Rev. Fac. Ing. - Univ. Tarapacá, vol. 13 no. 1, 2005, pp. 89-98

UN NUEVO ALGORITMO DISTRIBUIDO DE EXCLUSIÓN MUTUA QUE MINIMIZA EL INTERCAMBIO DE MENSAJES

Jorge Pérez Rojas[1] Christian F. Orellana[2]

Recibido el 7 de enero de 2004, aceptado el 5 de noviembre de 2004

RESUMEN

En este artículo presentamos un nuevo algoritmo de exclusión mutua distribuida basado en paso de token. Nuestro algoritmo utiliza dos estructuras dinámicas y distribuidas para proveer exclusión mutua: el *Bosque de Naimi* para dirigir las peticiones por el token y el *Árbol Virtual de Raymond* para servirlas. La estrategia utilizada combina las mejores características de dos algoritmos anteriores, citados en la literatura como los más eficientes en cuanto al tráfico de mensajes. Presentamos un estudio de desempeño mediante técnicas de simulación. Los resultados indican que nuestro algoritmo es el de mejor desempeño en cuanto al número de mensajes intercambiados por ingreso a sección crítica.

Palabras clave: Exclusión mutua distribuida, sincronización, algoritmos distribuidos.

ABSTRACT

In this paper we present a new token-based distributed mutual exclusion algorithm. The algorithm relies on two distributed dynamic structures in order to provide mutual exclusion: the "Naimi Forest" to route token requests, and the "Raymond Virtual Tree" to serve token requests. Our strategy combines the best characteristics of two algorithms cited as the most efficient in the literature concerning message traffic. We present a performance simulation study. Results show that our algorithm has better performance considering the number of message exchanged per critical section entry.

Keywords: Distributed mutual exclusion, distributed synchronization, distributed algorithms.

INTRODUCCIÓN

Los algoritmos de exclusión mutua distribuida proveen acceso sincronizado a recursos compartidos en un sistema físicamente distribuido, asegurando en todo momento que a lo más un proceso o nodo pueda estar ejecutando en su sección crítica.

Estos algoritmos pueden ser clasificados en dos grupos: los basados en permisos y los basados en privilegio o paso de *token* [9]. En los algoritmos del primer grupo, un nodo que quiere entrar a su sección crítica debe obtener el permiso de un subconjunto de nodos [10], [5], [1]. Los del segundo grupo confían en la existencia de un único token, que debe ser adquirido por un nodo que quiere entrar a su sección crítica [8], [6], [2], [7], [11], [12].

Los algoritmos basados en permisos no presentan un buen desempeño en cuanto al número de mensajes intercambiados por ingreso a sección crítica, al compararlos con los basados en token.

En el primer grupo, Ricart y Agrawala proponen un algoritmo que intercambia un número de mensajes proporcional a N, donde N es el número de nodos en el sistema [10]. Maekawa presenta un algoritmo, también basado en permisos, donde el número de mensajes intercambiados es proporcional a $\sqrt{N}$ [5]. En el segundo grupo existen varios algoritmos que intercambian un número de mensajes proporcional a $\log(N)$ en el caso promedio [8], [6], [7].

En el algoritmo propuesto por Raymond [8], si bien el número de mensajes es proporcional a $\log(N)$ en el caso promedio, en situaciones donde la demanda por el token es baja, la cantidad de mensajes es mucho mayor que $\log(N)$, mientras que en alta demanda es menor. En el algoritmo propuesto por Naimi et al. [6] la cantidad de mensajes es también $\log(N)$ en el caso promedio, con un mejor comportamiento en baja demanda comparado con el algoritmo de Raymond. En la literatura se cita al algoritmo de Raymond como el que alcanza un mejor desempeño en cuanto al número de mensajes cuando hay

[1] Departamento de Ingeniería de Sistemas, Universidad de Talca, Camino Los Niches Km. 1, Curicó, Chile. jperez@utalca.cl
[2] Departamento de Ciencias de la Computación, P. Universidad Católica de Chile, Vic. Mackenna 4860, Santiago, Chile. cforella@ing.puc.cl

alta demanda por el token, y al de Naimi como el de mejor desempeño cuando la demanda es baja.

En esta comparación se consideran sólo los algoritmos que no tienen un coordinador central y que intercambian mensajes de tamaño constante.

En este artículo proponemos un nuevo algoritmo de paso de token, sin coordinador central y con mensajes de tamaño constante, que combina las mejores características de los algoritmos de Raymond y Naimi. Cuando hay baja demanda por el token, alcanza un desempeño similar al del algoritmo de Naimi. Cuando hay alta demanda por el token, alcanza un desempeño similar al del algoritmo de Raymond.

EL ALGORITMO

En lo que sigue, supondremos que el sistema está compuesto por N nodos que no comparten memoria, donde cada nodo puede enviar mensajes a cualquier otro nodo usando un identificador único como dirección. La comunicación entre los nodos se asume perfecta.

En los algoritmos basados en token, cuando un nodo desea entrar a su sección crítica, primero debe adquirir el token. Sólo el nodo que tiene el token puede decidir cuál será el próximo nodo en tenerlo. Dado que existe sólo un token en el sistema, la exclusión mutua se encuentra garantizada.

Descripción informal

En los algoritmos de Naimi y Raymond, los nodos se organizan en una estructura de árbol, que es utilizada para dirigir las peticiones de ingreso a sección crítica. En el algoritmo de Naimi, el árbol es dirigido y dinámico; en el algoritmo de Raymond, en cambio, es no dirigido y estático.

El algoritmo de Naimi además mantiene una estructura de cola distribuida para servir las peticiones pendientes. Cada nodo sabe el identificador del próximo nodo en la cola. Cuando un nodo *i* que no tiene el token quiere entrar a su sección crítica, envía una petición a través del árbol, siguiendo el único camino desde *i* hasta la raíz. Todos los nodos en el camino, incluyendo a *i*, actualizan su padre al nodo *i*, convirtiéndose *i* en la nueva raíz del sistema. Note que si existen varias peticiones en tránsito pueden existir varios árboles, es decir, un bosque. Cuando todas llegan a destino, el bosque colapsa a un solo árbol. Cuando una petición llega hasta la raíz, se agrega a la cola distribuida. La forma en que el algoritmo de Naimi comprime los caminos lo hace alcanzar un buen desempeño en cuanto al número de mensajes intercambiados [6].

