **wait** $(waiting\_from_i = \emptyset)$;
(6) $cs\_state_i \leftarrow in$.

operation release_mutex() **is**
(7) $cs\_state_i \leftarrow out$;
(8) **for each** $j \in perm\_delayed_i$ **do** send PERMISSION(i) to p_j **end for**;
(9) $perm\_delayed_i \leftarrow \emptyset$.

when REQUEST(k, j) **is received do**
(10) $clock_i \leftarrow \max(clock_i, k)$;
(11) $prio_i \leftarrow (cs\_state_i \neq out) \wedge (\langle \ell rd_i, i \rangle < \langle k, j \rangle)$;
(12) **if** $(prio_i)$ **then** $perm\_delayed_i \leftarrow perm\_delayed_i \cup \{j\}$
(13) **else** send PERMISSION(i) to p_j
(14) **end if**.

when PERMISSION(j) **is received do**
(15) $waiting\_from_i \leftarrow waiting\_from_i \setminus \{j\}$.

[1]

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

Algoritmo de Ricart-Agrawala - 1981

Propiedades Ricart-Agrawala

- Acceso en Exclusión Mutua
- Ausencia de Bloqueos (deadlocks)
  - ¿Todos los procesos se quedan bloqueados en el preprotocol?
- Ausencia de Inanición
  - ¿Algún proceso se queda bloqueado en el preprotocol?

Generalizaciones del Mútex Distribuido

Algoritmo de Raymond - 1989

Algoritmo Multiplex de Acceso a la SC

- En un SD con N procesos
- Para cualquier instante, como máximo puede haber K procesos en SC, $K < N$
- Es una generalización del Algoritmo de Ricart-Agrawala

Idea Intuitiva

- Como máximo pueden entrar simultáneamente K procesos
- Un proceso puede entrar en la SC tan pronto como reciba N-K ACKs
- El resto de ACKs podrían llegar cuando el proceso está:
  - en la SC
  - esperando otra vez para acceder a la SC
- Por tanto, se debe tener en cuenta el número de ACKs y no contarlos para las sucesivas peticiones

Lectores y Escritores Distribuidos

Lectores y escritores

- Es una generalización del problema de la sección crítica.
- Consiste en 2 operaciones, read y write, que tienen que realizarse en exclusión mutua
- No puede haber 1 read y 1 write simultáneamente
- Puede haber varios read simultáneos
- No puede haber varios writes simultáneos

Lectores y Escritores Distribuidos

Lectores y escritores Distribuidos

- Se pueden considerar varias operaciones y varias reglas de exclusión entre ellas
  - las reglas se representan mediante matriz de concurrencia
- Se define la matriz de concurrencia, *exclude*, como una matriz de booleanos que es simétrica [2]
- Por ejemplo, sean op1 = read y op2 = write:
  - La matriz de concurrencia podría definir las exclusiones:
  - Exclude[read, write] = Exclude[write, write] = true
  - Exclude[read , read] = false

[2] Simétrica quiere decir que exclude[op1,op2]= exclude[op2,op1]

Lectores y Escritores Distribuidos

Implementación de Lectores Escritores Distribuidos

operation acquire_mutex() **is**
(1) $cs\_state_i \leftarrow trying$;
(2) $\ell rd_i \leftarrow clock_i + 1$;
(3) $waiting\_from_i \leftarrow R_i$; % $R_i = \{1, \ldots, n\} \setminus \{i\}$
(4) **for each** $j \in R_i$ **do** send REQUEST$(\ell rd_i, i)$ to p_j **end for**;
(5) **wait** $(waiting\_from_i = \emptyset)$;
(6) $cs\_state_i \leftarrow in$.

operation release_mutex() **is**
(7) $cs\_state_i \leftarrow out$;
(8) **for each** $j \in perm\_delayed_i$ **do** send PERMISSION(i) to p_j **end for**;
(9) $perm\_delayed_i \leftarrow \emptyset$.

when REQUEST(k, j) **is received do**
(10) $clock_i \leftarrow \max(clock_i, k)$;
(11) $prio_i \leftarrow (cs\_state_i \neq out) \wedge (\langle \ell rd_i, i \rangle < \langle k, j \rangle)$;
(12) **if** $(prio_i)$ **then** $perm\_delayed_i \leftarrow perm\_delayed_i \cup \{j\}$
(13) **else** send PERMISSION(i) to p_j
(14) **end if**.

when PERMISSION(j) **is received do**
(15) $waiting\_from_i \leftarrow waiting\_from_i \setminus \{j\}$.

3

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

Lectores y Escritores Distribuidos

Implementación de Lectores Escritores Distribuidos

operation begin_op() **is**
(1) $cs\_state_i \leftarrow trying$;
(2) $\ell rd_i \leftarrow clock_i + 1$;
(3) $waiting\_from_i \leftarrow R_i$; % $R_i = \{1, \ldots, n\} \setminus \{i\}$
(4′) **for each** $j \in R_i$ **do send** REQUEST$(\ell rd_i, i, op_type)$ to p_j **end for**;
(5) **wait** $(waiting\_from_i = \emptyset)$;
(6) $cs\_state_i \leftarrow in$.

operation end_op() **is**
(7) $cs\_state_i \leftarrow out$;
(8) **for each** $j \in perm\_delayed_i$ **do send** PERMISSION(i) to p_j **end for**;
(9) $perm\_delayed_i \leftarrow \emptyset$.

when REQUEST(k, j, op_t) **is received do**
(10) $clock_i \leftarrow \max(clock_i, k)$;
(11′) $prio_i \leftarrow (cs\_state_i \neq out) \wedge (\langle \ell rd_i, i \rangle < \langle k, j \rangle) \wedge exclude(op\_type, op\_t)$;
(12) **if** $(prio_i)$ **then** $perm\_delayed_i \leftarrow perm\_delayed_i \cup \{j\}$
(13) **else** send PERMISSION(i) to p_j
(14) **end if**.

when PERMISSION(j) **is received do**
(15) $waiting\_from_i \leftarrow waiting\_from_i \setminus \{j\}$.

4

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

Resumen

Resumen

- Algoritmos Centralizados
- Algoritmos Mutex Descentralizados
  - Token Ring
  - Algoritmo de Lamport (relojes de Lamport)
  - Algoritmo de Ricart-Agrawala (relojes de Ricart-Agrawala)
- Algoritmo Multiplex Distribuido
- Lectores y Escritores Distribuido

Tiempo, Estado y Relojes

30221 - Sistemas Distribuidos

Rafael Tolosana Calasanz

Dpto. Informática e Ing. de Sistemas