Diseño de un middleware tolerante a fallas basado en el protocolo Paxos

Ricardo Adán Madrid Trejo
Universidad Autónoma Metropolitana-Iztapalapa, San Rafael Atlixco No. 186, Col. Vicentina, Iztapalapa, C.P. 09340, México D.F., Teléfono: 01 55 5804 4600
richard.mt17@gmail.com

Ricardo Marcelín Jiménez
Universidad Autónoma Metropolitana-Iztapalapa, San Rafael Atlixco No. 186, Col. Vicentina, Iztapalapa, C.P. 09340, México D.F., Teléfono: 01 55 5804 4600
calu@xanum.uam.mx

Orlando Muñoz Texzocotetla
Universidad Autónoma Metropolitana-Iztapalapa, San Rafael Atlixco No. 186, Col. Vicentina, Iztapalapa, C.P. 09340, México D.F., Teléfono: 01 55 5804 4600
magicorlan@gmail.com

Resumen

El consenso, de manera genérica, consiste en un conjunto de procesos tales que cada uno registra algún valor de entrada y, basados en estos valores, deben coincidir en un valor de salida. Paxos es un algoritmo que resuelve el problema del consenso para el manejo consistente de registros duplicados en una red de procesos expuestos al riesgo de fallas de paro y recuperación. Es utilizado en la construcción de sistemas con requerimientos de alta disponibilidad. En este artículo se describe el sistema de archivos Babel y la experiencia al implementar Paxos para la administración de metadatos en dicho sistema. El énfasis de la exposición está puesto en la metodología con la que se probó la implementación, los problemas que se detectaron que hacen

imposible ejecutar correctamente el protocolo Paxos, así como y las soluciones que se propone para estos escenarios particulares.

Palabra(s) Clave(s): consenso, consistencia, metadatos, paxos.

1. Introducción

El sistema de archivos Babel, es un sistema de almacenamiento masivo de información con garantías de alta disponibilidad y escalabilidad. Se compone de un conjunto de máquinas con capacidades de almacenamiento y procesamiento conectadas mediante una red local. Los clientes de Babel perciben una sola máquina, denominada coordinador o *proxy*, que despacha las solicitudes de servicio (almacenamiento, búsqueda y recuperación de archivos) y administra los recursos (ver Fig. 1).

Los archivos que se reciben desde los clientes de Babel se guardan de manera redundante creando un exceso en la información que codifica a los archivos, y este exceso se guarda de forma distribuida entre los dispositivos de almacenamiento de Babel. Se trata de una solución que puede articular un número masivo de dispositivos y presentarlos bajo una interfaz única.

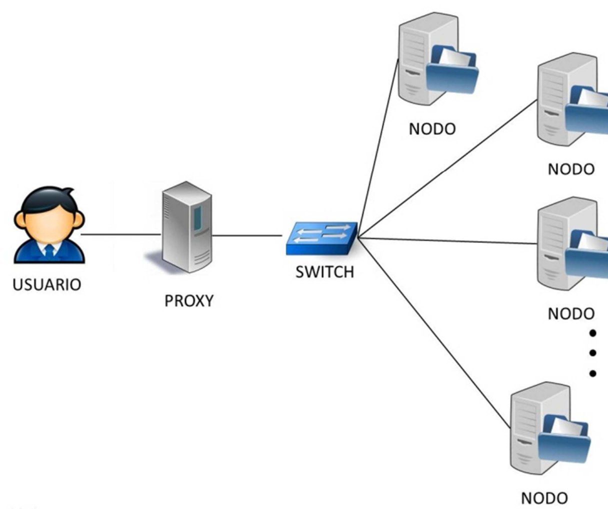

Fig. 1. Funcionamiento del sistema Babel.

1.1. Problemática

El *proxy* almacena los metadatos que hacen posible la recuperación de la información de los clientes que utilizan Babel. Actualmente se tiene un solo *proxy*, si éste fuera llevado al límite de sus capacidades por un exceso de peticiones podría crear un cuello de botella, asimismo sería el punto más débil del sistema si quedara fuera de servicio, entonces podría limitarse severamente la posibilidad de recuperar cualquier archivo almacenado al interior del sistema. Para evitar cualquiera de estas contingencias se propuso implementar un conjunto redundante de *proxies* que puedan garantizar la consistencia de la información que gestionan, haciendo que el sistema tenga alta disponibilidad.

Al contar con un conjunto redundante de proxies se solucionan los dos problemas mencionados anteriormente, sin embargo, en este nuevo escenario es necesario garantizar la consistencia de los metadatos, como se explica a continuación. Supóngase, por ejemplo, un usuario que es atendido por un proxy A, a través del cual almacena un archivo X. Los metadatos necesarios para recuperar X quedan registrados en A. Sin embargo, cuando el usuario regresa al sistema para recuperar su información, se le asigna un proxy B, para atender su nueva solicitud. Evidentemente, B debe disponer de una copia de los metadatos inicialmente registrados en A, de manera que pueda recuperar la información del usuario.

Lo anterior significa que se debe garantizar que cualquier metadato registrado por un proxy quede registrado en todos los demás pares en servicio, aun bajo ciertas restricciones en la operación de los participantes. La condición anterior describe un problema del cómputo distribuido conocido como el problema de consenso. La solución, si existe, debe exhibir tres propiedades: i) validez, ii) acuerdo, y iii) terminación. La primera indica que el valor de salida debe ser alguno de los valores inicialmente registrados. La segunda expresa el requisito de coincidencia. La tercera, requiere que la solución se alcance al cabo de un tiempo finito.

