UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"Modelos de comunicación de alto nivel con detección de fallas"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS

(COMPUTACIÓN)

P R E S E N T A:

ALEJANDRO CORNEJO COLLADO

DIRECTOR DE TESIS: Dr. Sergio Rajsbaum

México, D.F. 2007.

A mis padres,

por su cariño,

y a mi novia,

por su paciencia.

Agradezco a mi asesor Sergio Rajsbaum por su orientación y dedicación, sin la cual no existiría este trabajo. También agradezco a los profesores con quien tuve el gusto de tener clases en el posgrado, en especial a los sinodales, por su tiempo y sus consejos.

Índice general

Lista de Algoritmos

RESUMEN

Este trabajo estudia el modelo iterado de copia inmediata como medio de comunicación en sistemas asíncronos enriquecidos con detectores de fallas. Un sistema asíncrono es aquel donde no existen cotas sobre la velocidad del medio de comunicación o sobre la velocidad relativa de procesamiento de los elementos del sistema. Un detector de fallas es un oráculo local que puede ser consultado por cualquier proceso para obtener información sobre las fallas en el sistema. El tipo y la calidad de esta información depende del tipo de detector de fallas que se utilize.

El objetivo de este trabajo es proveer de un modelo de comunicación de alto nivel, como lo es el modelo iterado de copia inmediata, que no este sujeto a los resultados de imposibilidad asociados con los sistemas asíncronos. Es precisamente por esto que se trabaja en sistemas enriquecidos con detectores de fallas.

En esta dirección, este trabajo hace las siguientes contribuciones. Introducimos formalmente conceptos nuevos en la definición de fallas, las fallas implicitas y las fallas de equivalencia. Demuestra que el modelo iterado de copia inmediata es estrictamente más debil para tareas generales que el modelo de memoria compartida. Probamos que no es posible traducir literalmente la definición de tareas sin-espera a sistemas con detección de fallas. Finalmente se presenta una simulación que, bajo ciertas suposiciones, permite utilizar el modelo iterado de copia inmediata en sistemas asíncronos enriquecidos con detectores de fallas a tareas arbitrarias, sin pérdida de generalidad con respecto al modelo de memoria compartida tradicional.

Palabras clave: Teoría de sistemas distribuidos, detección de fallas, copia inmediata

Capítulo 1

Introducción

El advenimiento de la Internet y la disminución en el uso de las súper computadoras tradicionales a favor de la utilización de arreglos de computadoras de escritorio, son solo un par de ejemplos donde se subraya la importancia de los algoritmos de cómputo distribuido que sean eficientes y tolerantes a las fallas. En el proceso de desarrollo y análisis de un algoritmo distribuido es necesario considerar algún modelo de comunicación.

En los modelos síncronos de comunicación existe una cota superior conocida para el tiempo que le toma a un mensaje ser enviado de un proceso a otro, a esta cota se le llama δ. También existe una cota superior conocida entre las velocidades relativas de los procesos, y se le llama ρ. Sin embargo, este modelo no se asemeja a las redes reales, ya que estas cotas no siempre existen y pocas veces son conocidas. Por otro lado se ha definido el modelo asíncrono en el cual las cotas superiores δ y ρ sobre el medio de comunicación y las velocidades relativas de los procesos, no existen.

En medio de estos dos modelos se encuentran los modelos de sincronía parcial [1]. A grandes rasgos estos modelos de sincronía parcial se pueden dividir en dos grupos, aquellos donde las cotas δ y ρ existen pero no son conocidas, y aquellos donde las cotas

son conocidas pero solo se cumplen eventualmente[1].

Muchos de los problemas reales que se buscan resolver en sistemas distribuidos necesitan de algún tipo de cooperación entre los procesos. A un nivel más bajo, el concepto de ponerse de acuerdo en algún valor (o cualquier otra información) es clave para lograr dicha cooperación. Por ejemplo, en una votación electrónica distribuida existen varios candidatos posibles y se busca ponerse de acuerdo para que todos los participantes elijan a un único ganador. Aunque el concepto de ponerse de acuerdo resulta muy natural, es un problema sorprendentemente difícil de resolver.

Precisamente debido a la función fundamental que tiene la cooperación entre procesos en sistemas distribuidos, frecuentemente se escogen problemas de "acuerdo" para estudiar los distintos modelos de comunicación. Uno de estos problemas es conocido como el problema del consenso y se describe informalmente a continuación.

Definición 1. *En el consenso participan n procesos, cada uno comienza con un valor de entrada privado tomado de algún dominio V de valores.*

En cualquier solución del consenso todos los procesos correctos escogen un mismo valor de salida. Más aun, ese valor de salida fue entrada de al menos un proceso.

Más formalmente, decimos que un protocolo resuelve el problema del consenso si satisface las tres siguientes condiciones:

- Terminación: Todo procesos correcto eventualmente escoge un valor.
- Acuerdo: Todos los procesos correctos deciden el mismo valor.
- Validez: Todo valor escogido por un proceso correcto fue entrada de al menos un proceso.

[1] Aquí y en el resto del documento la palabra eventual se utiliza en el mismo sentido que en inglés. Es decir una propiedad se cumple eventualmente si existe un tiempo t a partir del cual la propiedad se cumpla.

La propiedad de validez es necesaria para descartar algoritmos triviales que devuelven un valor que no fue propuesto por nadie, o que siempre devuelven el mismo valor. El problema del consenso tiene solución tanto en el modelo síncrono como en el asíncrono, siempre y cuando el medio de comunicación sea confiable y todos los procesos sean correctos. Decimos que un medio de comunicación es confiable si todos los mensajes enviados entre procesos correctos son eventualmente recibidos.

Sin embargo, como veremos en las siguientes secciones, incluso si el medio de comunicación es confiable no es posible tolerar ni una falla en un modelo totalmente asíncrono. La consecuencia obvia de esto es que cuando se tratan con modelos asíncronos, muchos de los problemas de acuerdo no tienen una solución tolerante a fallas.

1.1. Trabajos previos

En 1983 Fisher, Lynch y Paterson [2] probaron que en el modelo asíncrono usando memoria compartida (los resultados se pueden extender al modelo de paso de mensajes) el problema del consenso no tiene solución aún si un solo proceso tiene una falla de tipo paro. A partir de este momento hubo un interés en identificar problemas que son solubles en el caso de que ocurran k fallas, pero no si ocurren $k + 1$ fallas. El problema de *acuerdo de k-conjunto* (*k-set agreement*) fue propuesto por Chaudhuri en 1990 [3] como un problema que cumplía con estas características.

Definición 2. *En el problema de acuerdo de k-conjunto participan n procesos, cada proceso comienza con un valor de entrada privado tomado de algún dominio V de valores.*

En cualquier solución de este problema cada proceso correcto escoge un valor de salida, tal que haya a lo más k valores de salida distintos. Más aun, todo valor decidido

fue entrada de al menos un proceso.

Este problema es la generalización natural al problema del consenso, en particular el consenso es el caso de $k = 1$, es decir cuando el conjunto de valores decididos contiene un solo elemento. En [3] se probó que existe un algoritmo capaz de resolver el problema del acuerdo de k-conjunto tolerando $k - 1$ fallas, y se conjeturó que no existe ningún algoritmo capaz de resolver el problema y que tolere k fallas.

No fue sino hasta 1993 cuando tres distintos grupos, el de Saks y Zaharoglou [4], el de Herlihy y Shavit [5] y el de Borowsky y Gafni [6] probaron que la conjetura de Chaudhuri era correcta. En [6] se introdujo el modelo de *copia inmediata* y se utilizó una variante del lema de Sperner para derivar los resultados, lo cual refleja la relación entre la solubilidad de problemas en sistemas distribuidos y el campo de la topología matemática. La copia inmediata es un objeto de comunicación que provee de una sola operación ISNAPSHOT que al ser invocada por un proceso p_i escribe un valor a la celda de memoria asignada a p_i y devuelve una copia del estado de la celdas de memoria de todos los demás procesos (incluyendo p_i) inmediatamente después de la escritura.

En esta misma línea de investigación fue que en 1997 Borowsky y Gafni [7] introdujeron la noción de un *modelo iterado*. Si un modelo utiliza un objeto compartido de comunicación X, en la versión iterada del modelo se utiliza un arreglo infinito de objetos X de *uso-único* (*one-shot*) para la comunicación. Un objeto es de uso-único cuando las operaciones solo pueden ser invocadas una vez por cada proceso. En particular en [7] se describió el modelo *iterado de copia inmediata* (es decir donde la comunicación se hace mediante objetos de copia inmediata de uso-único) y demostraron que este modelo induce una estructura topológica recursiva.

Por otro lado, en 1996 Chandra y Toueg [8] introdujeron el concepto de detectores

de fallas no confiables. Informalmente se puede describir a un detector de fallas como un oráculo local que provee de información sobre las fallas a cada proceso. El tipo y la calidad de la información devuelta depende de las características del detector de fallas en particular. Un ejemplo concreto es el detector de fallas Ω, este detector devuelve la identidad de un proceso que cumple con la propiedad de liderazgo eventual.

- Liderazgo eventual: Existe un tiempo t a partir del cual Ω devuelve el mismo identificador a todos los procesos. Más aun, este identificador corresponde a un proceso correcto.

Chandra, Hadzilacos y Toueg [9] probaron que es posible resolver el consenso en un sistema asíncrono enriquecido por un detector de fallas de la clase Ω. En [9] también se probó que Ω es el detector de fallas más *débil* capaz de resolver el consenso. Informalmente se dice que un detector de fallas $\mathcal{X}$ es más débil que otro detector de fallas $\mathcal{Y}$ si es posible construir a $\mathcal{X}$ usando $\mathcal{Y}$. Por lo tanto, las restricciones impuestas por Ω ayudan a caracterizar el nivel de sincronía mínimo necesario para poder resolver el consenso en un sistema que de otro modo sería asíncrono.

1.2. Objetivos y contribuciones

En este trabajo se combinan los enfoques del modelo iterado de copia inmediata con la abstracción de detectores de fallas. El objetivo de combinar estos enfoques es obtener los beneficios de la utilización de objetos de comunicación de alto nivel (como lo es el objeto de copia inmediata) y esquivar los resultados de imposibilidad de los modelos asíncronos al utilizar detectores de fallas no confiables.

Un resumen de los resultados preliminares de esta investigación se introdujeron en [10], que se publicará este mismo año en las memorias de la conferencia de principios

de cómputo distribuido (Principles of Distributed Computing) en Óregon, EUA. En esta publicación introducimos los conceptos básicos presentados en esta tesis y exploramos hasta que punto el modelo iterado de copia inmediata puede ser modificado para ser útil en el análisis de sistemas asíncronos enriquecidos con detectores de fallas no confiables.

- Se presentan nuevos conceptos en la definición de fallas y se introducen formalmente las fallas implícitas y las fallas de equivalencia.
- Demostramos la imposibilidad de generalizar la simulación [7] para protocolos que no sean *sin-espera* (*wait-free*).
- Se analiza el modelo iterado de copia inmediata enriquecido con detectores de fallas, y en esta dirección probamos que la definición de protocolos sin-espera no se puede aplicar en su forma tradicional a sistemas con detectores de fallas.
- Se prueba que la simulación de [7] no es suficiente para tener equivalencia entre modelos cuando ambos están enriquecidos con detectores de fallas.
- Finalmente se analizan los modelos de registros atómicos y el modelo iterado de copia inmediata utilizando la definición de fallas de equivalencia y enriquecidos con detección de fallas. Se introduce el algoritmo de *convergencia vectorial con cooperación* y se prueba la equivalencia de ambos modelos. Esta equivalencia es más general que la presentada en [7], ya que no exige que los algoritmos sean sin-espera.

