(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 935 964**

(51) Int. CI.:

G06F 9/50 (2006.01)
G06F 9/52 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **15.01.2019 E 19450001 (3)**

(97) Fecha y número de publicación de la concesión europea: **02.11.2022 EP 3683678**

(54) Título: **Método implementado por ordenador, programa informático y sistema de procesado de datos**

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
13.03.2023

(73) Titular/es:

IOV42 LIMITED (100.0%)
Regina House, 124 Finchley Road
London NW3 5JS, GB

(72) Inventor/es:

ZAPFEL, ROBERT

(74) Agente/Representante:

CURELL SUÑOL, S.L.P.

DESCRIPCIÓN

Método implementado por ordenador, programa informático y sistema de procesado de datos

La invención se refiere a un método implementado por ordenador para la elección, basada en procesos aleatorios, de un líder en una red distribuida de dispositivos de procesado de datos, comprendiendo dicha red distribuida una pluralidad de procesos asíncronos identificados, en el que la totalidad de dichos procesos identificados o un subconjunto de los mismos son procesos en ejecución que participan en la elección del líder.

Además, la invención se refiere a un producto de programa informático que comprende instrucciones que, cuando el programa es ejecutado por dispositivos de procesado de datos, tales como un ordenador, dispuestos en una red distribuida, consiguen que los dispositivos de procesado de datos lleven a cabo el método inventivo.

Además, la invención se refiere a un sistema de procesado de datos que comprende una pluralidad de dispositivos de procesado de datos que comprenden medios para llevar a cabo el método inventivo.

Campo de la invención

Los algoritmos de computación distribuida hacen uso de puntos de sincronización con el fin de distribuir y volver a recopilar el trabajo realizado por un grupo de procesos en ejecución asíncrona. Para alcanzar esta sincronización se elige un líder dentro del grupo de procesos. Este líder actúa como punto de contacto único para los clientes, distribuye la solicitud entre los procesos, espera para volver a recopilar resultados y envía una respuesta fusionada de vuelta al cliente.

Acordar la identidad de un líder en una red distribuida de dispositivos de procesado de datos requiere protocolos de consenso. Los protocolos de consenso garantizan que todos los procesos dentro del grupo de procesos elijan el mismo líder en un instante de tiempo dado. Los protocolos de consenso son cruciales para el funcionamiento de las tecnologías de registros distribuidos, con el fin de replicar, compartir y sincronizar datos digitales distribuidos geográficamente sobre múltiples emplazamientos, países o instituciones.

Según se usa en la presente memoria, un proceso es la instancia de un programa informático que se está ejecutando en un dispositivo de procesado de datos. Contiene el código de programa y su actividad. En una red distribuida, los procesos se ejecutan en dispositivos de procesado de datos que están distribuidos sobre dicha red, en donde dichos dispositivos de procesado de datos pueden funcionar como nodos de dicha red. A un proceso que se ejecuta en un nodo se le puede hacer referencia también como “nodo”. Un proceso es identificable dentro de la red por medio de un identificador exclusivo, tal como una dirección de red.

Según se usa en la presente memoria, la expresión “proceso identificado” designa cualquier proceso que se está ejecutando en un dispositivo de procesado de datos situado en la red y que es conocido por el sistema. La expresión “proceso en ejecución” se refiere a un proceso que está participando en ese momento en la elección del líder, en donde esto puede incluir todos los procesos identificados o solo un subconjunto de ellos.

En términos generales, es necesario que los algoritmos de elección de líder satisfagan un conjunto mínimo de requisitos que comprenden las siguientes condiciones:

- Vivacidad: todo proceso debe entrar finalmente en un estado de elegido o no elegido.
- Seguridad: solamente un único proceso puede entrar en el estado de elegido dentro de una ronda electiva (también conocida como condición de unicidad).
- Terminación: la elección debe finalizar en el transcurso de un espacio de tiempo finito.
- Acuerdo: todos los procesos saben quién es el líder.

Técnica anterior

Los algoritmos de elección de líder se pueden clasificar basándose en su estrategia de comunicación subyacente (o bien de un solo salto o bien multipaso) para la comunicación entre los procesos distribuidos:

- Síncronos: los procesos requieren una señal de reloj común para sincronizarse.
- Asíncronos: los procesos pueden trabajar a velocidades arbitrarias.