El conocido resultado de Fischer, Lynch y Patterson [1], demuestra que el consenso es imposible en un sistema totalmente asíncrono si, al menos, uno de los participantes puede experimentar una falla de paro a menos que se ofrezca un conjunto mínimo de garantías sobre el tiempo, el orden de las comunicaciones o el procesamiento [2,3].

1.2. Propuesta

Se implementó un mecanismo para garantizar la consistencia de los metadatos replicados en cada proxy. El objetivo de este trabajo es el desarrollo de un prototipo para garantizar la consistencia de las copias de una base de datos (BDD), que están a cargo de un conjunto redundante de servidores, usando para ello el protocolo Paxos, propuesto por Lamport [4]. Puede entenderse como la construcción de un mecanismo de comunicación (o middleware), tolerante a fallas, a través del cual los servidores se comunican para garantizar que cualquier cambio en una instancia de la BDD queda registrado en todas las demás copias. Se sabe que, bajo ciertas condiciones de operación, Paxos garantiza la solución del consenso.

La contribución de este trabajo consiste en usar la simulación de eventos discretos para reproducir diferentes escenarios de operación sobre los que puede desplegarse el protocolo Paxos. Gracias a esta metodología se pudo reconocer condiciones que en el trabajo original se asumen resueltas, o se dejan en manos del equipo a cargo de la implementación. Otra ventaja de este enfoque de construcción es que se plantea un módulo a cargo del protocolo, que puede exportarse con facilidad hacia un entorno de producción y reutilizarse en escenarios semejantes. Se presenta una instancia de Babel, con un conjunto redundante de proxies (Ver Fig. 2).

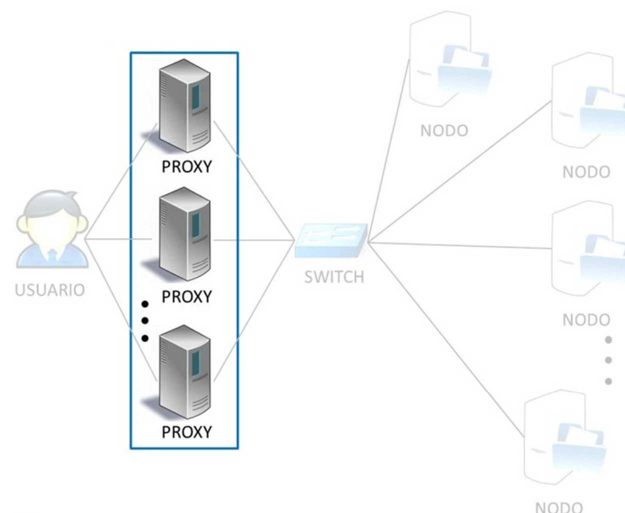

Fig. 2. Lugar donde se produce el problema de la consistencia de los metadatos.

El resto de este artículo incluye las siguientes partes o secciones. En la sección 2 se explica el algoritmo Paxos, los supuestos de operación con los cuales trabaja, sus propiedades y el funcionamiento de una instancia del algoritmo. En la sección 3 se describen varios escenarios de ejecución donde se detectaron las condiciones que imposibilitan resolver el problema del consenso, así como los mecanismos para solucionarlos. Finalmente en la sección 4 se detallan las conclusiones.

2. El protocolo paxos

El protocolo Paxos fue publicado por primera vez en 1989, el tono de la presentación es un divertimento que se aleja del estilo tradicional con el que se presentan los artículos de computación. Algunos investigadores consideraron que este tono retrasó su valoración por parte de la comunidad de los sistemas distribuidos. Fue reimpreso más tarde, con cambios menores, como un artículo de revista en 1998 [4]. Finalmente, el propio autor escribió una versión desprovista de toda alegoría, cuya lectura es relativamente más fácil y es un mejor punto de partida para embarcarse en su estudio [5].

La propuesta original de Paxos incluye el llamado protocolo múltiple o multi-Paxos, en el que se describe cómo puede extenderse el procedimiento básico para manejar de manera concurrente varias instancias del problema de consenso. A lo largo del tiempo, varios autores han hecho sus propias versiones modificando el algoritmo original para diferentes entornos. Con este hecho, Paxos se ha convertido en una familia de protocolos para la solución de consenso en una red de procesos susceptibles de experimentar varios tipos de falla.

En un reciente trabajo [6], se muestra un enfoque basado en Paxos para la construcción de un servicio de gestión de metadatos en sistemas de archivos distribuidos, alcanzando una alta disponibilidad sin incurrir en una penalización de rendimiento.

Recientemente, se publicó un nuevo algoritmo de consenso llamado Raft [7], que es similar en muchos aspectos a Paxos. Sin embargo, Paxos ha dominado la discusión de los algoritmos de consenso durante la última década y la mayoría de las implementaciones de consenso se basan en Paxos. Los autores de Raft hacen énfasis en que Paxos es bastante difícil de comprender, y es por eso que definieron un algoritmo de consenso para los sistemas prácticos describiéndolo de una manera que es mucho más fácil de aprender que Paxos.

2.1. Supuestos de operación

Para lograr el correcto funcionamiento del algoritmo Paxos se requiere el cumplimiento de los siguientes supuestos de operación. En muchos de estos, se trata de condiciones límite más allá de las cuales no se garantizan los resultados del protocolo.

Sobre los procesos

- Los procesos operan a una velocidad arbitraria.
- Los procesos pueden experimentar fallas de paro con su posible recuperación posterior.

- Los procesos disponen de un almacenamiento estable que les permite reconocer su historia y reintegrarse al protocolo después de recuperarse de su fallo.
- Los procesos no confabulan, mienten o hacen cualquier otro intento de desestabilizar el protocolo, es decir, los fallos bizantinos no se producen.