En el algoritmo de Raymond, cada nodo tiene una cola local donde almacena tanto las peticiones por el token recibidas desde sus vecinos en el árbol como las peticiones propias. Un nodo realiza una única petición por el token para servir todas las peticiones en su cola local. En todo momento, cada nodo sabe cuál de sus vecinos está más cerca del nodo que posee el token, y a él le envía su petición. En situaciones de alta carga el número de mensajes que se intercambian para pedir el token es pequeño dado que un nodo, luego de enviar la primera petición por el token, encola todas las siguientes peticiones de sus vecinos sin enviar más mensajes [8].

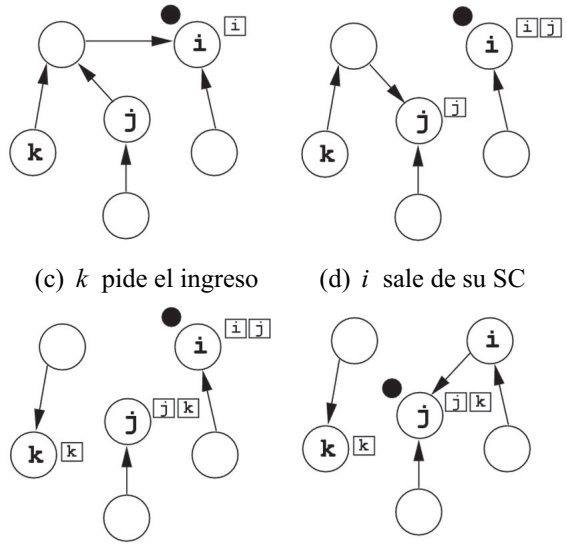
En el algoritmo de Raymond, un nodo que tiene el token lo administra de acuerdo a su cola local. Si su propio identificador se encuentra en la cabeza de la cola, puede ingresar a su sección crítica. Si otro identificador se encuentra a la cabeza, el token le es enviado, pidiéndose su retorno sólo si quedan peticiones pendientes.

En nuestro algoritmo existe una estructura de árbol inicial para dirigir las peticiones y cada nodo cuenta con una cola local para servirlas. Cuando un nodo quiere ingresar a su sección crítica y su cola local está vacía, envía una petición por el token, la que llega hasta el primer nodo que posee peticiones pendientes. Los nodos en el camino, excepto el último, actualizan su padre en el árbol como el nodo que generó la petición al igual que en algoritmo de Naimi. Así, se genera un nuevo árbol en el sistema por cada petición en tránsito. Existirán árboles independientes mientras existan peticiones pendientes en el sistema, a diferencia del algoritmo de Naimi en que los árboles colapsan cuando los mensajes de petición llegan a destino. El último nodo en el camino de una petición la agrega a su cola local como en el algoritmo de Raymond. Un nodo que posee el token lo administra de acuerdo a su cola local, de manera similar al algoritmo de Raymond.

Nuestra propuesta combina entonces las estrategias de Raymond y Naimi para obtener buen desempeño tanto en condiciones de baja como de alta carga. La idea central es aprovechar la compresión de caminos del algoritmo de Naimi y la estrategia de no enviar mensajes adicionales del algoritmo de Raymond. Todos los nodos que no tienen peticiones pendientes funcionarán de acuerdo a las reglas del algoritmo de Naimi. Los que sí las tienen, lo harán de acuerdo a las reglas del algoritmo de Raymond. Así, si hay una única petición pendiente en el sistema, nuestro algoritmo se comporta como el algoritmo de Naimi. Si cada uno de los nodos del sistema tiene una petición pendiente, nuestro algoritmo se comporta como el algoritmo de Raymond.

La Figura 1 muestra una ejecución de ejemplo del algoritmo propuesto. Inicialmente el nodo *i* tiene el token y está ejecutando en su sección crítica. Note que a medida que los nodos realizan sus peticiones aparecen árboles independientes en el sistema (Figuras 1(b) y (c)). Los árboles se unen a medida que las peticiones son servidas (Figura 1(d)).

(a) *i* ejecutando su SC (b) *j* pide el ingreso



(c) *k* pide el ingreso (d) *i* sale de su SC

Fig. 1 Ejemplo de ejecución del algoritmo.

Descripción formal

En nuestro algoritmo, los nodos intercambian dos tipos de mensaje: *Request* y *Token*. El mensaje *Request*(*k*) significa que el nodo *k* está haciendo una petición por el token. Un nodo que recibe un mensaje *Token*(*k*) se convierte en el propietario del token. El argumento de este mensaje indica el sitio al que el token debe ser devuelto cuando ya no se necesite. Si el argumento es NIL, el token no debe ser devuelto.

Cada nodo debe mantener las siguientes variables locales:

- *father*: toma valores en {1,...,*N*}, e induce la estructura de árbol del sistema. El valor inicial de esta variable en cada nodo debe ser tal que asegure que la relación $F=\{(i,father_i) \mid i\in\{1,...,N\}\}$ forme un árbol dirigido, donde $father_i$ representa el valor de la variable *father* del nodo *i*.
- *haveToken*: variable booleana que representa la posesión del token. Inicialmente verdadera sólo en el nodo que es raíz del árbol *F*.
- *usingToken*: variable booleana que indica si un nodo se encuentra ejecutando su sección crítica. Inicialmente falsa para todos los nodos.
- *tokenDemanded*: variable booleana que es verdadera mientras el nodo tenga peticiones pendientes por atender, sean éstas remotas o locales. Inicialmente falsa para todos los nodos.
- *q*: cola que almacena las peticiones pendientes que el nodo debe administrar. Almacena valores en {1,...,*N*}. Inicialmente vacía para todos los nodos.

