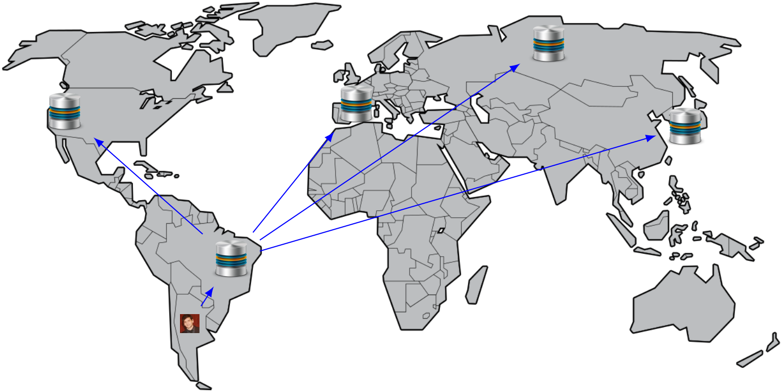
Verificación formal de protocolos distribuidos

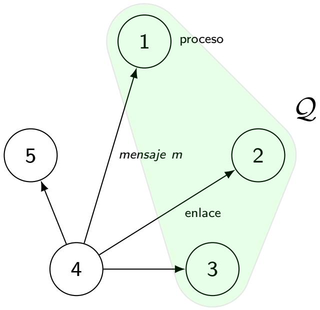
Alejandro Naser Pastoriza

IMDEA Software Institute



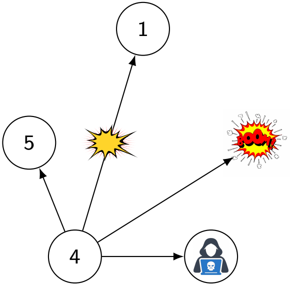
- Alta disponibilidad
- Alta confiabilidad
- Partición de datos
- etc.

Abstrayéndonos



- Objetivo: encapsular formas recurrentes de cooperación en conjunto con sus complejidades inherentes en abstracciones

Desafíos



- Objetivo: dotar a cada proceso con la suficiente información de manera tal que pueda continuar cooperando aún en presencia de fallos

Respecto de la correctitud...

- Las técnicas de testing, en el mejor de los casos, rasguñan la superficie de todos los posibles comportamientos
- Model checking es incompleto y no escala
- Las técnicas automáticas son limitadas por la indecidibilidad del problema subyacente
- Un enfoque más apropiado, entonces, es verificar formalmente las propiedades deseadas a través de pruebas matemáticas

Idea general

Abstracciones:

- Consenso
- Broadcast atómico

Protocolos:

- Single-Decree Paxos
- FLE Atomic Broadcast
- Vertical Paxos

Single Decree Paxos

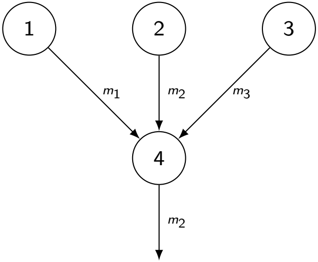
Paxos es un mecanismo para lograr acuerdo en un único valor sobre canales de comunicación no confiables

- Uno o más procesos proponen valores
- A lo más un único valor es elegido

Single Decree Paxos

Primera idea (incorrecta):

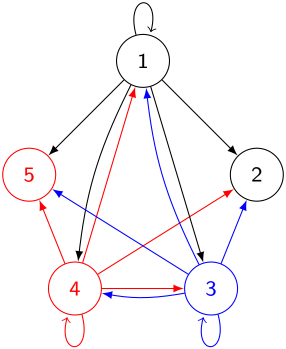
- Un único proceso acepta un valor



Single Decree Paxos

Segunda idea (incorrecta):

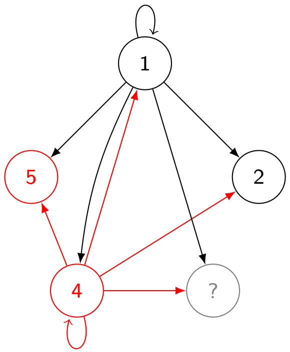
- Aceptar la primera propuesta recibida



Single Decree Paxos

Tercera idea (incorrecta):