Sobre las comunicaciones

- Los procesos envían mensajes a cualquier otro proceso.
- Los mensajes se envían de forma asíncrona y pueden tomar un tiempo arbitrariamente largo de entrega.
- Los mensajes enviados no se pierden.
- Los mensajes se entregan sin daños.

2.2. Roles

Paxos describe las acciones de los procesos por sus roles en el protocolo. Estos son los siguientes: el cliente, el aceptante, el proponente, el aprendiz y el líder. En las implementaciones típicas, un mismo proceso puede desempeñar uno o varios de estos roles sin detrimento del protocolo.

- **Cliente:** El cliente envía una petición al sistema distribuido para alcanzar el consenso sobre un valor en el que está interesado y espera una respuesta.
- **Aceptantes:** Los aceptantes se reúnen en grupos llamados cuorums. Un cuórum se define como un subconjunto de los aceptantes, que expresa la característica de seguridad de Paxos. Ello significa que en un cuórum existe al menos algún proceso sobreviviente que conserva el conocimiento de los resultados previos, esto es porque ha participado en anteriores rondas del protocolo. Por lo anterior se prefiere que el número de participantes sea impar. Una manera muy sencilla de formar un cuórum es por mayoría simple.

- **Líder:** Un líder aboga por la petición de un cliente tratando de convencer a los aceptantes de llegar a un acuerdo sobre un valor y actúa como un coordinador para evitar que el sistema se estanque en caso de conflictos.
- **Aprendices:** Una vez que una solicitud del cliente ha sido acordada por los aceptantes, el aprendiz puede tomar una acción en beneficio del cliente inicial (es decir, cumplir la solicitud y enviar una respuesta al cliente).

2.3. Propiedades del protocolo Paxos

Paxos define tres propiedades y garantiza que siempre se llevan a cabo, independientemente del patrón de fallas:

- **No trivialidad.** Solo los valores propuestos se pueden aprender (es decir, registrarlo en cada proceso).
- **Seguridad (Safety).** A lo más un valor se puede aprender.
- **Vitalidad (Liveness).** Si un líder vive lo suficiente, entonces es capaz de sacar adelante la solución de consenso.

2.4. Despliegue típico del protocolo

El protocolo básico es el más sencillo de la familia Paxos. Cada instancia del protocolo Paxos básico decide sobre un único valor de salida. El protocolo procede en varias rondas. Una ronda exitosa tiene dos fases. Un líder no debe iniciar Paxos si no se puede comunicar con al menos un cuórum de los aceptantes. A continuación se describen las dos fases por las que transita el protocolo Paxos.

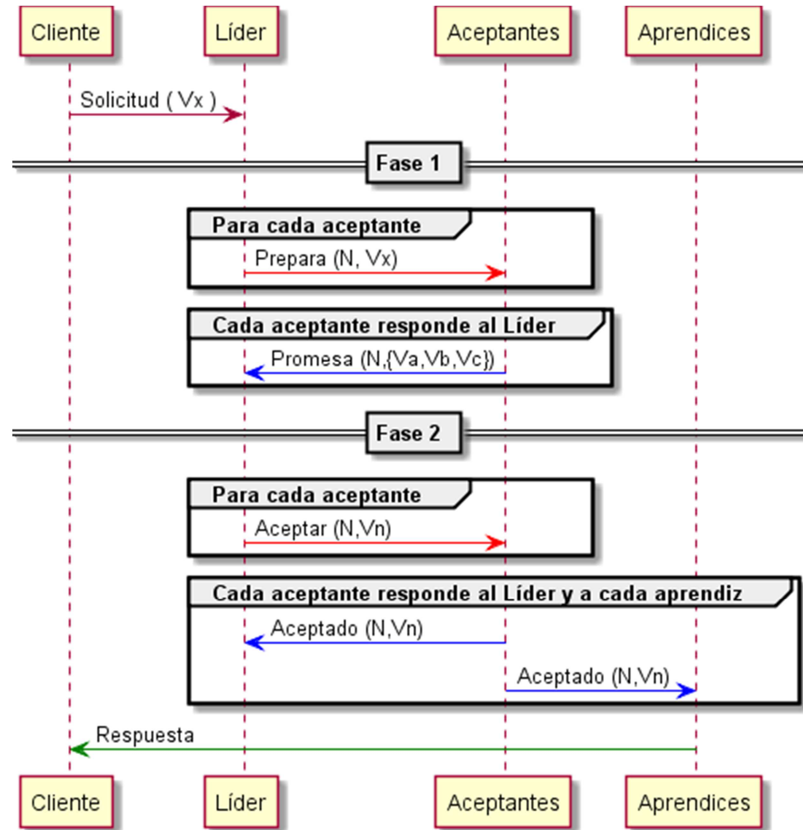

Fig. 3. Paxos básico. Primera ronda es exitosa.

- **Fase 1_a Prepara.** El líder crea una propuesta identificada con un número de ronda *N*. Este debe ser mayor que cualquier número de propuesta del que tenga conocimiento. Después, envía un mensaje *PREPARA* a todos los participantes, conteniendo el número de ronda. Esto comienza en la primera fase del protocolo. (Ver Fig. 3).
- **Fase 1_b Promesa.** Si el número de la propuesta *N* es mayor que cualquier número recibido previamente, entonces el aceptante envía un mensaje *PROMESA* al líder y con ello se compromete a ignorar todas las propuestas que tienen un número menor que *N*. Si el aceptante aceptó una propuesta en algún momento en el pasado, debe incluir el número de la propuesta anterior y el valor anterior en su respuesta al líder. En caso contrario, si el número de propuesta es menor que otro con el que ya se comprometió, el aceptante puede ignorar la

propuesta recibida. No tiene que responder en este caso para que funcione Paxos. Sin embargo, en bien de la economía de mensajes, el envío de una respuesta negativa (Nack) le diría al líder que puede poner fin a su intento de crear un consenso con la propuesta *N*. Esto es el acuse de respuesta de la fase 1. (Ver Fig. 3).