Cada nodo responde a los eventos externos de recibir un mensaje *Request* y de recibir un mensaje *Token*, y a los eventos internos cuando intenta ingresar a la sección crítica y al salir de ella.

La Figura 2 muestra el código que debe ejecutar cada nodo como respuesta a cada uno de los eventos.

PROPIEDADES

En esta sección demostramos propiedades del algoritmo con el objetivo de establecer su correctitud. Algunas demostraciones intermedias sirven, además, para establecer propiedades de evolución y desempeño, en particular el comportamiento del algoritmo en baja y alta carga.

En lo que sigue escribimos a_i para referirnos al valor de la variable local *a* en el nodo *i*. Abreviaremos los nombres de las variables *haveToken*, *tokenDemanded* y *usingToken* como *hT*, *tD* y *uT*, respectivamente.

Los siguientes son lemas preliminares que nos sirven en los teoremas posteriores. Son simples de deducir a partir del código del algoritmo.

Lema 1. $\forall i, tD_i \Leftrightarrow q_i \neq \phi$
Lema 2. $\forall i, \left(hT_i \wedge \neg uT_i\right) \Rightarrow \neg tD_i$
Lema 3. Si existe un mensaje en tránsito con argumento *i*, entonces tD_i
Lema 4. Si *i* quiere entrar a su sección crítica, o se encuentra en ella, entonces $i \in q_i$
Lema 5. Si un mensaje *Token*(*k*) viaja hacia *i*, entonces $q_i \neq \phi$
Lema 6. $i \in q_j \Rightarrow tD_i$

La siguiente definición se utiliza para establecer la propiedad de exclusión mutua.

Definición. Un nodo *i* se dice *privilegiado* si hT_i o existe un mensaje *Token*(*k*) en tránsito hacia *i*. Enfatizaremos la calidad de privilegiado de un nodo *i* escribiendo $\hat{i}$.

Teorema 1 (Exclusión Mutua). *El algoritmo provee exclusión mutua.*

Demostración. Un nodo *i* hace verdadero hT_i cuando recibe un mensaje *Token*. Un nodo *i* hace falso hT_i sólo cuando envía exactamente un mensaje *Token*. Un mensaje *Token* sólo puede ser enviado por un nodo *i* tal que hT_i. Dado que inicialmente existe un único nodo *i* tal que hT_i, o bien existirá un nodo *i* tal que hT_i, o bien exactamente un mensaje *Token* en tránsito. Luego, existe un único nodo privilegiado. Del código se puede observar que sólo un nodo *i* tal que hT_i puede entrar a su sección crítica. Luego, el algoritmo provee exclusión mutua.

Teorema 2 (Bosque de Naimi). *La relación* $F = \{(i, father_i) \mid i \in \{1,\ldots,N\}\}$ *forma un conjunto de árboles dirigidos (bosque), en que las raíces r de cada árbol, y sólo ellas, cumplen con* $tD_r \vee hT_r$.

Demostración. Inicialmente, la propiedad se cumple. Suponga que es cierta en un instante cualquiera. Demostraremos que, luego de ejecutar cualquier paso del algoritmo, la propiedad se mantiene.

– *Caso 1: i* quiere entrar a su sección crítica (Figura 2(a)).

Si $i = father_i$, entonces $hT_i \vee tD_i$. Si tD_i, sólo se ejecuta la primera línea del código. Si $\neg tD_i$, entonces necesariamente hT_i y se ejecutan las líneas 1-6. En ambos casos, no hay cambios en la relación *F* ni en el valor de $hT_i \vee tD_i$.

Si $i \neq father_i$, entonces $\neg hT_i \wedge \neg tD_i$. En este caso, se ejecutan las líneas 1-3 y 7-8. En la línea 3 se hace verdadero tD_i y tras la ejecución de la línea 8 se crea un nuevo árbol con raíz en *i*. La propiedad se mantiene.

(a) i quiere entrar a su SC

```
1  q.ENQUEUE(i)
2  if ¬ tokenDemanded
3     then tokenDemanded ← TRUE
4          if haveToken
5             then usingToken ← TRUE
6                  Permitir la entrada a la SC
7             else Send Request(i) to father
8                  father ← i
```

(b) *i* recibe *Request* (*j*)

```
1  if haveToken ∧¬ usingToken
2     then Send Token(NIL) to j
3          haveToken ← FALSE
4          father ← j
5     else if tokenDemanded
6             then q.ENQUEUE(j)
7             else Enviar Request(j) a father
8                  father ← j
```

(c) *i* recibe *Token(j)*

```
1  haveToken ← TRUE
2  q.ENQUEUE(j)
3  if q.head = i
4     then usingToken ← true
5          Permitir la entrada a la SC
6     else k ← q.DEQUEUE()
7          if q = ∅
8             then Enviar Token(NIL) a k
9                  father ← k
10                 tokenDemanded ← FALSE
11            else Enviar Token(i) a k
12         haveToken ← FALSE
```

(d) *i* sale de su SC

```
1  q.DEQUEUE()
2  usingToken ← FALSE
3  if q = ∅
4     then tokenDemanded ← FALSE
5     else k ← q.DEQUEUE()
6          if q = ∅
7             then Enviar Token(NIL) a k
8                  father ← k
9                  tokenDemanded ← FALSE
10            else Enviar Token(i) a k
11         haveToken ← FALSE
```

Fig. 2 Código que ejecuta *i* como respuesta a cada uno de los eventos del sistema.

– *Caso 2: i* recibe *Request*(*j*) (Figura 2(b)). Note que, por el Lema 3, *j* es raíz de un árbol en *F*.

Si $i = father_i$, entonces $hT_i \vee tD_i$, por lo que se ejecutan las líneas 2-4 o la línea 6, dependiendo del valor de uT_i. En el primer caso, $\neg tD_i$ (Lema 2), se hace falso hT_i, *i* deja de ser raíz y se agrega al árbol con raíz en *j*, manteniendo la propiedad. En el otro caso, no hay cambios en *F* ni en el valor de $hT_i \vee tD_i$, manteniendo también la propiedad.