1.3. Justificación

Los objetos de comunicación, como el de copia inmediata, ayudan a simplificar las tareas de análisis y diseño de algoritmos distribuidos. Esto se debe a que estos objetos

tienen propiedades precisas que ayudan a encapsular la complejidad del diseño de un algoritmo. Por lo tanto, gran parte de la complejidad del diseño se transfiere al objeto de comunicación, y por otro lado el análisis del algoritmo se facilita ya que se pueden suponer las propiedades del objeto (debido a que fueron probadas previamente).

Es en éste espíritu que se estudiaron objetos de comunicación como *copia atómica* [11], *copia inmediata* [12] y abstracciones como *alpha*[13] o *lambda*[14]. En la primera parte de esta tesis explicaremos brevemente algunos de estos objetos, y nos enfocaremos en un modelo formal presentado en [6] que utiliza solamente objetos de uso-único de copia inmediata para la comunicación.

En 1997 Borowsky y Gafni [12] estudiaron la relación entre el modelo de memoria compartida con registros atómicos y el modelo iterado de copia inmediata. En su investigación probaron que estos dos modelos son equivalentes para protocolos terminables (donde todos los procesos terminan la ejecución) con algoritmos sin-espera. Sin embargo, como ya mencionamos, en 1983 Fisher, Lynch y Paterson [2] ya habían probado que en el modelo asíncrono usando memoria compartida existen problemas (en particular el problema del consenso) que no tienen solución, aún si un solo proceso tiene una falla de tipo paro.

Por lo tanto, aunque el modelo iterado de copia inmediata provee de un objeto de alto nivel con propiedades deseables desde un punto de vista algorítmico (ademas de prestarse para análisis usando herramientas de topología matemática) no es suficientemente poderoso para resolver los problemas clave de sistemas distribuidos. Para eliminar esta limitación inherente a los sistemas asíncronos se ha estudiado enriquecer los sistemas con detectores de fallas no confiables [8]. En este trabajo nos proponemos extender este análisis para cubrir al modelo iterado de copia inmediata enriquecido con detectores de fallas.

De este modo será posible utilizar el modelo iterado de copia inmediata para desarrollar algoritmos, y al trabajar en sistemas enriquecidos con detectores de fallas es posible implementar algoritmos que requieran cierto nivel de sincronización, como el problema del consenso o el de acuerdo de k-conjunto y encontrar soluciones tolerantes a fallas.

1.4. Organización

En la primera parte de este documento presentaremos los modelos de comunicación más importantes en el área de sistemas distribuidos, en particular nos enfocaremos en los modelos de memoria compartida. Describiremos las relaciones entre ellos, así como sus limitaciones, se exploraran en detalle los sistemas asíncronos ya que por ser los que tienen menos restricciones también son los que presentan más complicaciones.

Después continuaremos con la descripción de los distintos tipos de fallas que pueden ocurrir en los modelos de comunicación. Al final del capítulo dos se describirá la abstracción de detectores de fallas y su función como un puente entre los sistemas asíncronos y los síncronos. En el capítulo tres describiremos la relación que existe entre el modelo iterado de copia inmediata y el modelo de registros atómicos. En particular el esquema presentado en [7] para simular el segundo en el primero, lo que demuestra que ambos son equivalentes para protocolos sin espera.

Luego continuaremos el análisis extendiendo ambos modelos con módulos de detección de fallas. Se presentaran varios resultados de imposibilidad y al final del capítulo tres se estudian ambos modelos bajo la definición de fallas de equivalencia, donde probaremos que estos dos modelos son equivalentes. Finalmente en el último capitulo se presentarán las conclusiones y algunas líneas para trabajos futuros.

Capítulo 2

Modelos

En general modelaremos un sistema distribuido como un conjunto Π de n procesos. Existen dos grandes categorías donde caen muchos de los modelos de comunicación distribuida, el modelo de paso de mensajes y el de memoria compartida, en este trabajo nos enfocaremos en el segundo.

Una vez dentro del contexto de un medio de comunicación, tanto los procesos como el medio pueden ser de carácter síncrono o asíncrono. En los sistemas asíncronos no se hace ninguna suposición sobre las velocidades relativas de los procesos o sobre el medio de comunicación, y por lo tanto tienen más aplicaciones prácticas, este documento estudiara protocolos dentro de este modelo. Por otro lado el sistema síncrono hace ciertas suposiciones para garantizar cotas inferiores y superiores en el medio de comunicación.

En el resto de este capítulo se presentaran con detalle las definiciones de los modelos de comunicación y se exploraran algunas de las relaciones entre ellos. Para facilitar tanto el análisis como el diseño de protocolos en sistemas distribuidos, es conveniente utilizar primitivos de comunicación de alto nivel. Sin embargo, se busca que dichos primitivos de comunicación se puedan construir en sistemas de cómputo reales.

En las siguientes secciones comenzaremos por describir las primitivos de comuni-

cación más sencillos, es decir registros seguros que solo permiten una escritura y una lectura a la vez, y a partir de ellos construiremos objetos de más alto nivel con distintas propiedades. En la última parte de este capitulo presentaremos la definición de distintos tipos de fallas. De particular interés son las fallas implícitas y las fallas de equivalencia, ya que estas definiciones son las que servirán para probar los resultados del capítulo tres. También introduciremos la abstracción de los detectores de fallas y sus propiedades.

2.1. Sistemas síncronos y asíncronos

Decimos que un sistema de cómputo distribuido es síncrono si se cumplen las dos siguientes propiedades:

- Existe una cota superior δ en el medio de comunicación (paso de mensajes o memoria compartida).
- Si $C_i(t)$ es la lectura del reloj local del proceso i en el tiempo t, entonces para toda $t \geq t'$ y todo proceso i se cumple que:

$$(1+\rho)^{-1} \leq \frac{C_i(t) - C_i(t')}{t - t'} \leq (1+\rho)$$

  Donde $\rho \geq 0$ encapsula las cotas inferiores y superiores de error en los relojes locales.

Un sistema asíncrono es lo opuesto, no existe una cota entre las velocidades relativas de los procesos o del medio de comunicación. En el tiempo que le toma a un proceso ejecutar un paso, otro proceso puede ejecutar un número no acotado pero finito de pasos. Además, la velocidad relativa entre dos procesos puede ser variable, es decir que mientras en algunas etapas un proceso p_i puede ser más rápido que otro proceso p_j, en otra etapa de la ejecución puede pasar lo contrario.

Debido a la falta de suposiciones los protocolos diseñados para funcionar en un sistema asíncrono se pueden aplicar directamente a entornos de computo reales (en contraste con los sistemas síncronos).

2.2. Memoria compartida (registros)

Consideremos un registro compartido capaz de guardar un número finito de valores distintos (es decir es un registro discreto). En vez de mandar o recibir mensajes un proceso realiza las operaciones de escritura y lectura sobre el registro, donde el contenido del mensaje es equivalente al valor que que se guarda o lee [15]. Para simplificar más este registro supondremos que escribe un proceso a la vez, y lee un proceso a la vez, llamaremos a este registro *SWSR*[1].

Incluso en este tipo más simple de registro hay lugar para inconsistencias, ya que es posible que ocurra una escritura y una lectura concurrentemente, y para este caso [15] describe tres tipos de comportamientos:

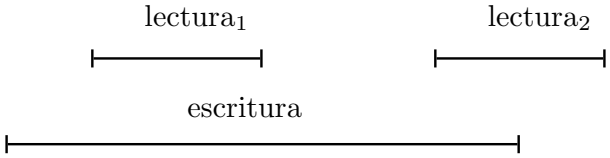
- Seguro: Un registro seguro es aquel que en el caso de una lectura y una escritura concurrente solamente garantiza que la lectura devuelve un valor dentro del rango valido que puede guardar el registro.

- Regular: En caso de una lectura concurrente con una escritura un registro regular devuelve siempre el valor que estaba antes de la escritura ó el valor que se pretende escribir.

- Atómico: En un registro atómico siempre devuelve un valor leído de manera que siempre sea posible ordenar (linearizar) las operaciones de escritura y lectura.

[1] Single-Writer Single-Reader

Con el siguiente diagrama ilustraremos las diferencias entre un registro regular y un registro atómico.



El registro mostrado es un registro binario que inicialmente guardaba el valor 0 y la operación "escritura" escribe el valor 1. Si el registro cumple con la propiedad de regularidad, es posible que las operaciones "lectura$_1$" y "lectura$_2$" devuelvan cualquiera de los siguientes valores,

lectura$_1$	lectura $_2$
0	0
0	1
1	0
1	1

Sin embargo, si el registro fuese atómico, solo las siguientes combinaciones son válidas,

lectura$_1$	lectura $_2$
0	0
0	1
1	1

En particular es claro que en la ejecución del registro regular donde devuelva a 1 a "lectura$_1$ y 0 a "lectura$_2$" no es posible ordenar de manera consistente a las operaciones, ya que aunque "lectura$_1$" ocurrió antes que "lectura$_2$" el valor devuelto por "lectura$_1$" es más reciente.

En [15] se probó que es posible construir un registro atómico SWSR a partir de registros seguros SWSR. Más tarde en [16] se dio una implementación de registros atómicos *SWMR*[2] a partir de registros atómicos SWSR. Por lo tanto aún si se supone que

[2]Single-Writer Multiple-Reader

solo se dispone del tipo más simple de registro (un registro seguro SWSR) es posible construir registros atómicos SWMR sin suponer nada más sobre el sistema.

2.3. Copia atómica

En el modelo de copia atómica se supone que cada proceso p_i tiene a su disposición un registro r_i, y solo p_i escribe en este registro (para evitar múltiples escrituras concurrentes) pero cualquier proceso puede leerlo, por lo tanto en un sistema con n procesos el modelo de copia atómica es implementado sobre un arreglo de n registros atómicos SWMR.

El objeto de copia atómica posee dos operaciones, una de escritura (WRITE) y otra de lectura (READ). Cuando la operación WRITE es invocada por un proceso p_i con un valor v, entonces el objeto escribe el valor v en el registro r_i. Cuando un proceso invoca la operación READ sobre el objeto de copia atómica, el objeto devuelve un arreglo de longitud n, donde la i'esima entrada corresponde al último valor escrito en el registro r_i (y por lo tanto escrito por el proceso p_i).

Los arreglos devueltos por la operación READ de este objeto son linearizables, es decir que dados dos arreglos devueltos por operaciones READ (posiblemente ejecutadas por distintos procesos) es posible ordenarlos consistentemente en el tiempo. En [11] se describe un algoritmo para implementar un objeto de copia atómica utilizando registros atómicos SWMR, por lo tanto es posible utilizar esta abstracción sin pérdida de generalidad.