- **Fase 2_a: Aceptar.** Si un líder recibe suficientes promesas de un cuórum de los aceptantes, es necesario que asocie un valor con su propuesta. Si alguno de los aceptantes había aceptado previamente cualquier propuesta, entonces ellos han enviado sus valores al líder, que ahora debe igualar el valor de su propuesta con el valor más reciente reportado por algún aceptante. Es decir, el valor asociado con el mayor número de propuesta reportado en un mensaje *PROMESA*, recién recibido. Si ninguno de los aceptantes había aceptado una propuesta hasta este punto, entonces el líder puede elegir cualquier valor para su propuesta. El líder envía un mensaje *ACEPTAR* a un cuórum de aceptantes con el valor elegido para su propuesta. Se comienza con la fase 2 del protocolo. (Ver Fig. 3).
- **Fase 2_b: Aceptado.** Si un aceptante recibe un mensaje *ACEPTAR* de la propuesta *N*, debe aceptarlo si y sólo si aún no ha prometido tener en cuenta sólo las propuestas que tengan un identificador mayor que *N*, es decir si aún no ha respondido a ningún mensaje *PREPARA* (enviar un mensaje *PROMESA*). En este caso, se debe registrar el valor correspondiente y enviar un mensaje *ACEPTADO* para el líder y cada aprendiz. Si no, se puede ignorar el mensaje *ACEPTAR*. Esto es el acuse de respuesta de la fase 2. (Ver Fig. 3).

Si el líder es relativamente estable, la fase 1 se convierte en innecesaria. Por lo tanto, es posible omitir la fase 1 para futuras instancias del protocolo con el mismo líder.

3. Escenarios críticos

Basados en multi-Paxos se construyó una descripción usando un autómata de estados finitos [8], con el que programamos una versión escrita en el lenguaje Python. La

simbología de esta descripción es la siguiente: se tienen estados representados por un rectángulo redondeado, la recepción de un mensaje representado por rectángulo con uno de sus extremos en forma de flecha, el envío de un mensaje representado por un pentágono y el procesamiento interno representado por un rectángulo (Ver Fig. 4). Luego, como parte del proceso de desarrollo, se decidió someter la implementación a una serie de pruebas de ingeniería. Considerando los alcances y los costos de estas pruebas se prefirió "incrustar" el código dentro de un simulador de eventos discretos.

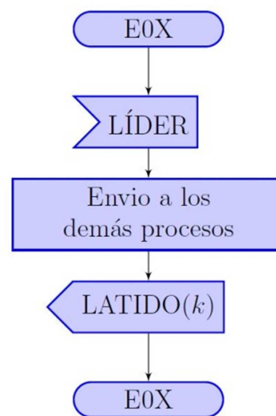

Fig. 4. Fragmento del autómata que describe al protocolo multipaxos.

Con ello se consiguió un control más fino sobre las diferentes condiciones de prueba, tales como el número de participantes, las tasas de falla y recuperación, entre otros factores. A partir de la agenda de experimentos se pudo descubrir que la implementación cumplía cabalmente con las especificaciones del protocolo, bajo condiciones estables. También se pudieron detectar una serie de condiciones de inestabilidad en las que se pierde la garantía de progreso del protocolo.

El trabajo original no menciona las contingencias que pueden llevar a estos casos, y mucho menos la manera como pueden resolverse. Se considera que la aportación de este trabajo radica en el hecho de proponer los mecanismos para detectar las situaciones de inestabilidad que se reconocieron, así como los procedimientos para llevar de nuevo al sistema hasta un estado donde se garantice el progreso.

En esta sección se describen los problemas encontrados al someter el prototipo a una agenda de pruebas asumiendo tasas aleatorias de paro y recuperación, sobre cada uno de los procesos participantes. Se reconoció que la clave para el funcionamiento correcto de Paxos es que siempre exista exactamente un proceso distinguido o líder, sobre el que recae la responsabilidad de administrar las etapas del protocolo, para cada instancia del consenso.

Desde la perspectiva de la implementación, se identificó los siguientes momentos críticos del protocolo:

- **Elección del líder:** un nuevo líder debe ser elegido al iniciar las operaciones o cuando se presume la ausencia del líder conocido.
- **Duplicación de funciones:** si existiera más de un líder, no se puede avanzar en el protocolo, un nuevo proceso de elección debe arrancarse para superar esta condición.
- **Asignación del número de instancia:** cada instancia del consenso debe recibir un número diferente que la identifica de forma única. Debe existir un mecanismo para repartir los posibles números de instancia entre los procesos que fungirán como clientes. Evidentemente, el conjunto de números gestionados por un cliente debe ser ajeno al conjunto gestionado por cualquier otro cliente.
- **Consistencia y actualización de los registros:** todos los participantes deben tener las mismas entradas en sus registros, o un valor nulo, en caso de ignorar el acuerdo alcanzado para una instancia. En esta última circunstancia deberá actualizarse invocando los servicios del líder.
- **Solicitudes pendientes:** el líder en funciones puede experimentar una falla durante una instancia del protocolo Paxos sin posibilidad de llegar a un consenso sobre la solicitud que recibe.
- **Falta de cuórum:** El líder arranca una instancia del protocolo Paxos, sin embargo no recibe respuestas suficientes para asumir la existencia de un cuórum, imposibilitando con ello el avance del consenso.