Si $i \neq father_i$, entonces $\neg hT_i \wedge \neg tD_i$, por lo que se ejecutan las líneas 7 y 8. No cambia el valor de $hT_i \vee tD_i$, *i* se agrega al árbol con raíz en *j*, y la propiedad se mantiene.

– *Caso 3: i* recibe *Token*(*j*) (Figura 2(c)).
Por los Lemas 5 y 1, tD_i. Luego, $i = father_i$. Sólo interesa analizar las líneas 6-12. En la línea 6, se define *k* como la cabeza de q_i. Note que, por el Lema 6, *k* es raíz en *F*. Si se ejecutan las líneas 8-10 y 12, *i* se incluye al árbol con raíz en *k* y hace falso $hT_i \vee tD_i$, por lo que se mantiene la propiedad. Si se ejecutan las líneas 11 y 12, no cambia *F* ni el valor de tD_i, por lo que se mantiene la propiedad.

– *Caso 4: i* sale de su sección crítica (Figura 2(d)).
Necesariamente, hT_i y por los Lemas 4 y 1, tD_i. Luego, $i = father_i$. Sólo interesa analizar las líneas 5-11. Usando la misma deducción que en el Caso 3, se concluye que la propiedad se mantiene.

Corolario 1. *Un mensaje Request*(*i*) *es transmitido en tiempo finito desde su nodo fuente i hasta un nodo j tal que* $j = father_j$, *quien no retransmitirá el mensaje.*

Cada árbol del Bosque de Naimi se comporta como una instancia del algoritmo de Naimi, en la forma en que sus nodos solicitan el ingreso a su sección crítica. El siguiente teorema nos permite establecer la forma en que las peticiones son servidas. Para ello, necesitamos algunas definiciones previas.

En adelante, la expresión $M(a) \mapsto b$ indica que hay un mensaje con argumento *a* viajando hacia *b*, quien consumirá el mensaje. Si el mensaje es *Request*(*a*), entonces *b* es raíz de un árbol del bosque de Naimi. Si el mensaje es *Token*(*a*), entonces $\hat{b}$ es el nodo privilegiado.

Sea $R = \{ r \in \{1,\ldots,N\} \mid tD_r \vee hT_r \}$ el conjunto de las raíces de los árboles del bosque de Naimi. Note que $\hat{p}$, el nodo privilegiado del sistema, se encuentra en *R*. Llamaremos R^* al conjunto de palabras formadas sobre *R*. Asociaremos a cada *i* en $R - \{\hat{p}\}$ una palabra $w_i = x_n x_{n-1} \cdots x_1 \in R^*$, construida con las siguientes reglas: $x_n = \hat{p}$, $x_1 = i$, y $x_k \in q_{x_{k+1}} \vee M(x_k) \mapsto x_{k+1}$, con $1 \leq k < n$.

Las palabras w_i representan una secuencia de nodos en *R*, desde *i* hasta el nodo privilegiado. El siguiente teorema asegura la unicidad de estas palabras, de lo que se deduce que *R* puede ser visto como un árbol, considerando las palabras w_i como el único camino entre la raíz del árbol y cada nodo *i*.

Teorema 3 (Árbol Virtual de Raymond). *Para todo i en* $R - \{\hat{p}\}$, *la palabra* w_i *es única.*

Inicialmente la propiedad se cumple, ya que $R = \{\hat{p}\}$. Suponga que se cumple en un instante cualquiera. Demostraremos que la propiedad se mantiene luego de ejecutar cualquier paso del algoritmo.

– *Caso 1: i* quiere entrar a su sección crítica (Figura 2(a)).

Si $i \in R - \{\hat{p}\}$, necesariamente tD_i. Luego, sólo se ejecuta la línea 1, y no hay cambios en *R*.

Si $i \notin R$, necesariamente $\neg hT_i \wedge \neg tD_i$, y se ejecutan las líneas 1, 3, 7 y 8. Luego de esto, *i* pasa a ser parte de *R* y se hace verdad $Request(i) \mapsto j$, con $j \in R$. Por hipótesis, w_j es única. La palabra w_i se construye como $w_i = w_j i$, que es también única.

Si $i = \hat{p}$, necesariamente $hT_{\hat{i}} \vee tD_{\hat{i}}$. Si $tD_{\hat{i}}$ sólo se ejecuta la línea 1, y no hay cambios en *R*. Si $\neg tD_{\hat{i}}$, entonces se ejecutan las líneas 1-3, 5 y 6, y no hay cambios en *R*.

En cualquiera de los casos anteriormente descritos, la propiedad se mantiene.

– *Caso 2: i* recibe *Request*(*j*) (Figura 2(b)).

Si $i \in R - \{\hat{p}\}$, necesariamente $tD_i \wedge \neg hT_i$, por lo que sólo se ejecuta la línea 6. Las únicas palabras *w* que pueden verse afectadas son aquellas que descansan en el predicado $Request(j) \mapsto i$, que se hace falso. Sin embargo, al finalizar la ejecución, $j \in q_i$, por lo que las palabras no cambian.

Si $i \notin R$, necesariamente $\neg hT_i \wedge \neg tD_i$, y se ejecutan las líneas 7 y 8. No se agregan nodos a *R*, y lo único que hace *i* es redirigir el mensaje *Request*(*j*) en dirección a un nodo $k \in R$.

Si $i = \hat{p}$, necesariamente $hT_{\hat{i}} \vee tD_{\hat{i}}$. Si $\neg hT_{\hat{i}}$, necesariamente $tD_{\hat{i}}$, y sólo se ejecuta la línea 6. Las únicas palabras *w* que pueden verse afectadas son aquellas que descansan en el predicado $Request(j) \mapsto \hat{i}$, que se hace falso. Sin embargo, al finalizar la ejecución, $j \in q_{\hat{i}}$, por lo que las palabras no cambian. Ocurre lo mismo si $hT_{\hat{i}} \wedge uT_{\hat{i}}$.