En una variante de la copia atómica el objeto solamente posee una operación compuesta WRITEREAD, y cada vez que es invocada, es equivalente a ejecutar una operación WRITE seguida de un READ (nótese que las operaciones NO son ejecutadas atómicamente). Esta variante de copias atómicas resulta más fácil de analizar, y también

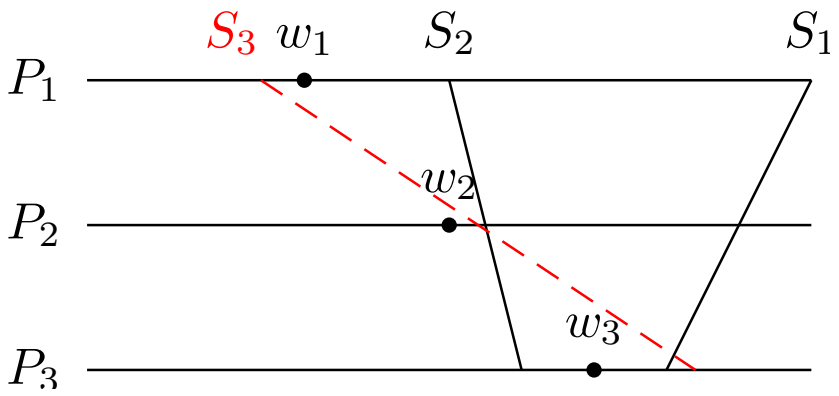
se puede utilizar sin perdida de generalidad ya que es equivalente a la copia atómica tradicional [17].

Sea Π un conjunto de n procesos, donde cada proceso $p_i \in \Pi$ ejecuta la operación $S_i \leftarrow \text{WRITEREAD}(v_i)$ (sin perdida de generalidad $\forall i, j\ v_i \neq v_j$ ya que podemos añadir el identificador del proceso al valor escrito). Supongamos que cada proceso p_i cuenta con un reloj local $time_i$, tal que inicialmente $time_i = 0$ y antes de ejecutar cada operación WRITEREAD se incrementa el valor del reloj en una unidad de tiempo. Definimos $time(S_j[k])$ como el valor del reloj $time_k$ cuando el proceso p_k escribió el valor $S_j[k]$ leído por el proceso p_j. Toda copia atómica devuelta cumplirá con las siguientes propiedades.

1. Auto contención: $\forall p_i \in \Pi.\ v_i \in S_i$

2. Linearizable: Si existe un k tal que $time(S_i[k]) > time(S_j[k])$ entonces $\forall p_l \in \Pi$, $time(S_i[l]) \geq time(S_j[l])$.

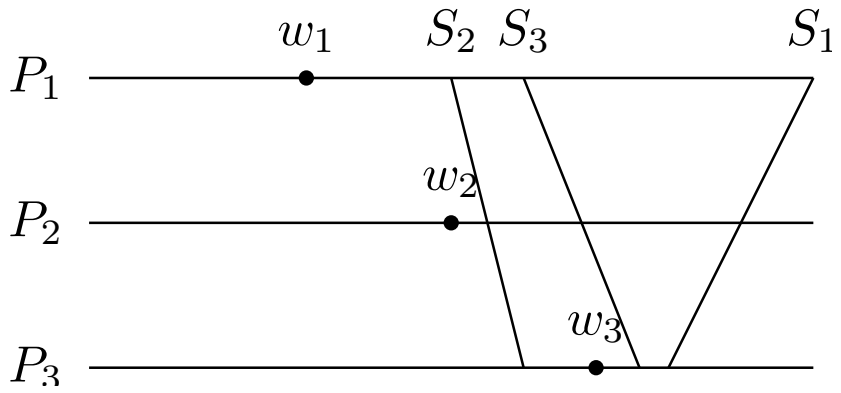
Para ilustrar más claramente lo que implica la propiedad de linearización sobre una copia atómica, el siguiente diagrama presenta una ejecución donde participan tres procesos p_1, p_2 y p_3 y cada uno ejecuta una operación WRITEREAD. Para cada proceso p_i se representan dos eventos, el evento w_i representa la escritura, y el evento S_i representa la lectura de la memoria.



Se puede ver que todos los procesos cumplen con la propiedad de auto contención ya que para todo evento w_i, el respectivo snapshot S_i se ejecuto después.

Sin embargo, los snapshots devueltos no cumplen con la propiedad de linerización, ya que a pesar de que $time(S_2[1]) > time(S_3[1])$ también es cierto que $time(S_3[3]) > time(S_2[3])$. En otras palabras, el proceso p_3 leyó el valor de p_1 antes que p_2, sin embargo p_2 leyó el valor de p_3 antes que p_3.

En contraste, en el siguiente diagrama se presenta una ejecución donde se cumplen tanto la propiedad de auto contención como la de linearización.



2.4. Copia inmediata

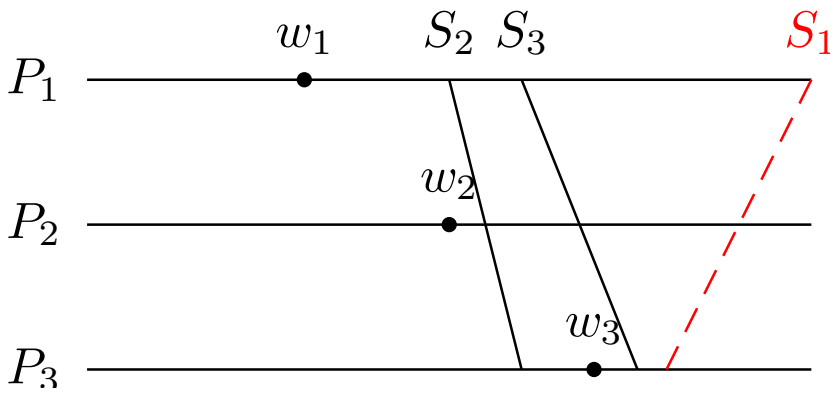
Este modelo es una abstracción de más alto nivel que la copia atómica, y representa un conjunto aún más restringido de ejecuciones. La diferencia clave es que en el modelo de copia atómica las operaciones WRITE y READ eran linearizables de manera separada, pero la operación conjunta WRITEREAD no lo era. En contraste, el modelo de copia inmediata se puede ver como las ejecuciones de copia atómica donde las operaciones WRITE y READ fueron hechas atómicamente, de manera que la operación conjunta WRITEREAD es linearizable.

Sea Π un conjunto de n procesos, donde cada proceso $p_i \in \Pi$ ejecuta la operación $S_i \leftarrow$ ISNAPSHOT(v_i) (sin perdida de generalidad $\forall i, j\ v_i \neq v_j$ ya que podemos

añadir el identificador del proceso al valor escrito). Toda copia inmediata cumple con las propiedades descritas para la copia atómica, y adicionalmente satisface la siguiente propiedad.

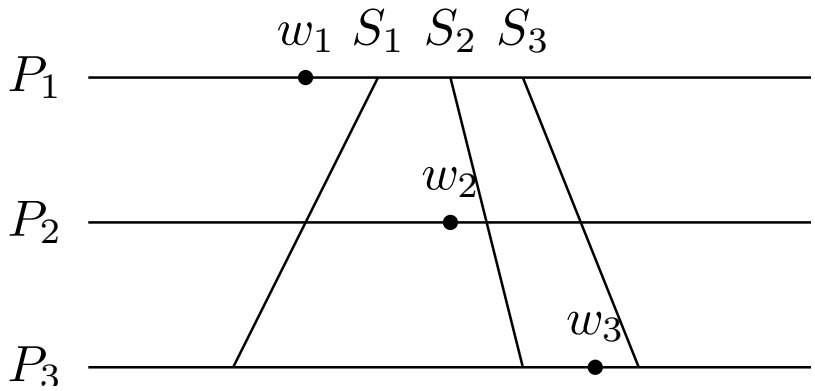
1. Inmediatez: $\forall p_i, p_j \in \Pi.\ v_i \in S_j \Rightarrow S_i \subseteq S_j$

Entonces podemos ver que las ejecuciones válidas de una copia inmediata son un subconjunto de las ejecuciones válidas de copia atómica, es decir que cualquier copia inmediata también es una copia atómica, pero lo converso no es cierto. Para ilustrar esto consideremos el siguiente diagrama.



Observemos que en el diagrama a pesar de que $w_1 \in S_2$ también es cierto que $w_3 \in S_1$ y $w_3 \notin S_2$, y por lo tanto no es cierto que $S_1 \subseteq S_2$. Entonces a pesar de que es una ejecución válida de una copia atómica no cumple con la propiedad de inmediatez y por lo tanto no es una copia inmediata válida.

El siguiente diagrama ilustra una ejecución válida de una copia inmediata, y por lo tanto también de copia atómica.



En este caso se siguen cumpliendo las propiedades de atomicidad y las operaciones separadas siguen siendo linearizables. No obstante, ahora se cumple la propiedad de inmediatez y por lo tanto la operación conjunta WRITEREAD es linearizable por lo que esta es una ejecución válida de una copia inmediata.

2.5. Modelo iterado de copia inmediata

En el modelo iterado de copias inmediata definido en [12] todo proceso tiene acceso a un arreglo compartido de objetos de *uso-único* de copias inmediata $M_0, M_1, \ldots$. Un objeto de uso-único es aquel que solamente permite que cada proceso ejecute una única vez cada operación. En particular en un objeto uso-único de copias inmediata cada proceso p_i puede llamar la operación ISNAPSHOT una sola vez.

Dentro de este modelo se supone que cada proceso comienza utilizando el objeto M_0 y procede en orden a ejecutar ISNAPSHOT del objeto $M_1, M_2, \ldots$. Aunque este modelo parece restringir demasiado la construcción de algoritmos, en [12] se prueba que es equivalente al modelo tradicional de registros compartidos para tareas sin-espera, esto se analizará a fondo en el siguiente capítulo.

Como en este modelo se usan objetos uso-único, cada procesos sólo puede ejecutar una vez la operación ISNAPSHOT y por lo tanto las propiedades que debe cumplir todo

snapshot devuelto (para un mismo objeto) pueden ser descritas de la siguiente manera.

1. Auto contención: $\forall p_i \in \Pi.\ v_i \in S_i$
2. Atomicidad: $\forall p_i, p_j \in \Pi.\ S_i \subseteq S_j$ ó $S_j \subseteq S_i$
3. Instantáneidad: $\forall p_i, p_j \in \Pi.\ v_i \in S_j \Rightarrow S_i \subseteq S_j$

La propiedad de atomicidad es equivalente a la propiedad de linearización para un objeto de tipo uso-único.

2.6. Fallas

En esta sección daremos una breve descripción de algunos tipos de fallas considerados en la ejecución de un sistema distribuido. En general, decimos que un protocolo es tolerante a t fallas de algún tipo, si en cualquier ejecución con n procesos, de los cuales $f \leq t$ tengan fallas de ese tipo, el resultado de la ejecución del protocolo cumple con la especificación del problema que resuelve. Un proceso ha fallado en una ejecución si su comportamiento difiere de aquel descrito por el protocolo que esta ejecutando, de otro modo es correcto.

Para determinar si un proceso ha fallado o no, es necesario especificar una definición de fallas, en las siguientes secciones presentaremos diferentes definiciones de fallas.

2.6.1. Fallas Bizantinas

Las fallas bizantinas recibieron este nombre porque están inspiradas en el problema de los generales bizantinos [18]. En este tipo de fallas un proceso no solo se comporta de manera incorrecta, sino que ademas puede interactuar con otros procesos bizantinos y tratar de “sabotear” la decision del grupo.