Para los fines del estudio, se asume que cada proceso a cargo del protocolo se encuentra asignado en un nodo diferente de la red de comunicaciones subyacente. Por ello, en lo sucesivo usaremos como sinónimos los términos “nodo” y “proceso”.

3.1. Elección de líder

Cuando el sistema inicia sus operaciones por primera vez, los nodos disparan la elección de líder. Una vez electo, el líder envía periódicamente latidos (heartbeats) a todos los aceptantes, con el fin de confirmar su disponibilidad. Si un aceptante no recibiera esta señal dentro de un período de tiempo, entonces asume que no hay un líder en funciones y comienza nuevamente el procedimiento de elección para reemplazar al nodo en este rol (Ver Fig. 5). A lo largo del proceso de elección pueden presentarse alguna de las siguientes situaciones: (a) que pase a la siguiente etapa y se concluya la elección sin contratiempos, (Ver Fig. 6), (b) que expire el plazo para establecer al nuevo líder, o (c) que un nodo caído se recupere en el curso de la elección.

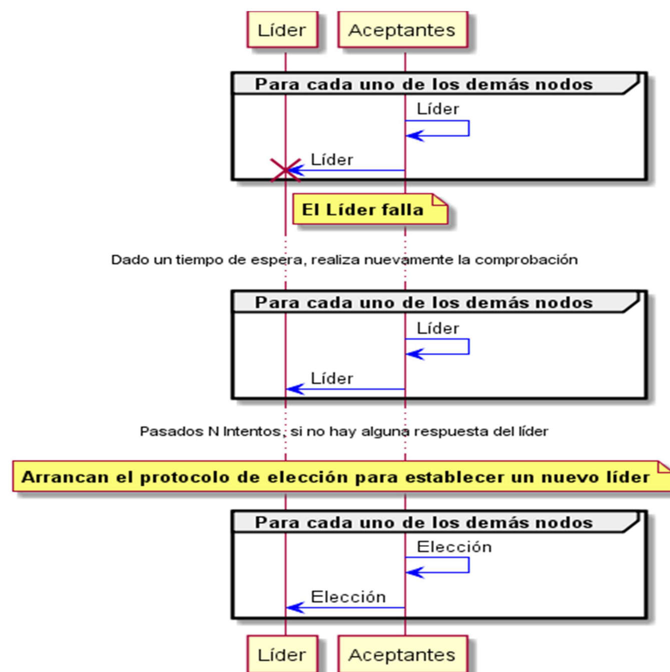

Fig. 5. Los aceptantes inician el protocolo de elección.

3.2. Duplicación de funciones

Se sabe que el protocolo Paxos necesita de un líder único para impulsar el consenso. Sin embargo, es posible que durante lapsos muy breves y luego de una combinación muy particular de eventos, existan dos nodos que asuman este rol al mismo tiempo, imposibilitando con ello el progreso del protocolo. Para evitar este conflicto se introduce un número de secuencia que se incrementa cada vez que se emite un latido. Al recibir este mensaje, los demás nodos registran el número de secuencia. En este escenario se pueden presentarse alguna de las siguientes situaciones: (a) regresa un anterior líder, o (b) ambos líderes en funciones emiten y reciben el mismo número de secuencia de latido. Se muestra el mecanismo para reconocer a un solo líder en funciones. (Ver Fig. 7).

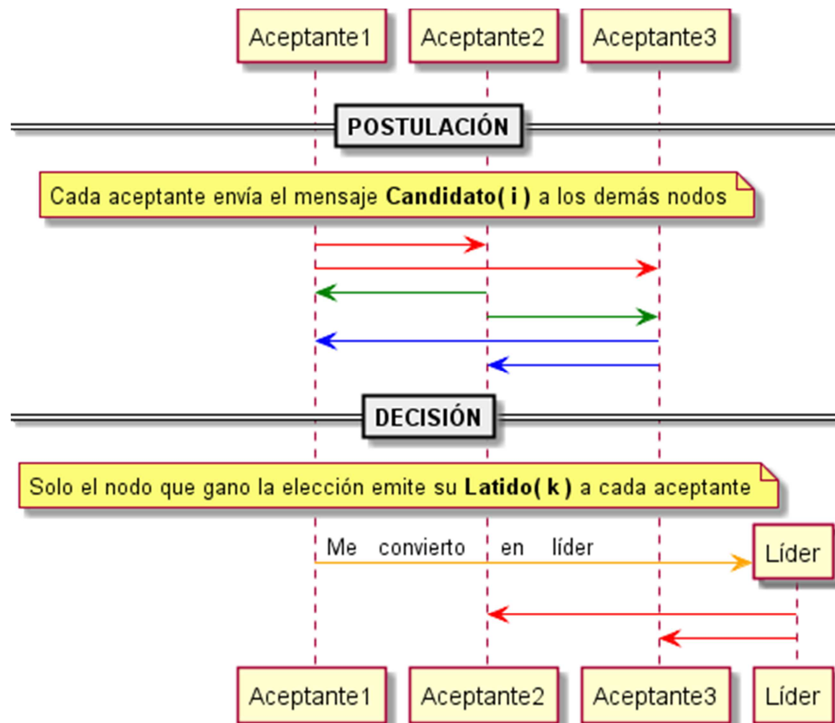

Fig. 6. Procedimiento normal de la elección de un líder.