Si $i = \hat{p}$ y $hT_i \wedge \neg uT_i$, se ejecutan las líneas 2-4. El nodo *i* deja de ser privilegiado, $\hat{j}$ se convierte en nodo privilegiado (ya que *Token*(NIL) se envía a $\hat{j}$) y se crea la arista $i \to \hat{j}$. Analizaremos cómo afecta esto a las palabras *w*. En lo que sigue, $\alpha \in R^*$ y w_m^- se refiere a la única palabra asociada a *m* antes de ejecutar el código. Si para algún *m*, $j \in w_m^-$ necesariamente $w_m^- = \hat{i}j\alpha$, ya que $Request(j) \mapsto i$. Note que $j \notin \alpha$, por la unicidad de w_m^-. Si $w_m^- = \hat{i}j\alpha$, es claro que $w_m = \hat{j}\alpha$, ya que $\hat{j}$ es el nuevo nodo privilegiado. La unicidad se mantiene. Si $w_m^- = \hat{i}l\alpha$, con $l \neq j$, $w_m = \hat{j}l\alpha$, ya que, al crearse la arista $i \to \hat{j}$, un potencial mensaje dirigido hacia *i* ahora está dirigido hacia $\hat{j}$. La unicidad se mantiene en este último caso, debido a que $j \notin \alpha$.

– *Caso 3*: $\hat{i}$ recibe *Token*(*j*) (Figura 2(c)).

Si *j* no es NIL, aquellas palabras que descansan en el predicado $Token(j) \mapsto \hat{i}$ no cambian, ya que, a pesar de que el predicado se hace falso, luego de la ejecución de las líneas 1 y 2, $j \in q_{\hat{i}}$.

Si $\hat{i}$ se encuentra en la cabeza de $q_{\hat{i}}$, se ejecutan las líneas 4 y 5. *R* no cambia, y las palabras *w* tampoco.

Si $k \neq \hat{i}$ se encuentra en la cabeza de $q_{\hat{i}}$ y se ejecutan las líneas 6, 8-10 y 12, el nodo *i* deja de ser privilegiado, $\hat{k}$ se convierte en nodo privilegiado (ya que *Token*(NIL) se envía a $\hat{k}$) y se crea la arista $i \to \hat{k}$. Analizaremos cómo afecta esto a las palabras *w*. Si para algún *m*, $k \in w_m^-$ necesariamente $w_m^- = \hat{i}k\alpha$, ya que $k \in q_{\hat{i}}$. Note que $k \notin \alpha$, por la unicidad de w_m^-. Si $w_m^- = \hat{i}k\alpha$, es claro que $w_m = \hat{k}\alpha$, ya que $\hat{k}$ es el nuevo nodo privilegiado. La unicidad se mantiene. Si $w_m^- = \hat{i}l\alpha$, con $l \neq k$, $w_m = \hat{k}l\alpha$, ya que, al crearse la arista $i \to \hat{k}$, un potencial mensaje dirigido hacia *i* ahora está dirigido hacia $\hat{k}$. La unicidad se mantiene en este último caso, debido a que $k \notin \alpha$.

Si $k \neq \hat{i}$ se encuentra en la cabeza de $q_{\hat{i}}$ y se ejecutan las líneas 6, 11 y 12, el nodo *i* deja de ser privilegiado y $\hat{k}$ se convierte en nodo privilegiado, ya que *Token*(*i*) se envía a $\hat{k}$. Analizaremos cómo afecta esto a las palabras *w*. Si $w_m^- = \hat{i}k\alpha$, $w_m = \hat{k}\alpha$ y es única. Si $w_m^- = \hat{i}l\alpha$, con $l \neq k$, $w_m = \hat{k}il\alpha$, ya que *i* permanece en *R* y $\hat{k}$ es el nuevo nodo privilegiado. Dado que $k \notin \alpha$, w_m es única.

– *Caso 4:* $\hat{i}$ sale de su sección crítica (Figura 2(d)).

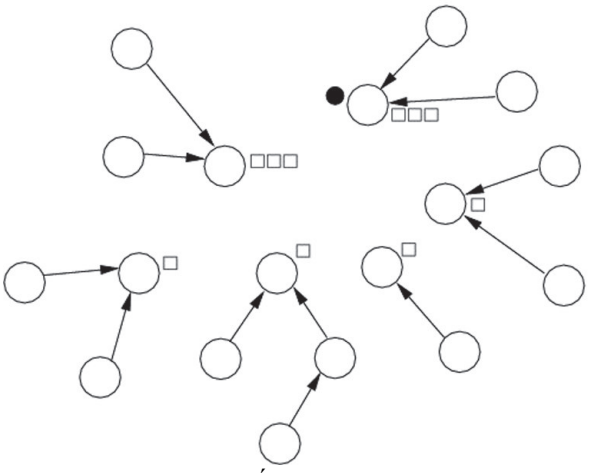
Si $|q_{\hat{i}}| = 1$, se ejecutan las líneas 1-4. *R* no cambia, $\hat{i}$ sigue siendo el nodo privilegiado, y no cambian las palabras *w*.

Si $|q_{\hat{i}}| > 1$, sólo interesa analizar las líneas 1, 2, 5-11. Del mismo argumento del Caso 3, se concluye la unicidad de las palabras *w*.

Todos los nodos que pertenecen al Árbol Virtual de Raymond rigen su comportamiento por las reglas del algoritmo de Raymond, en cuanto al servicio de las peticiones por ingreso a sección crítica.

La Figura 3 muestra un esquema de las estructuras mencionadas en los dos teoremas anteriores para un punto particular de una ejecución del algoritmo. Note que el token siempre viaja por las aristas del árbol virtual de Raymond. Esta observación, más el Corolario del Teorema del Bosque de Naimi, nos permite demostrar el siguiente teorema de justicia.

(a) Estructuras efectivas del algoritmo.