- Aceptar múltiples propuestas

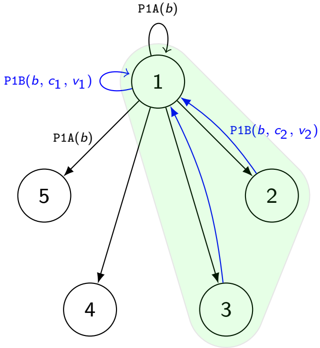


Single Decree Paxos

Fase 1:

- Proponer *ballot* b, previniendo que se acepten propuestas anteriores
- Recolectar información sobre propuestas ya aceptadas

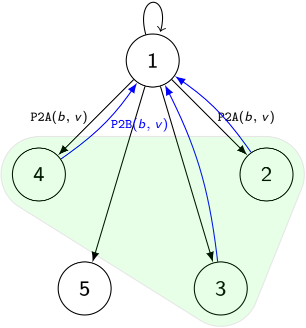
$\{\texttt{P1B}(b, c_j, v_j) \mid p_j \in \mathcal{Q}\}$ de un quórum $\mathcal{Q}$



Single Decree Paxos

Fase 2:

- Si $\forall p_j.\ p_j \in \mathcal{Q} \Rightarrow v_j = \bot$, es libre de proponer su valor
- En caso contrario, debe proponer v_{j_0} tal que $\forall p_j.\ p_j \in \mathcal{Q} \Rightarrow c_j \leq c_{j_0}$



$\{\mathtt{P2B}(b, v) \mid p_j \in \mathcal{Q}\}$ de un quórum $\mathcal{Q} \Rightarrow v$ es elegido

Single Decree Paxos

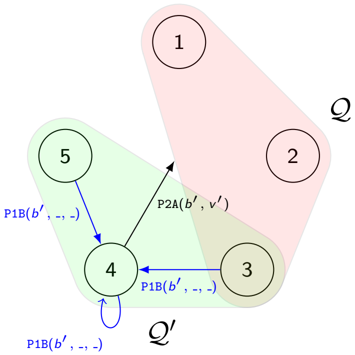
- Objetivo: demostrar que si v y v' son elegidos, entonces $v' = v$

- Es suficiente probar que si v es elegido en el ballot b y v' es propuesto en un ballot $b' > b$, entonces $v = v'$

- Más aún, debemos probar que si un valor v ha sido elegido o aún puede ser elegido en el ballot b y v' es propuesto en un ballot $b' > b$, entonces $v = v'$

Single Decree Paxos

- Cada $q \in \mathcal{Q}'$ se ha unido al ballot b'
- Cada $q \in \mathcal{Q}$ o bien ha recibido y aceptado $\mathtt{P2A}(b, v)$ o su ballot es menor o igual a b
- $3 \in \mathcal{Q} \cap \mathcal{Q}'$
  - $3 \in \mathcal{Q}' \Rightarrow \mathtt{bal} \geq b'$
  - $3 \in \mathcal{Q} \Rightarrow 3$ ha enviado $\mathtt{P2B}(b, v)$ o tiene $\mathtt{bal} \leq b$
  - Por lo tanto, 3 ha enviado $\mathtt{P2B}(b, v)$ y entonces $\mathtt{cbal} \geq b$