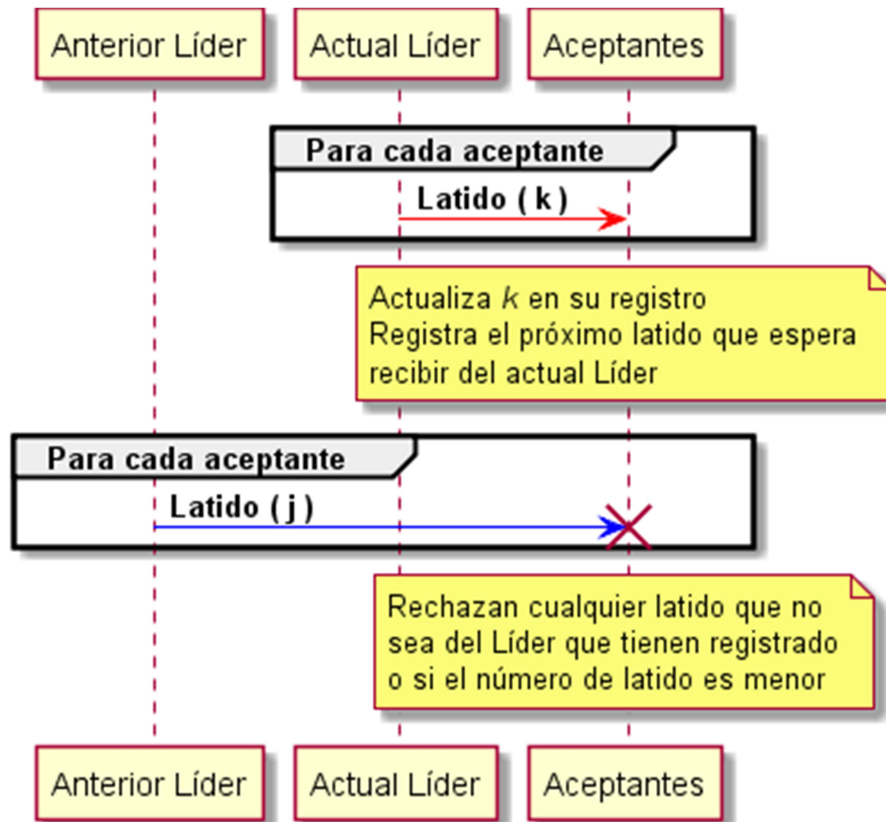

Fig. 7. Los aceptantes solo guardan el latido del líder que tienen registrado.

3.3. Asignación del número de instancia

Se introduce un mecanismo que permite a cada nodo cliente establecer el número de la instancia que él origina (es el dueño). Este mecanismo garantiza que cada número es único y sólo los dueños de la instancia pueden asignarlo. A continuación se describe este mecanismo: se tiene un registro que se incrementa en una unidad cada vez que un cliente establece una solicitud, llamado *vista*, también cada nodo conoce el total de nodos con los que trabaja, al que llamaremos *P*, se sabe que cada nodo tiene un número entero que lo identifica de manera única llamado *ID: 1, … P*. Mediante una sencilla operación matemática se asigna un número de instancia igual a *(vista*P)* + *(ID-1)*. Dicho de otra forma, todos los números de instancia de un mismo dueño son congruentes con su identificador, en módulo *P*.

3.4. Consistencia y actualización de los registros

Luego de recuperarse de una falla, un nodo se reincorpora al protocolo y puede reconocer la presencia de "huecos" en su registro. Esto es, instancias de las que desconoce el valor acordado. El mecanismo utilizado para determinar las instancias faltantes es el siguiente: Se sabe que el aceptante tiene conocimiento del próximo latido del líder. Si el número de latido que recibe coincide con el que espera, se asume que ha estado activo junto con el líder, entonces actualiza este número y continúa su operación normal. Por otro lado, si el número de latido es mayor, se asume que el aceptante estuvo ausente y arranca un procedimiento de actualización. Sin importar si es el dueño o no de una instancia faltante, es decir, instancias en las que no estuvo presente, el nodo debe intentar ponerse al día. Se analizan los siguientes casos: (a) el líder debe actualizar sus registros, (Ver Fig. 8). (b) un aceptante debe mantenerse al día acerca de las instancias que le hacen falta y (c) un aceptante determina qué instancias le hacen falta en sus registros, pero no es dueño de las mismas. Se muestra el mecanismo de actualización para estos dos últimos casos. (Ver Fig. 9).

3.5. Solicitudes pendientes

Este escenario se presenta cuando los clientes someten una solicitud al líder y éste cae en paro. Para solucionar este problema, al momento de que un aceptante envía una solicitud al líder, la almacena temporalmente en una lista de instancias pendientes, que permite conocer que instancias no han sido atendidas. Recordemos que cada nodo maneja sus propias instancias y que tenemos un almacenamiento estable. Cuando el líder le responde a un cliente, el cliente quita la instancia que emitió de la lista de instancias pendientes. Esto pasa también si existiera un nuevo líder, en cuyo caso, al tener un nuevo líder, eventualmente el cliente enviará su solicitud al nuevo líder para que pueda ser atendida.

3.6. Falta de cuórum

Esta situación se presenta cuando el líder se encuentra coordinando una instancia del protocolo pero no existe un cuórum de aceptantes que le responda, es decir, una mayoría simple. Esto se debe a que más de la mitad del conjunto de nodos fallan. En esta circunstancia, el líder arranca un temporizador y queda en espera de recibir respuestas de un cuórum de aceptantes. Si expira este temporizador y no existiera un cuórum para sacar adelante la instancia en progreso, entonces el líder hará caso omiso de la solicitud y continuará en funciones hasta que exista una mayoría y reciba nuevamente la solicitud del dueño de la instancia, para finalmente atenderla y duplicarla consistentemente en cada uno de los demás nodos. Se muestra el mecanismo del temporizador. (Ver Fig. 10).