Dado que un proceso que sufre de una falla bizantina puede tener un comportamiento arbitrario, este tipo de fallas comprenden a todas las demás (fallas tipo paro, fallas de omisión de mensajes, etc). Por lo tanto, un protocolo que es tolerante a un número t de fallas tipo bizantinas, también es tolerante a al menos t fallas de cualquier otro tipo. Dentro de las fallas bizantinas, a veces se considera que un proceso bizantino es capaz de falsificar mensajes de otros procesos, y en otras ocasiones se supone que los procesos correctos son capaces de autenticar la fuente de los mensajes.

2.6.2. Fallas de omisión

Las fallas de omisión fueron introducidas por [19] y generalizadas por [20]. Un proceso que sufre de este tipo de fallas puede dejar de recibir o de mandar mensajes. Es útil distinguir entre estos dos casos, y en general se dice que:

- Una falla de omisión de mensajes mandados ocurre cuando un proceso p manda un mensaje, pero este mensaje nunca es puesto en el canal de comunicación.
- Una falla de omisión de mensajes recibidos ocurre cuando un mensaje que fue puesto en un canal de comunicación dirigido a un proceso p, y p nunca lo recibe.

Aunque en esta descripción se habla de omisión de mensajes, estas definiciones se extienden naturalmente a modelos basados en memoria compartida. En memoria compartida mandar un mensaje es equivalente a escribirlo en memoria, y recibir un mensaje es análogo a leer la memoria.

Sin embargo, es posible eliminar completamente de una ejecución todas las fallas de tipo omisión siempre y cuando sean intermitentes. Informalmente se puede explicar este resultado suponiendo que cada proceso envía con todo mensaje una historia de todos los mensajes enviados anteriormente. Por lo tanto, si existe una falla de recepción o de envío

en una secuencia finita de mensajes entre dos procesos, al momento de recibir el primer mensaje después de la falla ambos procesos pueden actuar como si todos los mensajes se hubiesen recibido pero con un retraso.

Es decir, dado un proceso p que sufre de fallas de tipo omisión, es posible transformar a p para que todos los demás procesos lo perciban como un proceso correcto, o a un proceso con una falla de tipo paro.

2.6.3. Fallas de tipo paro

Un proceso sufre una falla de tipo paro cuando se detiene prematuramente durante la ejecución y deja de tomar pasos, más aún, una vez que un proceso se detiene no se vuelve a recuperar [21]. Formalmente podemos definir la noción de fallas dentro de una ejecución infinita, donde decimos que todo proceso que tome un número infinito de pasos es correcto, de otra manera ha fallado.

En el resto del documento, prestaremos atención especial a este tipo de fallas, y cabe notar que aunque solo se estén considerando fallas de tipo paro, en el modelo iterado es posible que un proceso p_i que es correcto bajo la definición previa, nunca sea capaz de entregar un mensaje a otros procesos (correctos o incorrectos). Esta situación es descrita en mas detalle en la siguiente sección.

2.6.4. Fallas implícitas

Las fallas implícitas no pueden ser definidas sin tener el contexto de alguna otra definición de fallas. Supongamos que se esta trabajando sobre algún modelo de comunicación con fallas de tipo DEF. Decimos que un proceso p_i sufre de una falla implícita cuando el proceso es incapaz de comunicarse con algún conjunto Q de procesos correctos pero p_i es correcto bajo la definición de DEF.

En particular, consideraremos que un proceso tiene una falla implícita en tiempo t, cuando para $t' > t$ sea incapaz de comunicarse con un conjunto de procesos Q. No tiene sentido considerar fallas transitorias ya que es posible eliminarlas utilizando la misma técnica descrita en la sección 2.6.2.

Por ejemplo, tomemos el modelo iterado de copia inmediata con fallas de tipo paro. Supongamos que la calendarización es tal que el proceso p_i siempre ejecuta la operación ISNAPSHOT al final escribiendo el valor v_i. En ese caso p_i vería a todo valor $v_j \; \forall j$, pero ningún proceso $p_j \neq p_i$ vería el valor v_i. Aunque bajo la definición de fallas de tipo paro p_i es correcto, resulta evidente que p_i es incapaz de comunicarse con el resto del sistema.

En este documento trataremos las fallas implícitas exclusivamente en el contexto de modelos iterados, pero el concepto se puede extender a otras situaciones.

2.6.5. Fallas de equivalencia

Consideremos una ejecución infinita con un conjunto P de procesos participantes, donde los procesos en $P' \subseteq P$ toman un número infinito de pasos (por lo tanto son correctos bajo la definición de fallas de paro). Es posible definir un orden parcial sobre P' utilizando la relación $\preceq$, donde para cualquier par de procesos $p_i, p_j \in P'$ decimos que $p_i \preceq p_j$ sii p_j recibe un número infinito de mensajes de p_i. Decimos que p_i y p_j son equivalentes (escrito $p_i \equiv p_j$) si es cierto que $p_i \preceq p_j$ y $p_j \preceq p_i$.

Cualquier proceso $p_i \in P'$ pertenece a una clase de equivalencia C_i. Más aún, p_i y p_j pertenecen a la misma clase sii $p_i \equiv p_j$. Para dos clases de equivalencia distintas C_i y C_j decimos que C_i es más pequeña que C_j (formalmente $C_i \prec C_j$) sii $\exists p_i \in C_i \wedge \exists p_j \in C_j$ tal que $p_i \preceq p_j$.

Lema 1. *Si* $\exists p_i \in C_i$ *y* $\exists p_j \in C_j$ *tal que* $p_i \preceq p_j$ *entonces* $\forall p_{i'} \in C_i . \forall p_{j'} \in C_j . p_{i'} \preceq p_{j'}$.

Demostración Fijemos $p_{i'} \in C_i$, como $p_i \in C_i$ entonces $p_i \preceq p_{i'}$. Fijemos $p_{j'} \in C_j$, como $p_j \in C_j$ entonces $p_{j'} \preceq p_j$. Finalmente, como $p_i \preceq p_j$ usando transitividad tenemos que $p_{i'} \preceq p_{j'}$. ■

Teorema 1. *Para dos clases distintas, si $C_i \prec C_j$ entonces $C_j \not\prec C_i$.*

Demostración Como $C_i \prec C_j$ entonces podemos aplicar el lema 1 y tenemos que $\forall p_i \in C_i . \forall p_j \in C_j \; p_i \preceq p_j$. Ahora si suponemos que $C_j \prec C_i$ entonces podemos volver aplicar el lema 1 y obtenemos que $\forall p_i \in C_i . \forall p_j \in C_j \; p_i \preceq p_j$. Combinando estos dos resultados obtenemos que $\forall p_i \in C_i . \forall p_j \in C_j \; p_i \equiv p_j$, y por lo tanto C_i y C_j son la misma clase, lo cual es una contradicción. ■

Teorema 2. *Para el modelo iterado de copias inmediatas y el modelo SWMR, cualquier par de clases distintas C_i y C_j son comparables, es decir siempre sucede que $C_i \prec C_j$ ó $C_j \prec C_i$.*

Demostración Supongamos lo contrario, entonces para todo $p_i \in C_i$ y $p_j \in C_j$ p_i y p_j son incomparables. Tomemos dos clases de equivalencia C_i y C_j, por definición $p_i \in C_i$ y $p_j \in C_j$ mandan un número infinito de mensajes que son recibidos por algún proceso. Como estos procesos mandan un número infinito de mensajes, entonces necesariamente ejecutan la operación WRITEREAD un número infinito de veces.

Por la propiedad de atomicidad de ambos modelos, p_i recibió un número infinito de mensajes de p_j o p_j recibió un número infinito de mensajes de p_i. Por lo tanto por definición es cierto que $p_i \preceq p_j$ ó $p_j \preceq p_i$. En cualquier eso implica que que C_i es comparable con C_j. ■

Ya que todas las clases de equivalencia son comparables podemos definir a la clase de equivalencia $C_{\text{mín}}$ como la más pequeña de todas las clases. Diremos que un proceso es correcto bajo la definición de fallas de equivalencia sii el proceso está en la clase $C_{\text{mín}}$. La primera observación interesante sobre esta nueva definición de fallas es que si existe al menos un proceso correcto bajo la definición de fallas de paro, entonces también existe al menos un proceso en $C_{\text{mín}}$.

Similarmente todo proceso en $C_{\text{mín}}$ es correcto bajo la definición de fallas de paro. Podemos observar que la diferencia más importante entre estas dos definiciones se da cuando existe más de una clase de equivalencia, en ese caso existe una clase $C_i \neq C_{\text{mín}}$, y todos los procesos en C_i son correctos bajo la definición de fallas de paro, pero no bajo la definición de fallas de equivalencia. Es decir que si se esta en un modelo de cómputo donde esto ocurra, y se esta utilizando la definición de fallas de tipo paro, entonces están ocurriendo fallas implícitas que serían eliminadas si se utilizara la definición de fallas de equivalencia.

2.7. Detectores de fallas no confiables

Como ya se mencionó antes, los sistemas distribuidos asíncronos son de especial interés por su simplicidad y aplicabilidad. Sin embargo, cuando se contemplan fallas, incluso una sola falla de tipo paro, es imposible resolver problemas relevantes como el consenso [2] y atomic broadcast [22] entre otros. Muchos algoritmos de interés práctico utilizan como base estos dos problemas, y por lo tanto para que el modelo asíncrono sea útil es necesario modificarlo. Dos de las modificaciones más populares para modelos asíncronos son los protocolos aleatorios y los detectores de fallas [8]. En este documento nos enfocaremos en la segunda.

En el marco de los detectores de fallas no confiables, cada proceso tiene acceso a

un módulo local de detección de fallas. Como los detectores de fallas no son confiables entonces pueden cometer errores, y por ser locales es posible que dos procesos distintos estén recibiendo distinta información sobre las posibles fallas. Un detector de fallas esta definido por dos propiedades que llamaremos completez y precisión. A continuación daremos una definición formal de un detector de fallas siguiendo la descripción de [8] y [9].

2.7.1. Patrón de fallas

Consideremos únicamente las fallas de tipo paro (como vimos antes las fallas de tipo omisión pueden ser eliminadas o descritas por una falla de tipo paro) y las fallas de equivalencia. En cualquiera de estos casos, las fallas no son intermitentes ya que una vez que un proceso falla, no se recupera.

Un patrón de fallas P es un una función que va de $\mathbb{R}$ a 2^{Π}, donde $P(t)$ representa la lista de procesos que han fallado hasta el tiempo t. Como los procesos no se recuperan de las fallas podemos decir que $\forall t \ : \ P(t) \ \subseteq \ P(t + \Delta)$. Definamos $incorrectos(P) = \bigcup_{t \in \mathbb{R}} P(t)$ y $correctos(P) = \Pi - incorrectos(P)$. Solamente son de interés los patrones de fallas donde $correctos(P) \neq \emptyset$.

2.7.2. Historia del detector de fallas

Cualquier detector detector de fallas $\mathcal{F}$ entrega en el tiempo t un valor $\mathcal{F}(t)$ dentro del dominio $Dom(\mathcal{F})$ que codifica información sobre las fallas del sistema. La historia H de $\mathcal{F}$ es una función que va de $\Pi \times \mathbb{R}$ a $Dom(\mathcal{F})$. Es decir que $H(i, t)$ representa el valor devuelto por el módulo local de detección de fallas del proceso i en el tiempo t.