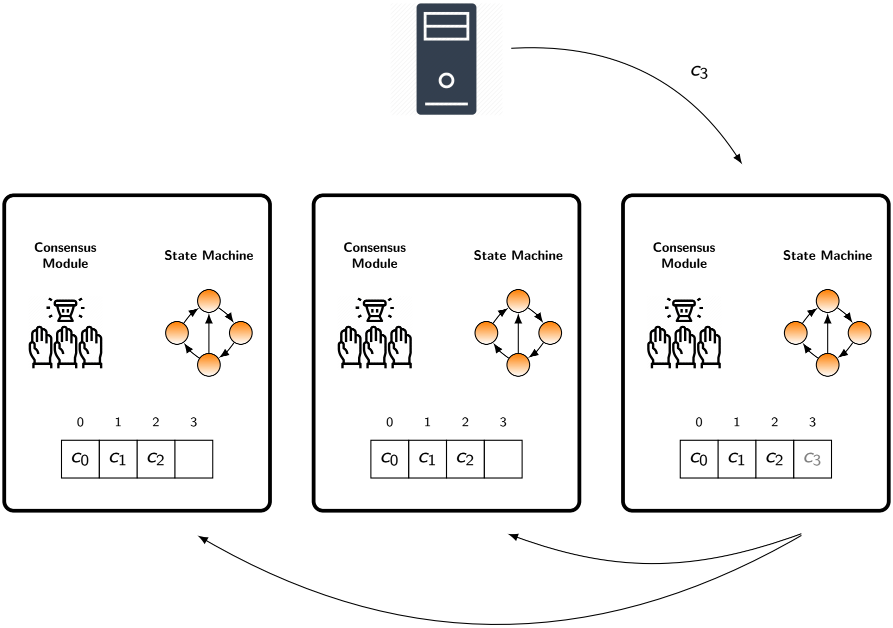
- $v' = v_{j_0}$ tal que $\forall p_j.\ p_j \in \mathcal{Q}' \Rightarrow c_j \leq c_{j_0}$ y por lo tanto $c_{j_0} \geq b$
- Por lo tanto $\mathtt{P2A}(c_{j_0}, v')$ es un mensaje enviado. Si $c_{j_0} > b$, entonces (HI) $v' = v$. Si $c_{j_0} = b$, entonces $\mathtt{P2A}(b, v)$ y $\mathtt{P2A}(b, v')$ son enviados, por lo que $v' = v$

FLE Atomic Broadcast

Un protocolo de broadcast atómico permite a un proceso transmitir un mensaje a un grupo de procesos de manera tal que:

- Los procesos acuerdan el conjunto de mensajes entregados
- Los procesos acuerdan también el orden en que esto ocurre

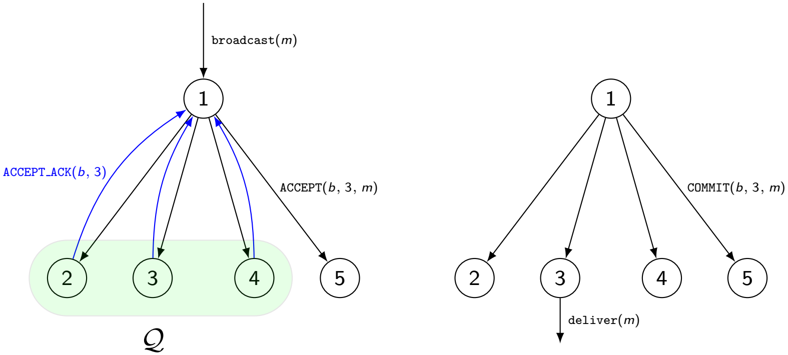
FLE Atomic Broadcast



FLE Atomic Broadcast

- Operación normal: Un único líder propone una secuencia de mensajes a sus seguidores con el objetivo de replicarlos y garantizar la durabilidad de los mensajes entregados y su orden

0	1	2	3
c_0	c_1	c_2	



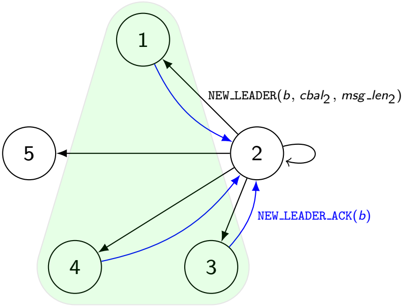
FLE Atomic Broadcast

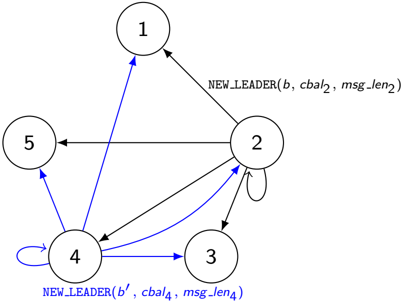
Recuperación a fallos:

- elegir un líder quien convenza a un quórum de participar de su ballot y cuyo estado domine al quórum
- garantizar que antes de que un seguidor empiece a aceptar propuestas del nuevo líder, este ha sincronizado su estado con aquel del líder

FLE Atomic Broadcast

- elegir un líder quien convenza a un quórum de participar de su ballot y cuyo estado domine al quórum

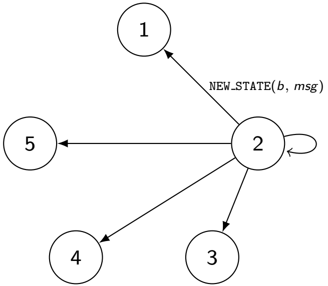


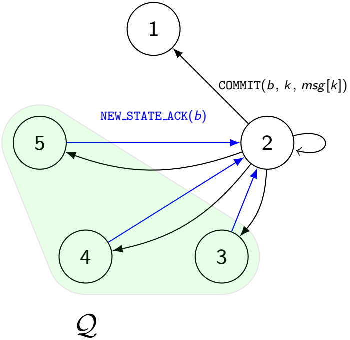


$(cbal_4, msg\_len_4) \geq (cbal_2, msg\_len_2)$

FLE Atomic Broadcast

- garantizar que antes de que un seguidor empiece a aceptar propuestas del nuevo líder, este ha sincronizado su estado con aquel del líder





FLE Atomic Broadcast

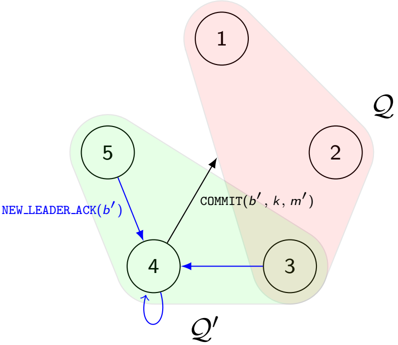
- Objetivo: demostrar que todos los procesos entregan un prefijo de una secuencia global común de mensajes

- Puesto que el k-ésimo mensaje entregado por un proceso p sólo puede ser entregado en respuesta a un mensaje $\texttt{COMMIT}(\_, k, \_)$, es suficiente demostrar que si $\texttt{COMMIT}(b, k, m)$ y $\texttt{COMMIT}(b', k, m')$ son mensajes enviados, entonces $m = m'$

- Más aún, debemos probar que si un mensaje m ha sido entregado o aún puede ser entregado en el ballot b para el slot k y $\texttt{COMMIT}(b', k, m')$ es un mensaje enviado con $b' > b$, entonces $m = m'$

FLE Atomic Broadcast

- Cada $q \in \mathcal{Q}'$ se ha unido al ballot b'
- Cada $q \in \mathcal{Q}$ o bien ha recibido y aceptado $\texttt{ACCEPT}(b, k, m)$ o $\texttt{NEW\_STATE}(b, msg)$ con $msg[k] = m$, o $q = \texttt{leader}(b)$, o su ballot es menor o igual a b
- $3 \in \mathcal{Q} \cap \mathcal{Q}'$
  - $3 \in \mathcal{Q}' \Rightarrow \texttt{bal} \geq b'$
  - $3 \in \mathcal{Q} \Rightarrow 3$ o bien ha recibido y aceptado $\texttt{ACCEPT}(b, k, m)$ o $\texttt{NEW\_STATE}(b, msg)$ con $msg[k] = m$, o $q = \texttt{leader}(b)$
  - Por lo tanto, en 3 debemos tener $(\texttt{cbal}, \texttt{len}(\texttt{msg})) \geq (b, k)$