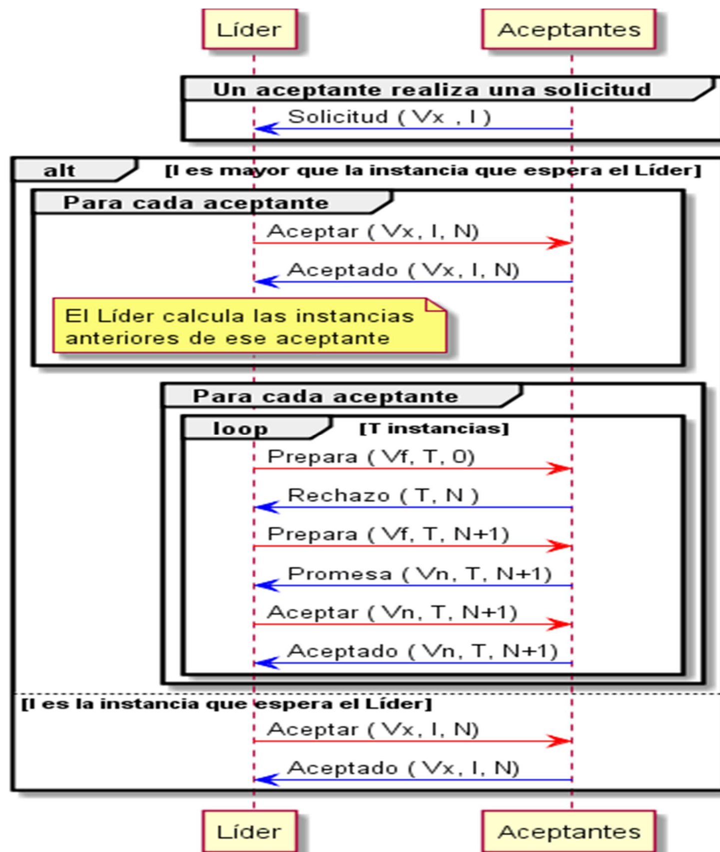

Fig. 8. Consistencia de los registros del líder.

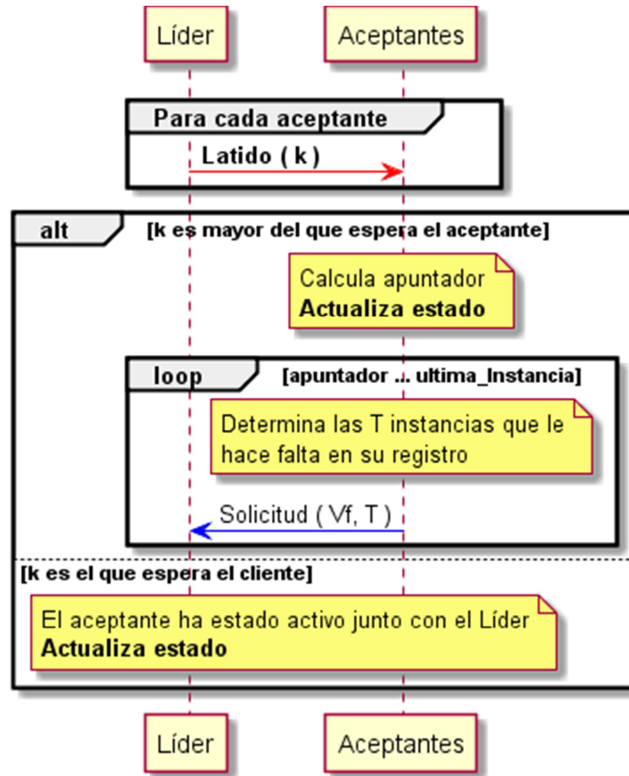

Fig. 9. Un aceptante reconoce si ha estado activo junto con el líder.

4. Conclusión

El algoritmo Paxos permite conseguir que un grupo de máquinas lleguen a un acuerdo sobre un valor. Multi-Paxos es una optimización en el uso de Paxos para manejar varias instancias consecutivas del mismo problema.

Se sabe que, bajo ciertas condiciones de operación, Paxos garantiza la solución de consenso. La contribución de este trabajo radica en que se reconocieron los escenarios en los que dichas condiciones pueden dejar de cumplirse, gracias a que se pudieron inyectar fallas arbitrarias durante la ejecución del protocolo en escenarios que se configuraron a conveniencia.

Asimismo, se desarrollaron los mecanismos que permiten solucionar estos problemas para restablecer las condiciones de operación, que garantizan el progreso del protocolo.

Se sabe demostrar que la fase 2 del algoritmo de consenso Paxos tiene el costo mínimo posible que cualquier otro algoritmo de consenso en presencia de fallas [9]. Por lo tanto, el algoritmo Paxos es esencialmente óptimo.

Paxos es un algoritmo muy robusto, permitiendo que cualquier mayoría simple de servidores progrese de manera segura. Esto hace que sea conveniente para situaciones que requieren alta disponibilidad.