(b) Bosque de Naimi y Árbol Virtual de Raymond

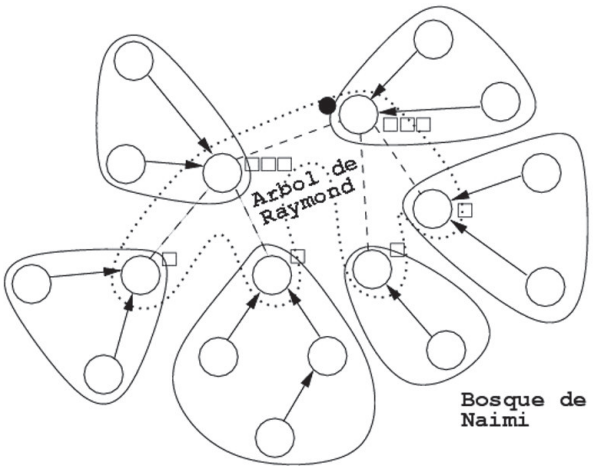


Fig. 3 Ejemplo de las estructuras para un punto en la ejecución del algoritmo

Teorema 4. *Cualquier petición por ingreso a una sección crítica es atendida en un tiempo finito.*

La demostración de este teorema se basa en el Corolario del Teorema del Bosque de Naimi, que asegura que una petición llegará en tiempo finito a un nodo del árbol virtual de Raymond. Las colas locales de cada uno de los nodos del árbol virtual de Raymond definen el recorrido que hará el token a través de este árbol, asegurando el servicio de cada una de las peticiones.

La correctitud del algoritmo se deduce de este último teorema unido al de exclusión mutua.

Note que si hay sólo una petición pendiente, el Árbol Virtual de Raymond está formado por sólo un nodo y existe un único árbol en el Bosque de Naimi compuesto por todos los nodos del sistema. Si los N nodos del sistema quieren ingresar a su sección crítica, existen N árboles independientes en el Bosque de Naimi. Bajo estas condiciones, todos los nodos pertenecen al Árbol Virtual de Raymond.

DESEMPEÑO

Estudiamos el desempeño de nuestro algoritmo comparándolo con los de Raymond y Naimi. Estos se citan en la literatura como los que alcanzan un mejor desempeño en cuanto al tráfico de mensajes de entre los algoritmos no centralizados basados en token y que intercambian mensajes de tamaño constante. Las métricas utilizadas son tráfico de mensajes y tiempo de espera. Es difícil desarrollar un estudio analítico de estas métricas ya que la cardinalidad del espacio de estados crece muy rápidamente al crecer el número de nodos en el sistema [3]. Por esto, utilizamos técnicas de simulación para medir el desempeño.

El modelo de simulación es similar al usado por Chang [3] y al usado por Johnson [4] en sus estudios de desempeño de algoritmos de exclusión mutua distribuida. Asumimos que en cada nodo, las peticiones por ingreso a la sección crítica se rigen por un proceso de Poisson de parámetro λ. Este parámetro entrega una noción del nivel de carga del sistema. El tiempo que tarda un nodo en ejecutar su sección crítica es modelado como una constante C. El tiempo de propagación de un mensaje cualquiera por la red es una constante T multiplicada por un número aleatorio con distribución uniforme entre 0 y 1.

Las dos métricas son definidas formalmente como:

- Número de mensajes por ingreso: número total de mensajes intercambiados dividido por el total de secciones críticas completadas por los nodos en el sistema, es decir, el promedio de los mensajes intercambiados cuando un nodo quiere entrar a su sección crítica.

- Tiempo de espera por ingreso: el promedio del tiempo total que pasa un nodo esperando el acceso a su sección crítica, es decir, el tiempo promedio entre que la petición es realizada y el ingreso se hace efectivo.

Para obtener resultados estadísticamente confiables, realizamos simulaciones *a largo plazo* ejecutando, en cada experimento, 100.000 ingresos a sección crítica. El parámetro T fue tomado como 0.1 y el parámetro C como 0.01. Estos valores son consistentes con los utilizados en los estudios anteriormente mencionados.

En los primeros experimentos consideramos un número de nodos fijo $N = 31$ y tomamos mediciones para distintos valores de λ en el intervalo [0, 1]. La Figura 4 muestra los resultados obtenidos al medir el número de mensajes para distintas topologías iniciales. Nuestro algoritmo está etiquetado como NxR. En cada caso, el algoritmo propuesto alcanza un desempeño similar al de Naimi en baja carga, y similar al de Raymond hacia alta carga. En los casos 4(a)–(c) nuestro algoritmo es el que presenta el mejor desempeño para todos los niveles de carga. En el caso de la Figura 4(d) el algoritmo de Raymond es el que tiene mejor desempeño. Sin embargo, dado que la topología es de estrella, este algoritmo se comporta exactamente como un algoritmo centralizado en donde el nodo central de la estrella es el coordinador. Por lo tanto, la comparación no es justa ya que, a pesar de la topología inicial, los otros dos algoritmos siguen siendo distribuidos. Los resultados nos muestran que en los casos distribuidos, nuestro algoritmo es el de mejor desempeño con respecto al número de mensajes intercambiado por ingreso a sección crítica.

En la Figura 5 se muestran los resultados para el número de mensajes en alta y baja carga para distinto número de nodos, considerando siempre un árbol binario como topología inicial. Al variar el número de nodos, nuestro algoritmo sigue manteniendo la propiedad de comportarse como el algoritmo de Naimi en baja carga, y como el algoritmo de Raymond en alta carga.

La Figura 6 muestra comparaciones para el tiempo de espera en distintos experimentos. En los casos (a) y (b), se compara el tiempo de espera para dos topologías iniciales con un número $N = 31$ de nodos. En ambos casos, el algoritmo de Naimi presenta un mejor comportamiento al compararlo con el algoritmo de Raymond, tanto en baja como en alta carga. Nuestro algoritmo entonces es mejor que el algoritmo de Raymond en cuanto al tiempo de espera, pero se comporta peor que el algoritmo de Naimi en alta carga.