- Si $\texttt{cbal} > b$, entonces (HI) $msg[k] = m$. Si $\texttt{cbal} = b$ entonces $\texttt{len}(\texttt{msg}) \geq k$, por lo que $msg[k] = m$
- Por lo tanto si $\texttt{COMMIT}(b', k, m')$ es un mensaje enviado, entonces $m = m'$

Vertical Paxos

La mayoría de las soluciones por replicación requieren $2f + 1$ procesos para tolerar f fallos crash-stop. Esto es costoso: si la información se persiste en todos los procesos, sólo $f + 1$ son necesarios para garantizar durabilidad

Vertical Paxos

- El fallo de sólo un proceso bloquearía el procesamiento de mensajes, por lo que para recuperarse se requiere cambiar la membresía para reemplazar los procesos fallidos por procesos frescos

- Los procesos necesitan acordar la próxima configuración que se reduce a consenso, lo cual nuevamente requiere $2f + 1$ procesos

Vertical Paxos

Una solución consiste en utilizar un servicio de configuración independiente con $2f + 1$ procesos únicamente para realizar consenso en la configuración y sólo $f + 1$ para replicar la información relevante

Vertical Paxos

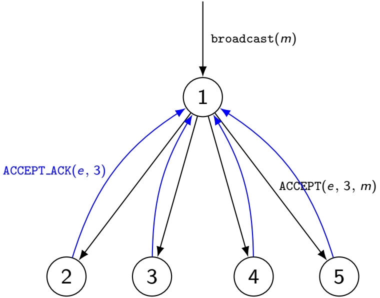
Un protocolo de broadcast atómico permite a un proceso transmitir un mensaje a un grupo de procesos de manera tal que:

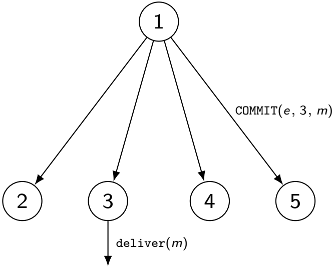
- Los procesos acuerdan el conjunto de mensajes entregados
- Los procesos acuerdan también el orden en que esto ocurre

Vertical Paxos

- Operación normal: Un único líder propone una secuencia de mensajes a todos los miembros de su configuración con el objetivo de replicarlos y garantizar la durabilidad de los mensajes entregados y su orden

0	1	2	3
c_0	c_1	c_2	





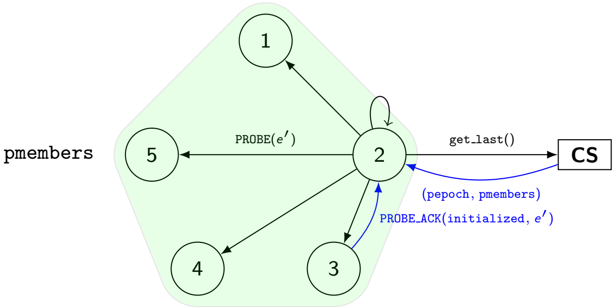
Vertical Paxos

Recuperación a fallos:

- elegir un líder quien convenza a todos los miembros de una configuración de participar en su época y cuyo estado contiene todos los mensajes aceptados
- garantizar que antes de que un seguidor empiece a aceptar propuestas del nuevo líder, este ha sincronizado su estado con aquel del líder

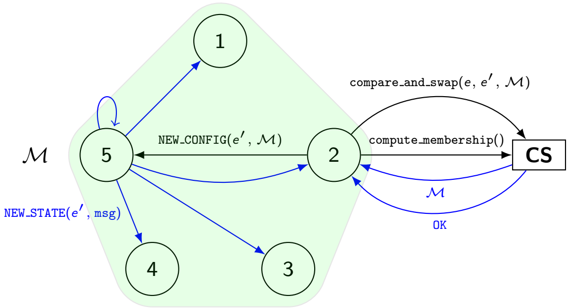
Vertical Paxos

- elegir un líder quien convenza a todos los miembros de una configuración de participar en su época y cuyo estado contiene todos los mensajes aceptados