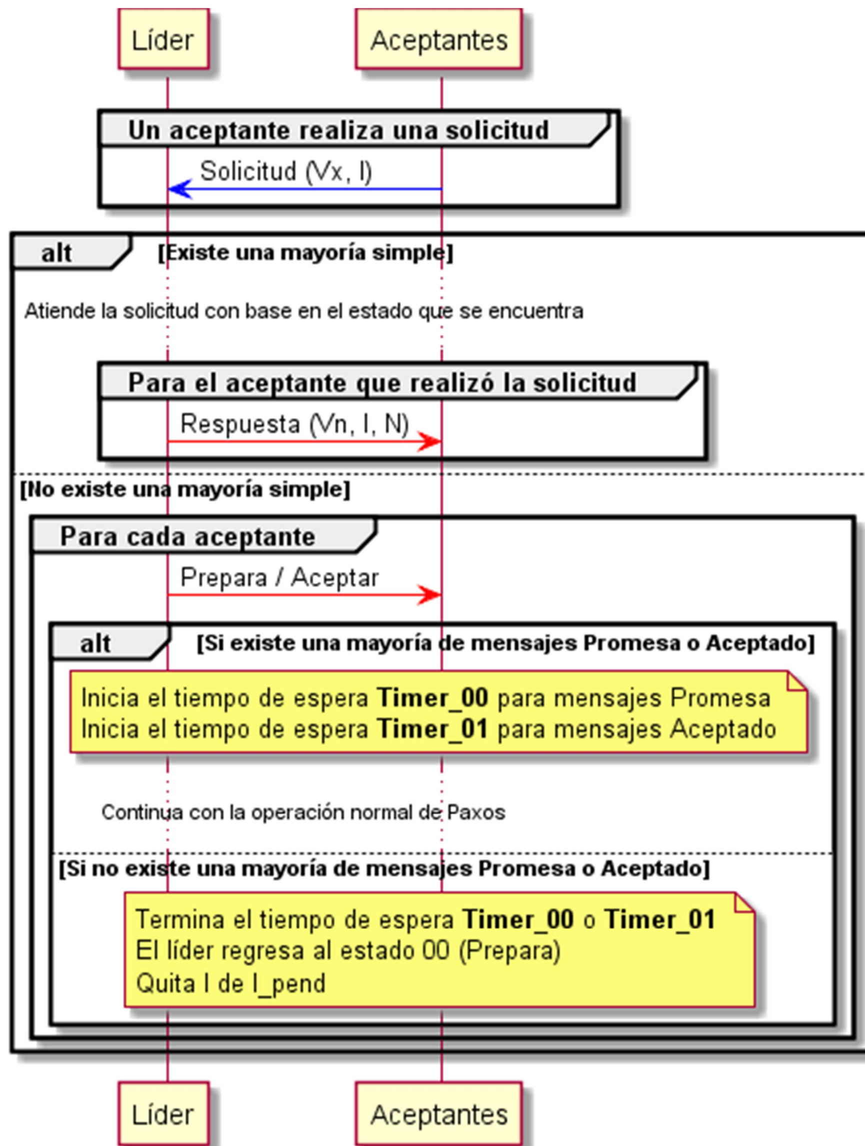

Fig. 10. Temporizador para la recepción de mensajes promesa o aceptado.

5. Referencias

[1] M. J. Fischer, N. A. Lynch, M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process”. *J. ACM*. Vol. 32. No. 2. April 1985. 374-382 pp.

[2] H. Attiya, J. Welch, Distributed Computing: Fundamentals, Simulations and Advanced Topics. 2004. John Wiley & Sons.

[3] T. D. Chandra, S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems”. J. ACM. Vol. 43. No. 2. March 1996. 225-267 pp.

[4] L. Lamport, "The Part-time Parliament”. ACM Trans. Comput. Syst. Vol. 16. No. 2. May 1998. 133-169 pp.

[5] L. Lamport, "Paxos made simple”. ACM SIGACT News. Vol. 32. No. 4. Dec 2001. 51—58 pp.

[6] D. Stamatakis, N. Tsikoudis, O. Smyrnaki, K. Magoutis. “Scalability of replicated metadata services in distributed file systems”. En KarlMichael Göschka y Seif Haridi, editors, Distributed Applications and Interoperable Systems. Vol. 7272 of Lecture Notes in Computer Science. Springer Berlin Heidelberg. 2012. 31-44 pp.

[7] D. Ongaro, J. Ousterhout. “In search of an understandable consensus algorithm”. Proceedings of the 2014 USENIX. Conference on USENIX Annual Technical Conference. Berkeley, CA, USA. 2014. 305-320 pp.

[8] Fred B. Schneider, "Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial”. ACM Comput. Surv. Vol. 22. No. 4. Dec. 1990. 299-319 pp.

[9] Keidar, S. Rajsbaum, “On the cost of fault-tolerant consensus when there are no faults - a tutorial”. In R. de Lemos, T. S. Weber, and J. B. C. Jr., editors. LADC. Vol. 2847 of Lecture Notes in Computer Science. 2003. 366–368 pp.

6. Autores

Ing. Ricardo Adán Madrid Trejo se graduó en 2012 como ingeniero en sistemas computacionales en el Tecnológico de Estudios Superiores de Ecatepec. Actualmente estudia la maestría en ciencias y tecnologías de la información (CyTI) en la Universidad Autónoma Metropolitana Unidad Iztapalapa. Sus intereses de investigación incluyen sistemas paralelos y distribuidos, simulación de eventos discretos, sistemas tolerantes a fallas y computación móvil.

M. en C. Orlando Muñoz Texzocotetla obtuvo el grado de maestría en ciencias y tecnologías de la información (CyTI) en la Universidad Autónoma Metropolitana Unidad Iztapalapa, en 2011. Actualmente estudia el doctorado en CyTI en la misma universidad. Sus intereses de investigación incluyen aprendizaje maquinal, métodos de optimización, minería de datos, big data y computación en la nube.

Ricardo Marcelín Jiménez se graduó en 1987 como ingeniero en electrónica, con especialidad en comunicaciones, por la Universidad Autónoma Metropolitana. En 1992 se graduó de Maestro en Ciencias, con especialidad en computación por el CINVESTAV-IPN. En 2004 consiguió el grado de Doctor en Ciencias, con la especialidad en Computación por parte de la UNAM. Sus intereses de investigación incluyen sistemas paralelos y distribuidos, simulación de eventos discretos, sistemas tolerantes a fallas y computación móvil.