Las Figuras 6(c) y (d) muestran comparaciones para distinto número de nodos en baja y alta carga. Estos gráficos confirman que, también para esta métrica, nuestro algoritmo se comporta como el algoritmo de Naimi en baja carga y como el de Raymond en alta carga.

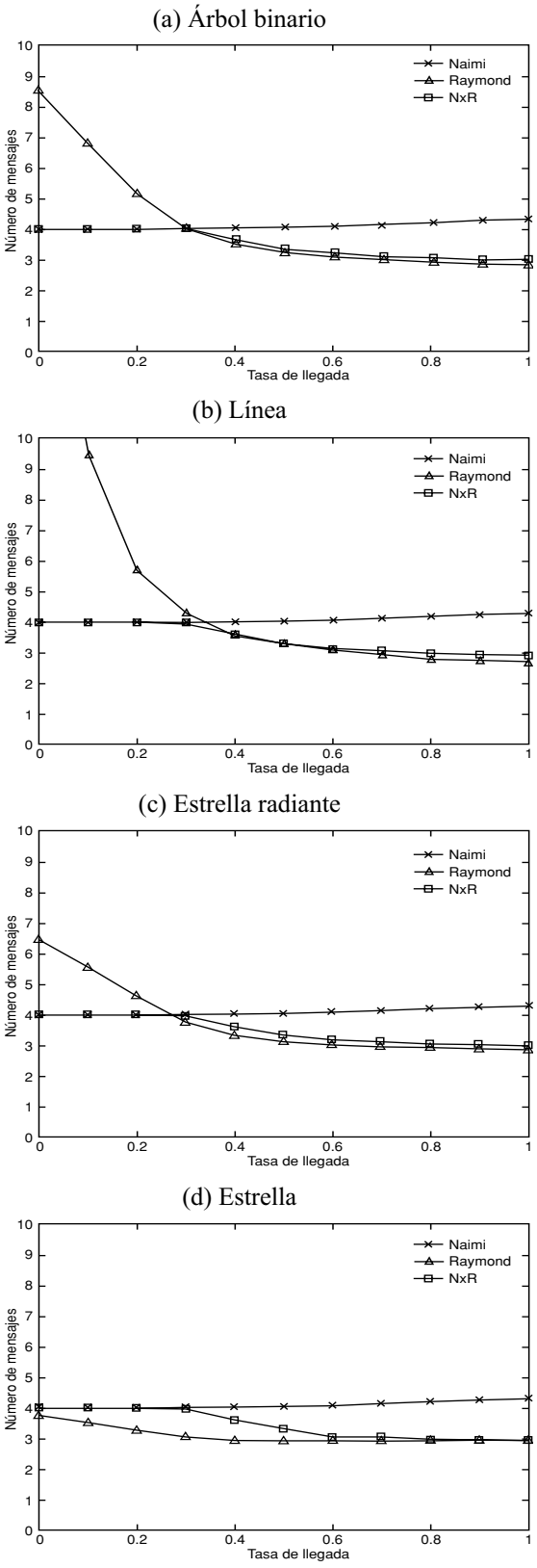


Fig. 4 Cantidad de mensajes intercambiados para distintas topologías iniciales.

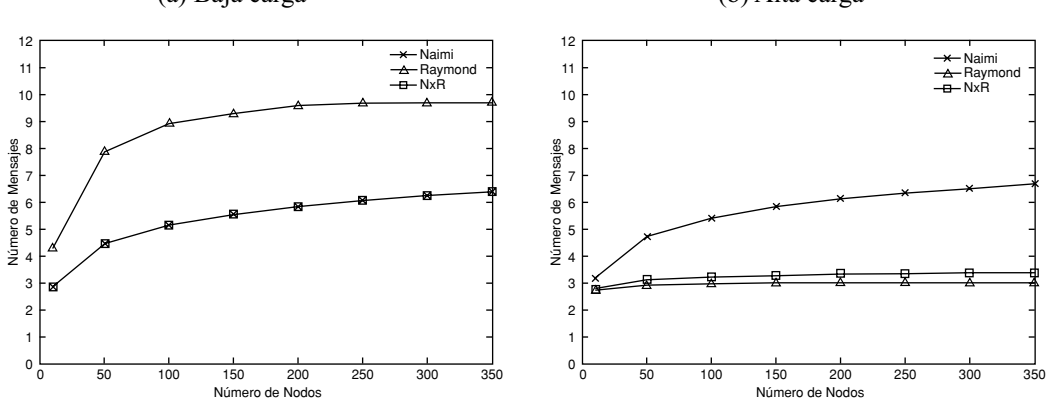


Fig. 5 Cantidad de mensajes intercambiados en alta y baja carga, para distinto número de nodos.