Vertical Paxos

- si algún proceso responde con PROBE_ACK(FALSE, e'), sabemos que la época probada no puede ser operativa, por lo que probamos una época precedente
- si algún proceso (digamos 5) responde con PROBE_ACK(TRUE, e'), podemos garantizar que el estado de este proceso contiene todos los mensajes aceptados



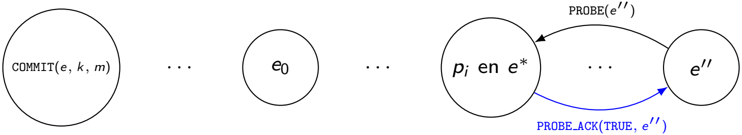
Vertical Paxos

- Objetivo: demostrar que todos los procesos entregan un prefijo de una secuencia global común de mensajes

- Puesto que el k-ésimo mensaje entregado por un proceso p sólo puede ser entregado en respuesta a un mensaje $\texttt{COMMIT}(\_, k, \_)$, es suficiente demostrar que si $\texttt{COMMIT}(e, k, m)$ y $\texttt{COMMIT}(e', k, m')$ son mensajes enviados, entonces $m = m'$

- Es suficiente probar que si $\texttt{COMMIT}(e, k, m)$ es un mensaje enviado y en un proceso tenemos $\texttt{cepoch} = e' > e$ entonces $\texttt{msg}[k] = m$

Vertical Paxos



- Todos los procesos en e se encuentran inicializados, por lo que la prueba no puede haber ido más allá de e. Por lo tanto $e \leq e^* < e''$

- Sea $e_0 = \texttt{cepoch}$ en p_i al enviar el mensaje PROBE_ACK. Si $\texttt{cepoch} < e$, entonces p_i no sería un miembro de e y por lo tanto no podría ser miembro de e^*. Luego $e \leq e_0$. Por otro lado, todos los miembros de la época probada e^* deben tener $\texttt{cepoch} \leq e^* < e''$. En consecuencia $e \leq e_0 < e''$

- Si $e_0 > e$ entonces (HI) $\texttt{msg}[k] = m$. Si $e_0 = e$, cada proceso en e debe haber recibido y notificado su mensaje ACCEPT o NEW_STATE antes de notificar PROBE(e''). Por lo tanto $\texttt{msg}[k] = m$

Prueba del Invariante 5. Probamos el invariante por inducción en e'. Asúmase que el invariante se cumple para cada $e' < e''$. Probamos ahora que se cumple para $e' = e''$ por inducción en la longitud de la ejecución del protocolo.

Considérese la transición en la línea 51, cuando un futuro líder p_i se incorpora a la época e''. Mostramos que luego de esta transición tenemos $\mathsf{msg}[k] = m$. Puesto que p_i fue elegido como el futuro líder de la época e'', este proceso ha respondido con $\mathtt{PROBE\_ACK}(\mathtt{TRUE}, e'')$ a $\mathtt{PROBE}(e'')$. Por lo tanto, p_i era miembro de una configuración en una época $e^* < e''$ que estaba siendo probada. La prueba finaliza cuando al menos un proceso envía un mensaje $\mathtt{PROBE\_ACK}(\mathtt{TRUE}, e'')$. La hipótesis de que $\mathtt{COMMIT}(e, k, m)$ es un mensaje enviado impica que en algún punto $\mathtt{NEW\_STATE}(e, msg)$ es un mensaje enviado con $msg[k] = m$ y este mensaje es notificado por todos los miembros de e excepto su líder, o todos los miembros en e respondieron con $\mathtt{ACCEPT\_ACK}$ a $\mathtt{ACCEPT}(e, k, m)$. Esto significa que todos los miembros en e se encuentran inicializados. Esto, junto con el Invariante 2, implican que la fase de prueba no puede haber ido más allá de e. Por lo tanto, $e \leq e^* < e''$.