En particular consideremos aquel detector de fallas $\mathcal{F}'$ que como salida entrega una lista de procesos que se sospecha han fallado. Entonces $H(i, t)$ es una función que va

de $\Pi \times \mathbb{R}$ a 2^{Π} y le entrega al proceso i en el tiempo t cuantos procesos sospecha han fallado. En particular si $j \in H(i,t)$ diríamos que en el tiempo t el proceso i sospecha que el proceso j ha fallado. Formalmente, un detector de fallas $\mathcal{F}$ es una función que mapea todo patron de fallas P a un conjunto de historias posibles $\mathcal{F}(P)$.

2.7.3. Propiedades de un detector de fallas

Como ya se menciono, cualquier detector de fallas $\mathcal{F}$ se define en términos de dos propiedades llamadas completez y precisión.

Completez

- Completez fuerte: Eventualmente todo proceso que falla se encontrara bajo sospecha permanente.
- Completez débil: Eventualmente todo proceso que falla es permanentemente sospechado por algún proceso correcto.

En [8] se describe un algoritmo que transforma cualquier detector de fallas que satisfaga la propiedad de completez débil a completez fuerte, y por lo tanto por simplicidad se supondrá que los detectores de fallas cumplen con completez fuerte.

Precisión

- Precisión fuerte: Ningún proceso es puesto bajo sospecha por un proceso correcto antes de fallar.
- Precisión débil: Algún proceso correcto nunca es sospechado por un proceso correcto.

- Precision eventual fuerte: Eventualmente ningún proceso correcto es sospechado por ningún proceso correcto.
- Precisión eventual débil: Eventualmente algún proceso correcto no es sospechado por ningún proceso correcto.

Capítulo 3

Equivalencia entre modelos

En el capítulo pasado se describieron algunos de los modelos de comunicación basados en memoria compartida más relevantes. En este capítulo compararemos el modelo de registros atómicos SWMR y el modelo iterado de copia inmediata, primero consideraremos los dos modelos originales, y después los analizaremos cuando han sido enriquecidos con detectores de fallas. Por comodidad cuando utilicemos el término de registros atómicos nos referimos a los registros atómicos SWMR descritos en el capítulo 2.

3.1. Registros atómicos y el modelo iterado de copia inmediata

El modelo de copia atómica descrito en [11] es equivalente al de registros atómicos ya que es posible implementar los registros usando objetos de copia atómica y viceversa. Más aun, en [17] se describe una variante del modelo de copia atómica en el que se utiliza un objeto compartido con una sola operación compuesta WRITEREAD, ejecutar esta operación es equivalente a ejecutar un WRITE seguido por un READ. Cuando sea conveniente utilizaremos para las demostraciones el modelo de copia atómica ya que

es equivalente al de registros atómicos y por lo tanto se puede utilizar sin pérdida de generalidad.

Por otro lado, en el modelo iterado de copia inmediata se cuenta con un arreglo infinito de objetos uso-único $M_0, M_1, \ldots$, cada objeto es usado en secuencia y solamente provee la operación ISNAPSHOT. En la siguiente sección probaremos que es posible simular el modelo iterado de copia inmediata dentro del modelo de registros atómicos.

3.1.1. El modelo iterado de copia inmediata usando registros atómicos

Para simular el modelo iterado de copia inmediata en el modelo de registros atómicos es suficiente probar que es posible implementar un algoritmo que pueda ser ejecutado una vez por cada proceso (objetos de uso-único) y que cumpla con las características de ISNAPSHOT. En [12] se describe el algoritmo del *conjunto participante* que implementa elegantemente la operación ISNAPSHOT.

Algoritmo 1 Conjunto participante usando registros atómicos para el proceso p_i

```
1:  function ParticipatingSet(v)
2:      W[i] ← v
3:      F[i] ← n + 1
4:      repeat
5:          F[i] ← F[i] − 1
6:          for j ∈ 1 . . . n do f_i[j] ← F[j]
7:          S_i ← { W[j] | f_i[j] ≤ F[i] }
8:      until |S_i| ≥ F[i]
9:      return S_i
10: end function
```

Intuitivamente el algoritmo se puede visualizar como un cono, donde todos los procesos empiezan en la parte más ancha del cono (donde caben exactamente n procesos) y van cayendo hacia abajo (el cono termina en el nivel 1, donde sólo puede haber 1 proceso). En esta implementación se utilizan dos arreglos W y F de n registros atómicos.

Al inicio del algoritmo el proceso p_i escribe el valor de entrada a la copia inmediata en la entrada i'esima de W e inicializa la i'esima entrada de F con el valor $n + 1$. En cada ronda, los procesos decrementan la variable F (su nivel dentro del cono) y leen el valor de F de los n procesos (línea 6). Decimos que un proceso p_j esta en un nivel $level(p_j) = l_j$ cuando $F[j] = l_j$, y el proceso devuelve la copia inmediata cuando ve a l_j o más procesos en el mismo nivel o niveles inferiores.

Para probar que el algoritmo del conjunto participante implementa correctamente un objeto uso-único de copia inmediata es necesario probar que todos los conjuntos posibles devueltos por este algoritmo cumplen con las propiedades de auto-contención, atomicidad e inmediatez.

Lema de auto-contención. *Si un proceso p_i invoca el algoritmo del conjunto participante con un valor v_i, y el algoritmo devuelve un conjunto S_i, entonces $v_i \in S_i$.*

Demostración Sea S_i el conjunto devuelto cuando p_i invocó el algoritmo del conjunto participante con el valor v_i. Como el algoritmo devolvió un conjunto S_i entonces ejecuto las líneas 1-9 al menos una vez, y al ejecutar la línea 2 escribió el valor v_i en la variable $W[i]$.

Más aun, el arreglo local f_i contiene el resultado de la lectura de los registros atómicos compartidos F (línea 6), y como los registros atómicos son SWMR y solamente el proceso p_i escribe en la entrada i, entonces se puede garantizar que $f_i[i] = F[i]$.

Por lo tanto, cuando se ejecuta la línea 7 siempre es cierto que $f_i[i] = F[i] \leq F[i]$ y cualquier conjunto S_i devuelto contiene el elemento $W[i]$ donde el proceso escribió su

valor v_i. ■

Lema de nivel. *Un proceso p_i en un nivel l_i ve a lo más l_i procesos en su nivel y niveles inferiores.*

Demostración Supongamos por contradicción que existe un tiempo t en el cual un proceso p_i en un nivel l_i ve a más de l_i procesos. Es decir, el proceso p_i ve a un conjunto P tal que $|P| \geq l_i + 1$. Como un proceso solamente agrega a otros procesos al conjunto de salida cuando tienen un nivel inferior al suyo entonces para cualquier proceso $p \in P$ es cierto que $level(p) \leq l_i$.

Sabemos que $l_i < n$ ya que de lo contrario implicaría que $|P| \geq n + 1$ y esto es imposible porque solo participan n procesos. Más aún, como el nivel de un proceso es estrictamente decreciente entonces existe un tiempo $t' < t$ y un conjunto $Q \subseteq P$ tal que en t' $\forall q \in Q$, $level(q) = l_i + 1$ y $\forall p \in (P - Q)$, $level(p) \leq l_i$.

Sin embargo, eso implica que al menos un proceso $q \in Q$ ve a $l_i + 1$ en su nivel y niveles inferiores, y por lo tanto q terminaría el algoritmo y nunca bajaría al nivel l_i, lo que es una contradicción. ■

Un proceso no detiene su ejecución en un nivel l_i hasta no ver a al menos l_i procesos en su nivel y niveles inferiores, y por el lema de nivel nunca hay más de l_i procesos en el nivel l_i e inferiores. Por lo tanto, una consecuencia del lema de nivel es que un proceso termina en el nivel l_i sii ve a exactamente l_i procesos en el nivel actual e inferiores.

Lema de atomicidad. *Sean S_i y S_j conjuntos devueltos por el algoritmo del conjunto participante, entonces $S_i \subseteq S_j$ ó $S_j \subseteq S_i$.*

Demostración Sea S_i el conjunto devuelto por el proceso p_i con $|S_i| = l_i$, y sea S_j el conjunto devuelto por el proceso p_j con $|S_j| = l_j$, más aun por el lema de nivel sabemos que p_i terminó en el nivel l_i y p_j terminó en el nivel l_j.

Supongamos sin pérdida de generalidad que $l_i \geq l_j$, entonces es cierto que el proceso p_j vio menos procesos (y de niveles menores) que p_i, probaremos que esto implica que $S_j \subseteq S_i$.

Supongamos por contradicción que existe un proceso $p_k \in S_j$ que no es visto por p_i (es decir $p_k \notin S_i$). Eso implica que en cuando p_i devolvió S_i, $nivel(p_k) > l_i$. Sin embargo, como eventualmente p_k es visto por p_j entonces eventualmente $nivel(p_k) \leq l_j$ y como se supuso que $l_i \geq l_j$ eso implica que $nivel(p_k) \leq l_i$.

Por lo tanto en algún punto se cumplió que $nivel(p_k) = l_i + 1$, y $\forall p \in S_i, nivel(p) \leq l_i$. Pero esto implica que cuando p_k esta en el nivel $l_i + 1$ ve a $l_i + 1$ o más procesos y por lo tanto terminaría y nunca bajaría al nivel l_i para ser visto por p_j, lo cual es una contradicción. ■

Lema de inmediatez. *Sea S_i y S_j conjuntos devueltos por el algoritmo del conjunto participante, si suponemos que $p_j \in S_i$, eso implica que $S_j \subseteq S_i$.*

Demostración Sea S_i el conjunto devuelto por el proceso p_i con $|S_i| = l_i$, y sea S_j el conjunto devuelto por el proceso p_j con $|S_j| = l_j$. Por el lema de nivel sabemos que p_i terminó en el nivel l_i y p_j terminó en el nivel l_j.

Como se cumple la propiedad de atomicidad es cierto que $S_i \subseteq S_j$ ó $S_j \subseteq S_i$, y como p_j fue visto por p_i ($p_j \in S_i$) entonces sabemos que $l_j \leq l_j$. Esto implica que no es posible que $S_i \subsetneq S_j$, y por eliminación se cumple que $S_j \subseteq S_i$. ■

Con los lemas de auto-contención, atomicidad e inmediatez hemos probado que el algoritmo del conjunto participante puede ser utilizado para implementer un objeto uso-único de copia inmediata. De aquí se desprende que cualquier algoritmo en IIS puede ser simulado (y por lo tanto implementado) por el modelo de registros atómicos. En la siguiente sección examinaremos la dirección opuesta.

3.1.2. Registros atómicos usando el modelo iterado de copia inmediata

Para simular el modelo registros atómicos en el modelo iterado de copia inmediata es suficiente probar que existe un algoritmo que implementa la operación WRITEREAD del modelo de copia atómica (que ya se probo equivalente a registros atómicos en [17]). En [7] se describe un algoritmo que aquí llamaremos *convergencia vectorial* que implementa esa operación.