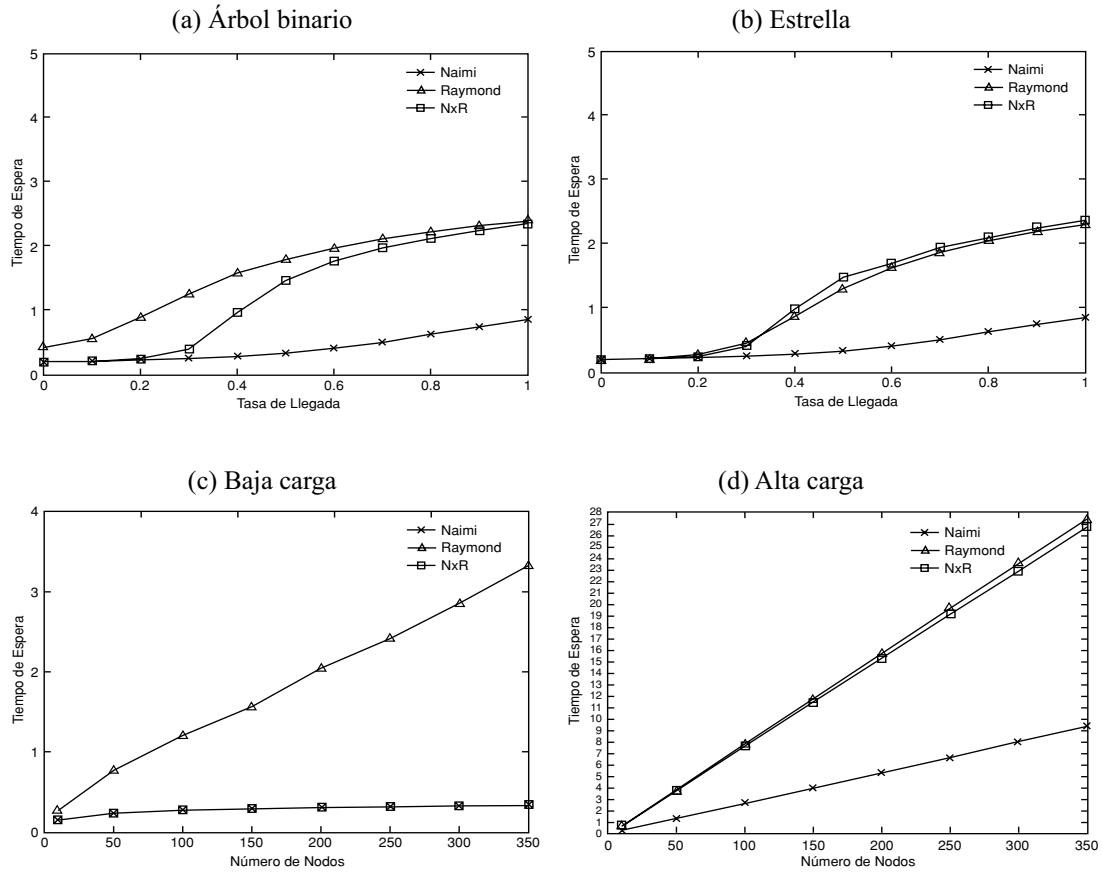


Fig. 6 Comparaciones para el tiempo de espera.

Los resultados empíricos muestran que, al considerar sólo el número de mensajes intercambiados, nuestro algoritmo es el que presenta el mejor desempeño. Sin embargo, al considerar el tiempo de espera, el algoritmo de Naimi presenta mejor desempeño, pero sólo en condiciones de alta carga.

CONCLUSIONES

En este artículo presentamos un nuevo algoritmo de exclusión mutua distribuida basado en paso de token. Nuestro algoritmo combina las mejores características de dos algoritmos anteriores propuestos por Raymond [8] y Naimi [6]. Estos son citados como los más eficientes de entre los no centralizados de paso de token que intercambian mensajes de tamaño constante. Nuestro algoritmo dirige las peticiones como el algoritmo de Naimi, a través de la estructura que llamamos *Bosque de Naimi*, y las sirve como el algoritmo de Raymond, a través de la estructura que llamamos *Árbol Virtual de Raymond*.

Presentamos un estudio de desempeño basado en técnicas de simulación. Los resultados empíricos muestran que nuestro algoritmo es el más eficiente en cuanto al número de mensajes intercambiados, al compararlo con los algoritmos de Raymond y Naimi.

REFERENCIAS

[1] D. Agrawal and A. El Abbadi. "Efficient solution to the distributed mutual exclusion problem". In Proceedings of the eighth annual ACM Symposium on Principles of distributed computing, pages 193 – 200, Edmonton, Alberta, Canada, August 1989. ACM Press.

[2] S. Banerjee and P.K. Chrysanthis. "A New Token Passing Distributed Mutual Exclusion Algorithm". In Proc. of the 16th. International Conference on Distributed Computing Systems (ICDCS 96), pages 717-725. IEEE Computer Society, May 1996.

[3] Ye-In Chang. "A Simulation Study on Distributed Mutual Exclusion". Journal of Parallel and Distributed Computing, 33(2):107-121, 15 March 1996.

[4] Theodore Johnson. "A Performance Comparison of Fast Distributed Mutual Exclusion Algorithms". In Proceedings of the 9th International Symposium on Parallel Processing (IPPS'95), pages 258-264, Los Alamitos, CA, USA, April 1995. IEEE Computer Society Press.

[5] Mamoru Maekawa. "A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems". ACM Transactions on Computer Systems, 3(2):145-159, May 1985.

[6] Mohamed Naimi, Michel Trehel and Andre Arnold. "A log(N) Distributed Mutual Exclusion Algorithm based on Path Reversal". Journal of Parallel and Distributed Computing, 34(1):1-13, April 1996.

[7] M.L. Neilsen and M. Mizuno. "A DAG-Based Algorithm for Distributed Mutual Exclusion". In Proc. of the 11th. International Conference on Distributed Computing Systems (ICDCS 91), pages 354-360, Arlington, Texas, USA, May 1991. IEEE Computer Society.

[8] Kerry Raymond. "A Tree-Based Algorithm for Distributed Mutual Exclusion". ACM Transactions on Computer Systems, 7(1):61-77, February 1989.

[9] Michel Raynal. "A Simple Taxonomy for Distributed Mutual Exclusion Algorithms". ACM SIGOPS Operating Systems Review, 25(2):47-50, April 1991.

[10] Glenn Ricart and Ashok K. Agrawala. "An Optimal Algorithm for Mutual Exclusion in Computer Networks". Communications of the ACM, 24(1): 9-17, January 1981.

[11] Michel Trehel and Mohamed Naimi. "An Improvement of the log(n) Distributed Algorithm for Mutual Exclusion". In 7th International Conference on Distributed Computing Systems (ICDCS '87), pages 371-377, Washington, D.C., USA, September 1987. IEEE Computer Society Press.

[12] Min-You Wu and Wei Shu. "An Efficient Distributed Token-Based Mutual Exclusion Algorithm with Central Coordinator". Journal of Parallel and Distributed Computing, 62(10):1602-1613, October 2002.