Sea e_0 el valor de cepoch en p_i justo antes de la transición en la línea 51. Si $e_0 < e$ entonces p_i no sería un miembro de e, pues todos los miembros de e se encuentran inicializados y tienen $\mathsf{cepoch} = e$, y en consecuencia, por el Invariante 3, no podría haber sido elegido como el futuro líder de la época e''. En consecuencia, $e_0 \geq e$. Por otro lado, como todos los miembros de la época e^* siendo probada tienen $\mathsf{cepoch} \leq e^*$, debemos tener $e_0 \leq e^* < e''$. En consecuencia, $e \leq e_0 < e''$.

Si $e < e_0$, entonces por la hipótesis inductiva, tenemos $\mathsf{msg}[k] = m$ justo luego de la transición en la línea 51, como queríamos. Asúmase ahora que $e_0 = e$. Como $\mathtt{COMMIT}(e, k, m)$ es un mensaje enviado, entonces cada proceso en e recibe $\mathtt{NEW\_STATE}(e, msg)$ con $msg[k] = m$ y responde con $\mathtt{NEW\_STATE\_ACK}(e)$ o cada proceso en e recibe $\mathtt{ACCEPT}(e, k, m)$ y responde con $\mathtt{ACCEPT\_ACK}(e, k)$. El Invariante 2 implica que p_i envió su mensaje $\mathtt{NEW\_STATE}$, $\mathtt{NEW\_STATE\_ACK}$, $\mathtt{ACCEPT}$ o $\mathtt{ACCEPT\_ACK}$ antes de enviar $\mathtt{PROBE\_ACK}(\mathtt{TRUE}, e'')$. Entonces lo requerido sigue del Invariante 4.

Esto implica, en particular, que cualquier mensaje $\mathtt{NEW\_STATE}(e'', msg)$ enviado con $e'' > e$ debe tener $\mathsf{msg}[k] = m$, por lo que el resultado se cumple luego de la transición en la línea 56. Las líneas 67 y 14 aseguran que el valor no será sobreescrito durante el caso libre de fallos del protocolo. El resultado es trivial para el resto de las transiciones. □

Prueba del Invariante 6. Probamos el invariante por inducción en la longitud de la ejecución del protocolo. Inicialmente $\mathsf{cbal} = 0$ para todos los procesos y no existen mensajes enviados, por lo que la afirmación es trivialmente verdadera. Para el paso inductivo, supóngase que $\mathtt{committable}(b, k, m)$ se cumple. El resultado sigue trivialmente de la hipótesis inductiva para todas las transiciones excepto aquellas en las líneas 37 y 47.

Considérese entonces la transición en la línea 37 por un proceso p_i la cual gestiona la recepción de mensajes $\mathtt{NEW\_LEADER\_ACK}(b')$ de un quórum $\mathcal{Q}$ en respuesta a un mensaje $\mathtt{NEW\_LEADER}(b', cbal, msg\_len)$. La hipótesis implica que o bien $\mathtt{acceptable}(b, k, m)$ o $\mathtt{recoverable}(b, k, m)$ se cumple.

Si $\mathtt{acceptable}(b, k, m)$ se cumple, entonces existe un quórum $\mathcal{Q}'$ tal que cada proceso ha recibido un mensaje $\mathtt{ACCEPT}(b, k, m)$ y ha respondido con $\mathtt{ACCEPT\_ACK}(b, k)$ o este tiene $\mathsf{bal} \leq b$. Sea $q \in \mathcal{Q} \cap \mathcal{Q}'$. Dado que $q \in \mathcal{Q}'$ ha recibido y notificado $\mathtt{NEW\_LEADER}(b', cbal, msg\_len)$, este tiene $\mathsf{bal} \geq b' > b$. Entonces, q debe haber enviado su mensaje $\mathtt{ACCEPT\_ACK}$ antes del mensaje $\mathtt{NEW\_LEADER\_ACK}$. Por la precondición en la línea 14 y el hecho de que si $\mathsf{status} \in \{\text{LEADER}, \text{FOLLOWER}\}$ entonces $\mathsf{bal} = \mathsf{cbal}$, cuando q envió el mensaje $\mathtt{ACCEPT\_ACK}$, este debe haber tenido $\mathsf{cbal} = b$. Entonces, cuando q envió el mensaje $\mathtt{NEW\_LEADER\_ACK}$ este tenía $\mathsf{cbal} \geq b$. En consecuencia, $cbal \geq b$.