Algoritmo 2 Convergencia vectorial en IIS para el proceso p_i

1: **function** CONVERGINGVECTOR(v_i)
2: $W_i[i] \leftarrow v_i$
3: $F_i[i] \leftarrow F_i[i] + 1$
4: **repeat**
5: $S_i \leftarrow M_{r_i}.ISnapshot(< F_i, W_i >), r_i \leftarrow r_i + 1$
6: **for** $j \in 1 \ldots n$ **do**
7: $k \leftarrow argmax\{F_k[j] | k \in \{1, \ldots, n\}\}$
8: $< F_i[j], W_i[j] > \leftarrow < F_k[j], W_k[j] >$
9: **until** $\forall < F, - > \in S_i, F = F_i$
10: **return** W_i
11: **end function**

El algoritmo supone que inicialmente el vector W_i contiene valores nulos y el vector F_i fue inicializado a 0. En particular la variable $F_i[i]$ cumple el propósito de ser el reloj local de cada proceso, ya que es cierto que inicialmente $F_i[i] = 0$ y al ejecutar cada llamada al algoritmo de convergencia vectorial se incrementa dicha variable (línea 3).

Para probar la correctez de este algoritmo primero hay que probar que todo vector devuelto por el mismo cumple con las propiedades de auto contención y linearización.

Finalmente, es necesario probar que toda invocación de este algoritmo devuelve un vector (es decir, que el algoritmo termina).

Lema de auto-contención. *Todo vector W_i devuelto por una llamada al algoritmo de convergencia vectorial con parámetro v_i es tal que $W_i[i] = v_i$.*

Demostración Como el algoritmo devolvió un vector W_i entonces ejecutó las líneas 1-10 al menos una vez. Al ejecutar la línea 2 escribió el valor v_i en la entrada $W_i[i]$.

Más aun, todos los procesos mantienen un estimado de niveles F_i, y solo el proceso p_i escribe en la i'esima entrada de dicho estimado. Entonces en la línea 7 cuando $j = i$ siempre es cierto que $k = i$, y por lo tanto $W_i[i]$ no es modificado y el conjunto devuelto siempre es tal que $W_i[i] = v_i$. ■

Lema de linearización. *Sean W_i y W_j vectores devueltos por el algoritmo de convergencia vectorial, dichos vectores son linearizables.*

Demostración Supongamos por contradicción que W_i y W_j no son linearizables, entonces existe un k tal que $time(W_i[k]) > time(W_j[k])$ y existe una l tal que $time(W_i[l]) < time(W_j[l))$.

Sin pérdida de generalidad suponemos que el vector W_i fue devuelto por el proceso p_i después de usar el objeto uso-único de copia inmediata M_p y el vector W_j fue devuelto por el procesos p_j después de usar el objeto M_q y que $p \geq q$.

Dado que $p \geq q$ sabemos que el proceso p_i tomo una copia inmediata usando el objeto M_q. Existen dos posibilidades, si $p = q$ entonces tanto p_i como p_j vieron un solo estimado y por la propiedad de atomicidad el estimado es el mismo y por lo tanto $W_i = W_j$. Si $p > q$ entonces p_i vió a más de un estimado, porque de otro modo hubiese terminado después de usar el objeto q y nunca hubiese usado el objeto p.

Sin embargo, el proceso p_j sí terminó después de usar el objeto M_q y por lo tanto vio un sólo vector F_j. Entonces por la propiedad de atomicidad de la copia inmediata es cierto que p_i vio el estimado F_j en su copia inmediata sobre el objeto M_q.

Más aun, como las actualizaciones de los vectores de estimados son estrictamente crecientes, entonces cualquier entrada de F_i después de utilizar el objeto M_p es estrictamente más grande que F_j. Sin embargo esto es una contradicción ya que habíamos supuesto que existe una l tal que $time(W_i[l]) < time(W_j[l])$. ■

Hasta ahora hemos probado que todo vector devuelto por el algoritmo de convergencia vectorial satisface las propiedades de auto-contención y linerización. Sin embargo, falta probar que toda invocación del algoritmo devuelve un vector (y por lo tanto termina). Como veremos en lo que resta de esta sección, esta pregunta no es trivial, y si no se hacen ciertas suposiciones sobre el protocolo que se esta simulando usando el algoritmo de convergencia vectorial no se puede garantizar que la simulación siempre termine.

En particular, probaremos que para el caso de protocolos sin-espera [23] el algoritmo de convergencia vectorial termina.

Definición 3. *Diremos que un protocolo es sin-espera si satisface las dos siguientes condiciones (tal como son definidas en [23])*

1. *Ningún proceso toma un numero infinito de pasos sin decidir.*

2. *Si un proceso no decidido no tiene invocaciones pendientes, entonces ha habilitado una transición de salida (para tomar una decisión u otra invocación).*

Dado que estamos simulando un protocolo de registros atómicos en el modelo iterativo de copia inmediata, una invocación al algoritmo de convergencia vectorial es vista solamente como un paso para el protocolo que esta siendo simulado. Ahora probaremos

que cuando se simulan protocolos sin-espera el algoritmo de convergencia vectorial termina.

Teorema 3. *Si se esta simulando un protocolo sin-espera en el modelo de registros atómicos utilizando el algoritmo de convergencia vectorial toda invocación al algoritmo por un proceso correcto devuelve un vector.*

Demostración Supongamos que existe una ejecución de un protocolo sin-espera donde algún proceso correcto p_i invoca el algoritmo de convergencia vectorial y nunca termina. Eso implica que en la condición en la línea 9 nunca se cumple, y por lo tanto siempre ve al menos un vector F distinto del suyo.

Para que esto suceda un número infinito de veces es necesario que exista un proceso p_j que completa un número infinito de llamadas al algoritmo de convergencia vectorial y por lo tanto ejecuta un número infinito de pasos. Sin embargo, la definición de un protocolo sin-espera exige que este proceso p_j termine en un número finito de pasos, lo cual es una contradicción. ■

Por el teorema pasado y los lemas de auto-contención y linearización del algoritmo de convergencia vectorial podemos concluir que cualquier protocolo sin-espera en el modelo de registros atómicos puede ser simulado en el modelo iterado de copia inmediata. Combinando esto con los resultados de la sección pasada, hemos probado que para protocolos sin-espera el modelo de registros atómicos y el modelo iterado de copia inmediata son equivalentes.

Una pregunta natural es si es posible que exista algún algoritmo capaz de simular la operación básica de WRITEREAD del modelo registros atómicos dentro del modelo iterado de copia inmediata independientemente del uso de esta operación. En particular, ¿Existe algún algoritmo que simule la operación WRITEREAD dentro del modelo iterado de copia inmediata que sirva para protocolos que no sean sin-espera?

Teorema 4. *No existe un algoritmo capaz de simular la operación* WRITEREAD *dentro del modelo iterado de copia inmediata.*

Para probar este teorema, mostraremos que es posible que el agendador de procesos "juegue" en contra nuestra y no permita que ningún algoritmo sea capaz de simular la operación WRITEREAD para protocolos arbitrarios.

Demostración Supongamos que existe un algoritmo capaz de simular la operación WRITEREAD. Sean p_i y p_j procesos participando en este algoritmo, y supongamos que la calendarización de los procesos es tal que p_i y p_j están en dos clases de equivalencia distintas[1] y $p_i \preceq p_j$.

Como el algoritmo funciona independientemente del protocolo que se esta simulando, supondremos un protocolo en el que p_i ejecuta un número infinito de operaciones WRITEREAD (dicho protocolo no es sin-espera).

Sea W_j el primer vector devuelto por p_j. Dado que p_i ejecuta un número infinito de operaciones WRITEREAD existe un vector W_i devuelto por p_i tal que $time(W_i[i]) > time(W_j[i])$. Sin embargo, este vector forzosamente rompe con la propiedad de linearización ya que necesariamente es cierto que $time(W_j[j]) > time(W_i[j])$ porque los valores propuestos por p_j nunca son vistos por p_i porque ambos pertenecen a clases de equivalencia distintas. ■

Entonces por el teorema 4 sabemos que no existe ningún algoritmo en el modelo de iterado de copia inmediata capaz de simular la operación WRITEREAD para protocolos arbitrarios. En la siguiente sección examinaremos que ocurre cuando enriquecemos tanto el modelo registros atómicos como el modelo iterado de copia inmediata con detectores de fallas.

[1]Nos estamos refiriendo a las clases de equivalencia definidas en la sección 2.6.5

3.2. Enriqueciendo los modelos con detectores de fallas

Dado un modelo de comunicación es posible enriquecerlo con un detector de fallas $\mathcal{F}$ al proveer a cada proceso con una operación FD-QUERY que al ser invocada devuelve información sobre las fallas. El tipo de información devuelta y la calidad de la misma depende del detector de fallas $\mathcal{F}$ que se este usando.

Además de cumplir con la especificación de $\mathcal{F}$ la información devuelta por FD-QUERY depende del patrón de fallas, y el patrón de fallas esta definido en función de la definición de una falla. En general utilizaremos la variable $\mathcal{F}$ para referirnos a cualquier detector de fallas, cuando la definición de fallas sea de tipo paro (descrita en la sección 2.6.3).

3.2.1. El modelo iterado de copia inmediata usando registros atómicos cuando ambos cuentan con un detector de fallas

En la sección 3.1.1 presentamos un algoritmo capaz de simular la operación de IS-NAPSHOT en el modelo registros atómicos. Tanto la correctez de los conjuntos devueltos por el algoritmo del conjunto participante como el hecho de que toda invocación del algoritmo termina, no dependen de características del patrón de uso de los objetos de copia inmediata. Por lo tanto, cualquier protocolo especificado en el modelo de registros atómicos con un detector de fallas $\mathcal{F}$ puede ser simulado dentro del modelo iterado de copia inmediata enriquecido con el mismo detector de fallas $\mathcal{F}$.

3.2.2. Registros atómicos usando el modelo iterado de copia inmediata cuando ambos el sistema tiene detectores de fallas

El algoritmo de convergencia vectorial descrito en la sección 3.1.2 es capaz de simular una operación WRITEREAD dentro del modelo iterado de copia inmediata, siempre y cuando el protocolo que este usando la operación WRITEREAD sea sin-espera.

En un modelo enriquecido con detectores de fallas la noción de sin-espera no se puede aplicar en el sentido convencional. Cuando un modelo de comunicación no tiene acceso a un detector de fallas, es factible exigir que en un protocolo sin-espera todo proceso decida después de ejecutar un número finito de pasos, independientemente del progreso de los demás procesos. Sin embargo, en un sistema con detección de fallas un proceso podría esperar por otros procesos correctos antes de terminar (ya que el detector de fallas le garantiza que la espera no será infinita).

En sistemas sin detección de fallas la decisión de un proceso correcto no está condicionada por el progreso de algún otro proceso ya que en un sistema asíncrono es imposible distinguir entre un proceso que ha fallado (con una falla de tipo paro) de uno que es extremadamente lento. Es precisamente esta limitación la que provoca que en modelos de comunicación como el de registros atómicos no se pueda resolver el problema del consenso. Sin embargo, esto no es cierto cuando el sistema tiene detectores de fallas, y por lo tanto la definición tradicional de protocolos sin-espera no es útil. No obstante, en sistemas sin detección de fallas muchas veces se utiliza el término sin-espera para describir protocolos tolerantes a $n - 1$ fallas, y en este sentido si se puede aplicar esta definición a sistemas con detectores de fallas.

Por lo tanto, como consecuencia del teorema de imposibilidad de la sección 3.1.2 podemos concluir que el algoritmo de convergencia vectorial no funciona para simular protocolos que utilicen detectores de fallas (ya que no cumplen con la restricción de

ser sin-espera).Más aun, no existe ningún algoritmo capaz de simular todos los protocolos en el modelo de registros atómicos enriquecido dentro del modelo iterado de copia inmediata cuando el sistema cuenta con detectores de fallas.