Si, por otro lado, $\mathtt{recoverable}(b, k, m)$ se cumple, entonces existe un quórum de procesos $\mathcal{Q}'$ cuyos procesos p son tales que $p = \mathtt{leader}(b)$, p ha recibido y notificado $\mathtt{NEW\_STATE}(b, msg)$ con $msg[k] = m$, o este tiene $\mathsf{bal} \leq b$. Sea $q \in \mathcal{Q} \cap \mathcal{Q}'$. Puesto que $q \in \mathcal{Q}'$ ha recibido y notificado $\mathtt{NEW\_LEADER}(b', cbal, msg\_len)$, este tiene $\mathsf{bal} \geq b' > b$. Entonces, q debe haber enviado su mensaje $\mathtt{NEW\_STATE}$ o $\mathtt{NEW\_STATE\_ACK}$ antes del mensaje $\mathtt{NEW\_LEADER\_ACK}$. Las líneas 39 y 43 implican que $\mathsf{cbal} = b$ cuando q envió este mensaje. Entonces, cuando q envió el mensaje $\mathtt{NEW\_LEADER\_ACK}$ este tenía $\mathsf{cbal} \geq b$. En consecuencia, $cbal \geq b$.

Puesto que p_i envió $\mathtt{NEW\_LEADER}(b', cbal, msg\_len)$, y por la línea 38, $\mathsf{bal} = b'$ y $\mathsf{status} = \text{RECOVERING}$, el Invariante 5 implica que en p_i tenemos $\mathtt{len}(\mathsf{msg}) = msg\_len$ y $\mathsf{cbal} \geq cbal \geq b$.

Si $\mathsf{cbal} > b$, la hipótesis inductiva implica que $\mathsf{msg}[k] = m$. Supóngase entonces que $\mathsf{cbal} = b$. Por la línea 30, cuando q notificó $\mathtt{NEW\_LEADER}(b', cbal, msg\_len)$ este debe haber tenido $\mathtt{len}(\mathsf{msg}) \leq msg\_len$. En el caso en que $\mathtt{acceptable}(b, k, m)$ se cumple, q ha recibido y notificado $\mathtt{ACCEPT}(b, k, m)$. Entonces, cuando q envió el mensaje $\mathtt{NEW\_LEADER\_ACK}$ este tenía $\mathtt{len}(\mathsf{msg}) \geq k$ y por lo tanto $msg\_len \geq k$. Entonces, por el Invariante 4, debemos tener $\mathsf{msg}[k] = m$. Si, por otro lado, $\mathtt{recoverable}(b, k, m)$ se cumple, entonces existe msg tal que $msg[k] = m$ y $\mathtt{NEW\_STATE}(b, msg)$ es un mensaje enviado, en cuyo caso, por el Invariante 3, msg es un prefijo de msg y por lo tanto $\mathsf{msg}[k] = m$.

En consecuencia, el mensaje $\mathtt{NEW\_STATE}(b', msg')$ enviado durante la transición actual tiene $msg'[k] = \mathsf{msg}[k] = m$. Ningún mensaje $\mathtt{COMMIT}$, $\mathtt{ACCEPT\_ACK}$ o $\mathtt{ACCEPT}$

- Estas pruebas son igualmente complejas y pueden también albergar errores sutiles

- Por lo tanto, para eliminar categóricamente errores de nuestros algoritmos, aspiramos a construir pruebas de verificación automatizadas chequeadas por máquina de sus propiedades de safety

- Utilizamos Dafny, un lenguaje de programación verifier-friendly basado en lógica de Floyd-Hoare que automatiza la verificación vía el SMT solver Z3

FIN