Este resultado nos lleva a buscar una tarea que se pueda resolver en un modelo de registros atómicos con un detector de fallas$\mathcal{F}$ y que no tenga solución en el modelo iterado de copia inmediata enriquecido con $\mathcal{F}$. En [9] se probó que Ω es el detector de fallas más débil para resolver el consenso y en [24, 25] se describen algoritmos para resolver el consenso en el modelo de registros atómicos utilizando Ω.

Lema 2. *En el modelo iterado de copia inmediata enriquecido con un detector de fallas* $_\Omega$ *el problema del consenso no tiene solución.*

Para probar este resultado reduciremos el problema de resolver el consenso en un sistema con n procesos enriquecido con Ω, a resolver el consenso distribuido en un sistema con $n-1$ procesos sin un detector de fallas. En [2] ya se probó que resolver el consenso distribuido tolerante a una falla en un sistema asíncrono es imposible, por lo tanto podemos concluir que no es posible resolver el consenso en el modelo iterado de copia inmediata enriquecido con Ω.

Demostración Supongamos que existe un algoritmo que resuelve el consenso en el modelo iterado de copia inmediata enriquecido con el detector de fallas Ω. Consideremos una ejecución donde participan n procesos y el proceso p_1 es correcto y Ω lo devuelve como líder desde el principio de la ejecución.

Es posible que el el proceso p_1 siempre sea calendarizado al final cuando ejecuta la operación ISNAPSHOT y por lo tanto ningún otro proceso reciba mensajes de p_1. Sin embargo, si esto sucede entonces implicaría que los procesos $p_2, \ldots, p_n$ nunca ven al proceso p_1 y por lo tanto el detector de fallas Ω no les da ninguna información útil.

Por lo tanto si fueran capaces de resolver el consenso implicaría la existencia de un algoritmo para resolver el consenso con $n - 1$ procesos sin utilizar detectores de fallas, lo cual se sabe imposible [2]. ■

Hasta ahora hemos probado que el modelo de registros atómicos enriquecido con un detector de fallas $\mathcal{F}$ es estrictamente más poderoso que el modelo iterado de copia inmediata enriquecido con el mismo detector de fallas. Por lo tanto, los resultados demostrados hasta este punto apoyan que el modelo iterado de copia inmediata no es apropiado para el estudio de algoritmos que requieran el uso de detectores de fallas. En la siguiente sección estudiaremos métodos para tratar de superar esta limitación.

3.3. Superando las limitaciones del modelo iterado de copia inmediata

Una alternativa para aprovechar las ventajas del modelo iterado de copia inmediata y poder utilizar el concepto de detectores de fallas, es alterar la definición de fallas a una que sea más adecuada al modelo de comunicación. En la sección 2.6.5 se definieron las fallas de equivalencia, en esta sección analizaremos las características del modelo de registros atómicos y el modelo de copia inmediata utilizando esta definición de fallas.

3.3.1. Registros atómicos usando fallas de equivalencia

Consideremos el modelo de registros atómicos enriquecido con detectores de fallas donde en vez de considerar las fallas tradicionales consideramos las fallas de equivalencia definidas en la sección 2.6.5. Utilizando los conceptos de las fallas de equivalencia podemos hacer la siguiente observación.

Lema 3. *Si tanto p_i como p_j invocan la operación* WRITEREAD *un número infinito de veces entonces ambos pertenecen a la misma clase de equivalencia.*

Demostración Tomemos una invocación de la operación WRITEREAD por p_i en algún tiempo t. Existe un tiempo $t' > t$ en el cual p_j ejecuta una operación WRITEREAD y ve al proceso p_i (por la propiedad de linearización), de lo contrario implicaría que p_j deja de invocar operaciones WRITEREAD antes del tiempo t' (lo cual contradice la suposición del lema). Entonces por definición es cierto que $p_i \preceq p_j$.

De manera similar, para toda invocación de WRITEREAD por el proceso p_j en algún tiempo τ, existe un tiempo $\tau' > \tau$ en el cual p_i ejecuta una operación WRITEREAD y ve al proceso p_j. Esto implica que $p_j \preceq p_i$.

Como se cumple que $p_i \preceq p_j$ y $p_j \preceq p_i$ podemos concluir que $p_i \equiv p_j$ y por lo tanto ambos procesos pertenecen a la misma clase de equivalencia. ■

Del lema 3 se sigue que en el modelo de registros atómicos todos los procesos que son correctos bajo la definición de fallas tipo paro pertenecen a la misma clase de equivalencia. Más aun, ya se había visto que si existe al menos un proceso correcto bajo la definición de fallas tipo paro entonces la clase $C_{\text{mín}}$ esta bien definida, y por lo tanto todos los procesos correctos bajo la definición de fallas tipo paro también son correctos bajo la definición de equivalencia. En el caso no haya ningún proceso correcto bajo la definición tradicional, por vacuidad se cumple el mismo enunciado.

Corolario 1. *Del lema 3 se sigue que la definición tradicional de fallas y la definición de equivalencia producen exactamente la misma partición de procesos correctos e incorrectos en el modelo de registros atómicos.*

Sea $\mathcal{F}$ un módulo de detección de fallas bajo la definición de fallas de paro, y $\mathcal{F}'$ un módulo de detección de bajo la definición de fallas de equivalencia. Como ya vimos, en

el modelo puro de registros atómicos ambos módulos tienen la misma especificación. Sin embargo, de manera independiente al modelo de comunicación podemos abstraer el módulo de $\mathcal{F}'$ como una extensión de $\mathcal{F}$ que toma en cuenta información adicional sobre el patrón de lecturas/escrituras de los procesos.

Definición 4. *$SWMR_{F_C}$ es el modelo de registros atómicos enriquecido con un módulo de detección de fallas $\mathcal{F}'$ modificado para tomar la información de lecturas/escrituras de los procesos del resultado de cada ejecución del algoritmo del conjunto participante.*

Por el corolario 1 se sigue que si el modelo SWMR_{F_C} se utiliza para cualquier propósito que no involucre invocar el algoritmo del conjunto participante (y por lo tanto no se utilice la información "extra" sobre el patrón de lectura y escritura) entonces su comportamiento es idéntico al del modelo de registros atómicos enriquecidos con un módulo $\mathcal{F}$.

Supongamos que existe una tarea que tiene solución en el modelo iterado de copia inmediata enriquecido con un detector de fallas $\mathcal{F}'$ bajo la definición de fallas de equivalencia. Como ya se probo en la sección 3.1.1 el algoritmo del conjunto participante se puede utilizar para simular la operación de ISNAPSHOT y la salida del módulo de detección de fallas $\mathcal{F}'$ será equivalente a la del módulo $\mathcal{F}_C$ utilizado por el modelo SWMR_{F_C}.

Por lo tanto, es posible simular cualquier protocolo dentro del modelo iterado copia inmediata con un detectores de fallas $\mathcal{F}'$ utilizando el modelo SWMR_{F_C}.

3.3.2. Modelo iterado de copia inmediata usando fallas de equivalencia

Las fallas de equivalencia presentan ventajas cuando son utilizadas en el modelo iterado de copia inmediata, ya que capturan a las fallas implícitas descritas en la sección

2.6.4. Formalmente, decimos que un proceso p_i sufre una falla implícita cuando la calendarización es tal que el proceso p_i nunca es visto por uno o más procesos correctos, pero es considerado correcto bajo la definición de fallas de paro.

Son precisamente estas fallas implícitas las que provocan que no sea posible simular la operación de WRITEREAD en el modelo iterado de copia inmediata sin suponer protocolos sin-espera. En la sección 3.2.2 descubrimos que cuando se trabajan con detectores de fallas, esta suposición es demasiado restrictiva. Dado que las fallas de equivalencia eliminan el problema de las fallas implícitas en esta sección trataremos de probar que usando fallas de equivalencia el modelo iterado de copia inmediata es tan poderoso como el de registros atómicos.

Lema 4. *Aún con fallas de equivalencia el algoritmo de convergencia vectorial no garantiza terminación si se quita la suposición de protocolos sin-espera.*

Este lema se sigue de que aún en una ejecución sin fallas implícitas, el algoritmo de convergencia vectorial requiere que un proceso vea solamente un vector único de estimados al ejecutar ISNAPSHOT. Sin embargo, es posible que en una ejecución un proceso p_i siempre sea bloqueado por dos procesos p_1 y p_2 que sean agendados antes de p_i de manera alternante.

A continuación se presenta una modificación al algoritmo de convergencia vectorial que supera las limitaciones del algoritmo original. La modificación consiste en dos arreglos compartidos adicionales, el arreglo ${W_i}^P$ y el ${F_i}^P$. Estas dos arreglos solo son modificados en la línea 1, y siempre contienen el valor de los arreglos W_i y F_i resultado de la última llamada a *convergencia vectorial con cooperación*. Además se agregaron las líneas 7-9, donde hay otro punto de salida del algoritmo.

Con excepción del nuevo punto de salida en la línea 9, el resto del algoritmo queda intacto, por lo tanto para comprobar su correctez solamente es necesario probar que

Algoritmo 3 Convergencia vectorial con cooperación en IIS para el proceso p_i

1: **function** AIDEDCONVERGINGVECTOR(v_i)
2: $W_i{}^P \leftarrow W_i, F_i{}^P \leftarrow F_i$
3: $W_i[i] \leftarrow v_i$
4: $F_i[i] \leftarrow F_i[i] + 1$
5: **repeat**
6: $S_i \leftarrow M_{r_i}.ISnapshot(< F_i, W_i, F_i{}^P, W_i{}^P >), r_i \leftarrow r_i + 1$
7: **if** $\exists F_j{}^P \in S_i.F_j^P[i] = F_i[i]$ **then**
8: $< F_i, W_i > \leftarrow < F_j{}^P, W_j{}^P >$
9: **return** W_i
10: **for** $j \in 1 \ldots n$ **do**
11: $k \leftarrow argmax\{F_k[j] | k \in \{1, \ldots, n\}\}$
12: $< F_i[j], W_i[j] > \leftarrow < F_k[j], W_k[j] >$
13: **until** $\forall < F, - > \in S_i, F = F_i$
14: **return** W_i
15: **end function**

cualquier vector devuelto en la línea 9 cumpla con la propiedad de auto-contención y linearización. Sin embargo, cualquier vector devuelto en la línea 9, fue escrito por algún proceso p_j como ${W_j}^P$ al ejecutar ISNAPSHOT y como ${W_i}^P$ contiene el resultado de la última llamada convergencia vectorial con cooperación, se sigue que este vector ya había sido devuelto por algún proceso en la línea 14. Por lo tanto, todos los vectores devueltos en la línea 9 ya habían sido devueltos en la línea 14, y las pruebas de correctez de la sección 3.1.2 garantizan la correctez de estos vectores.

Hasta ahora solamente hemos probado que las modificaciones al algoritmo no afectan su correctez, sin embargo no hemos analizado que ventajas tiene sobre el algoritmo de convergencia vectorial original. En particular, cuando se utiliza la definición de fallas de equivalencia este algoritmo garantiza la terminación de todo proceso correcto (sin necesidad de suponer protocolos sin-espera).

Lema 5. *Si existe al menos un proceso correcto bajo la definición tradicional de fallas entonces tambien existe al menos un proceso correcto bajo la definición de fallas de equivalencia.*

Demostración Por definición, los procesos correctos bajo la definición de fallas de equivalencia son aquellos que pertenecen a la clase $C_{\text{mín}}$. La clase $C_{\text{mín}}$ es la clase más pequeña de todas. Por lo tanto, si existe al menos un proceso correcto bajo la definición tradicional de fallas, entonces existe al menos una clase de equivalencia, y de esto se sigue que la clase $C_{\text{mín}}$ esta bien definida para ejecuciones con al menos un proceso correcto bajo la definición de fallas. ■

Del lema anterior se sigue que cuando no todos los procesos son incorrectos bajo la definición tradicional de fallas, entonces bajo la definición de fallas de equivalencia existe al menos un proceso correcto. Esto lo utilizaremos en el teorema siguiente, ya que

supondremos que existe al menos un proceso correcto, lo cual es una suposición débil ya que los sistemas donde todos los procesos son incorrectos no son de interés práctico.

Teorema 5. *Toda invocación de convergencia vectorial con cooperación por un proceso correcto p_i devuelve un vector W_i.*

Demostración Supongamos que existe un proceso correcto p_i que ejecuta el algoritmo de convergencia vectorial con cooperación con valor v_i en el tiempo t y el algoritmo nunca regresa un vector.

Eso implica que la condición en la línea 13 nunca se cumple y por lo tanto el proceso p_i ve un número infinito de vectores distintos. Por lo tanto existe al menos un proceso que termina un número infinito de llamadas al algoritmo de convergencia vectorial con cooperación. Más aun, la definición de correctez garantiza que p_i es visto un número infinito de veces por cualquier proceso que ejecute un número infinito de pasos.

Definamos a Q como el conjunto de procesos que terminan un número infinito de llamadas al algoritmo de convergencia vectorial con cooperación y por lo tanto ven a p_i un número infinito de veces.

Existe un tiempo $t' > t$ en el que para todo proceso $p_j \in Q$ se cumple que ${W_j}^P[i] = W_i[i]$ y ${F_j}^P[i] = F_i[i]$ (puesto que ven al proceso p_i y terminan las llamadas al algoritmo de convergencia vectorial con cooperación.

Sin embargo, eso implica que existe un tiempo $\tau > t'$ tal que p_i ve un vector ${F_j}^P[i] = F_i[i]$, pero cuando esto ocurre p_i regresa un vector en la línea 9, y esto es una contradicción. ■

Por lo tanto hemos probado que bajo la definición de fallas de equivalencia, si existe al menos un proceso correcto, entonces es posible simular cualquier protocolo de registros atómicos en el modelo iterado de copia inmediata. En particular si el protoco-

lo utiliza un módulo de detección de fallas $\mathcal{F}$ también se puede simular en el modelo iterado de copia inmediata utilizando el mismo detector de fallas.

Capítulo 4

Conclusiones

En este trabajo comenzamos por probar que el modelo iterado de copia inmediata no es apropiado para analizar sistemas distribuidos enriquecidos con detectores de fallas no confiables, al menos no cuando se utiliza la definición tradicional de fallas. Las fallas implícitas introducidas en el capítulo dos representan errores dentro de un sistema que se "escapan" a la definición de fallas tradicional. En particular cuando se trata con modelos iterados es posible que debido a la calendarización en la ejecución de un conjunto de procesos existan fallas implícitas en la comunicación. El propósito de las fallas de equivalencia es encapsular tanto a las fallas tradicionales como a las fallas implícitas.

Utilizando el protocolo introducido [7] es posible simular protocolos sin-espera dentro del modelo de registros atómicos utilizando el modelo iterado de copia inmediata. No obstante, se demostró en el último capitulo que es imposible generalizar esta simulación para que funcione en protocolos que no son sin-espera. En el caso del modelo de registros atómicos y el modelo iterado de copia inmediata enriquecidos con detectores de fallas, se describió porque no es posible traducir literalmente el concepto de un protocolo sin-espera al ámbito de un sistema con detección de fallas no confiable.

De estos resultados se desprende el hecho de que no solamente la simulación pro-

puesta en [7] no se puede utilizar en sistemas con detectores de fallas, sino que no existe ningún algoritmo capaz de realizar esta simulación. Este resultado sugiere que el modelo iterativo de copia inmediata no es apropiado para estudiar sistemas enriquecidos con detectores de fallas.

Sin embargo, si se repite el mismo análisis utilizando la definición de fallas de equivalencia los resultados son más prometedores. Al utilizar las fallas de equivalencia desaparecen las fallas implícitas que aparecían en el modelo iterado de copia inmediata. La ausencia de fallas implícitas nos permite suponer ciertas cosas sobre la ejecución de procesos correctos, por ejemplo que en una ejecución infinita todo par de procesos correctos pueden enviar y recibir información infinita.

Finalmente, desarrollamos un algoritmo de convergencia vectorial con cooperación que permite simular ejecuciones arbitrarias de operaciones del modelo de registros atómicos. Utilizando este algoritmo se pueden definir las condiciones necesarias para simular un modelo dentro de otro y así obtener la equivalencia entre el modelo de registros atómicos y el modelo iterativo de copia inmediata. Más aun, esta equivalencia es más general que la presentada en [7] ya que no exige que los algoritmos sean sin-espera, solo es necesario utilizar la definición de fallas de equivalencia y suponer ejecuciones donde existe al menos un proceso correcto.

Por lo tanto, en instancias donde sea posible remplazar la definición de fallas tradicionales por la definición de fallas de equivalencia, y las ejecuciones contengan al menos un proceso correcto, el modelo iterado de copia inmediata es apropiado para estudiar sistemas enriquecidos con detectores de fallas no confiables. Para traducir cualquier resultado obtenido dentro del modelo iterado de copia inmediata al modelo de registros atómicos es suficiente utilizar un módulo de detección de fallas en el modelo de registros atómicos que encapsule las fallas implícitas del algoritmo del conjunto

participante.

4.1. Problemas abiertos y trabajo futuro

En el capítulo tres se probó que utilizando la definición de fallas de equivalencia es posible implementar algoritmos capaces de simular el modelo de registros atómicos dentro del modelo iterado de copia inmediata. Esto se debe en parte a que las fallas de equivalencia encapsulan tanto las fallas de tipo paro como las fallas implícitas en el modelo iterado de copia inmediata. También se probo que las fallas de equivalencia son idénticas a las fallas de tipo paro en el modelo de registros atómicos. Resulta natural preguntar si existe alguna definición de fallas que permita simular WRITEREAD en el modelo iterado de copia inmediata pero que no incluya a las fallas de tipo equivalencia. Es decir, ¿Cual es la definición de fallas más débil bajo la cual ambos modelos son equivalentes?. La conjetura del autor es que las fallas de equivalencia son las fallas más débiles que permiten la equivalencia de estos dos modelos.

Por otro lado, una de las razones por las cuales el modelo iterado de copia inmediata es relevante en sistemas distribuidos es porque su naturaleza iterativa produce una estructura recursiva que facilita el análisis de protocolos utilizando herramientas topológicas [7]. Sería interesante utilizar estas herramientas para ayudar a explicar los que cambios ocurren en un sistema distribuido cuando se introduce un detector de fallas. Esto permitiría caracterizar de manera elegante cual es el detector de fallas más débil para resolver una tarea dada.

Utilizando los resultados presentados en este trabajo es posible utilizar el modelo iterado de copia inmediata enriquecido con un módulo de detección de fallas $\mathcal{F}$ bajo la definición de fallas de equivalencia para estudiar las propiedades del sistema desde un punto de vista topológico. Dados dos módulos de detección de fallas $\mathcal{F}$ y $\mathcal{F}'$, donde $\mathcal{F}$

es más débil que $\mathcal{F}'$, sería interesante examinar la diferencia en la estructura topológica inducida por ambos detectores de fallas dentro del modelo iterativo de copia inmediata.

Bibliografía

[1] Cyntha Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial syncrhony. *Journal of the ACM*, 35(2):288–323, April 1988.

[2] Michael Fisher, Nancy Lynch, and Michael Paterson. Impossibility of Distributed Consensus with one Faulty Process. *Journal of the ACM*, 43(2):225–267, 1985.

[3] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Compuation*, 105:132–158, July 1993.

[4] Michael Saks and Fotios Zaharoglou. Wait-Free k-set agreement is Impossible: The Topology of Public Knowledge. *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pages 101–110, 1993.

[5] Maurice Herlihy and Nir Shavit. The Asynchronous Computability Theorem for t-Resilient Tasks. *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pages 111–120, 1993.

[6] Elizabeth Borowsky and Eli Gafni. Generalized FLP Impossibility Result for t-Resilient Asynchronous Computations. *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pages 91–100, 1993.

[7] Elizabeth Borowski and Eli Gafni. A Simple Algorithmically Reasoned Characterization of Wait-free Computation. *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing*, pages 189–198, 1997.

[8] Tushar Deepak Chandra and Sam Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43:675–722, 1996.

[9] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43:675–722, 1996.

[10] Alejandro Cornejo, Sergio Rajsbaum, Michel Raynal, and Cornetin Travers. Failure Detectors are Schedulers (Brief Announcement). *Proceedings of the 26th ACM Symposium on Principles of Distributed Computing*, August 2007.

[11] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merrit, and Nir Shavit. Atomic Snapshots of Shared Memory. *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, pages 1–13, 1990.

[12] Elizabeth Borowski and Eli Gafni. Immediate Atomic Snapshots and Fast Renaming. *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing*, pages 41–51, 1993.

[13] Rachid Guerraoui and Michel Raynal. The Alpha of Indulgent Consensus. *The Computer Journal*, 50(1):53–67, January 2007.

[14] Rachid Guerraoui and Michel Raynal. The information structure of indulgent consensus. *IEEE Transactions on Computers*, 53(4):453–466, May 2004.

[15] Leslie Lamport. On interprocess communication. *Distributed Computing*, 1:77–101, 1986.

[16] Ambuj Singh, James Anderson, and Mohamed Gouda. The Elusive Atomic Register. *Journal of the ACM*, 41(2):311–339, 1994.

[17] Hagit Attiya and Ophir Rachman. Atomic Snapshots in $O(nlogn)$ Operations. *Proceedings of the 12th ACM Symposium of Principles of Distributed Computing*, pages 29–40, 1993.

[18] Leslie Lamport, Robert Shostak, and Marshal Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[19] Vassos Hazdilacos. Issues of Fault Tolerance in Concurrent Computations. *PhD thesis, Harvard*, 1984.

[20] Kenneth Perry and Sam Toueg. Distributed Agreement in the Presence of Processor and Communication Faults. *IEEE. Transactions on Software Engineering*, 12(3):477–482, 1986.

[21] Leslie Lamport and Michael Fischer. Byzantine generals and transaction commit protocols. *Technical Report 62, SRI International*, April 1982.

[22] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.

[23] Maurice Herlihy. Impossibility and Universality Results for Wait-Free Syncrhonization. *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing*, pages 276–290, 1988.

[24] Rachid Gerraoui and Michel Raynal. The Alpha and Omega of Asynchronous Consensus. *Technical Report PI-1676, IRISA*, 2005.

[25] Wai-Kau Lo and Vassos Hadzilacos. Using Failure Detectors to Solve Consensus in Asynchronous Shared-Memory Systems. *Proceedings of the 8th International Workshop on Distributed Algorithms*, pages 280–295, 1994.

[26] James R. Munkres. *Topology*. Prentice Hall, 2nd edition, 2000.