En el caso de redes de un solo salto (en las que toda la comunicación se realiza directamente sin ningún intermediario), los algoritmos de elección de líder dividen el tiempo en ranuras de tiempo iguales y envían sus mensajes dentro de esas ranuras de tiempo, en donde los algoritmos se pueden clasificar sobre la base de la

información almacenada:

- Indiferentes: estos algoritmos no poseen ningún historial. La probabilidad de transmisión puede cambiar a lo largo del tiempo, pero es la misma para todos los nodos.

- Uniformes: en estos algoritmos se guarda un historial de estados del canal. La probabilidad de transmisión es una función del historial del canal y es la misma para todos los nodos. El algoritmo no conoce el número de nodos.

- No uniformes: en estos algoritmos se guarda tanto el historial de estados del canal como el historial de transmisión. La probabilidad de transmisión depende de estos dos historiales y es diferente para varios nodos. El algoritmo conoce el número de nodos.

Para redes multipaso, en las que todos los nodos tienen identificadores exclusivos y todas las conexiones son bilaterales y FIFO, no hay necesidad de conocer a priori el número de nodos participantes y no hay restricciones sobre el número máximo de los mismos. Las redes de este tipo pueden funcionar con los siguientes algoritmos de elección de líder:

- Algoritmos de determinación de extremos, en los que se eligen líderes sobre la base del valor de ciertos parámetros, tales como el poder computacional o la energía restante. Son ejemplos de los mismos: LEAA (Algoritmo de elección de líderes para redes ad hoc), CBLEAA (LEAA basado en candidatos), SEFA (Determinación segura de extremos), SPLEA (Elección segura de líderes basada en preferencias);

- Algoritmos jerárquicos, en los que una red se divide en conglomerados [del inglés, *clusters*], incluido un grupo de líderes de conglomerado denominados cabeza de conglomerado y algún otro nodo del conglomerado. Son ejemplos de los mismos: LPL (del inglés, *Election Portion Leader*), SPHA (Algoritmo Jerárquico de Propósito Especial), LELN (Algoritmo de Elección de Líderes en Red de Bajo nivel);

- Algoritmos de elección aleatoria, en los que los nodos tienen identificadores exclusivos y se organizan en grafos acíclicos de orientación ad hoc. En la actualidad, la aleatorización se usa únicamente para la formación de grafos ad hoc, mientras que la elección de líder principal con dichos grafos utiliza otros medios, como la votación. Son ejemplos de los mismos: TORA, Maplani, Derhab.

La presente invención pertenece a la categoría de algoritmos de elección de líder aleatoria relacionados con redes multipaso.

El TORA es un algoritmo de elección aleatoria de líderes, en el que la elección se basa en identificadores de nodo exclusivos. En este algoritmo, se asigna una altura a cada nodo. El algoritmo se basa en la construcción de árboles direccionales “sin bucles ni DAG”: en estos árboles, todos los nodos se guían hacia un nodo de destino. Requiere que los nodos únicamente se comuniquen con sus vecinos actuales. La elección del líder se realiza comparando parámetros de cada nodo. Se dibuja una arista desde un nodo de mayor altura a un nodo de menor altura. Cada nodo tiene cinco parámetros, sobre cuya base se efectúa una comparación de la altura. La comparación se realiza comenzando con el primer parámetro. Si este parámetro tiene el mismo valor para los dos nodos, se tiene en cuenta el siguiente parámetro. Si los cuatro primeros parámetros son iguales, se compara el quinto parámetro, ya que es el identificador de nodo exclusivo, que es diferente para cada nodo. El líder es entonces el nodo con la mayor altura. En caso de una partición de red, el nodo que detecta la partición emite una indicación hacia otros nodos de su componente de manera que dejen de realizar cambios de altura. El TORA utiliza una fuente de tiempo global y acepta segmentaciones y fusiones.

El Maplani es un algoritmo de elección de líderes para redes ad hoc de móviles. Se basa en el TORA. A diferencia del TORA, el Maplani usa seis parámetros en lugar de cinco. El parámetro adicional es el ID de un nodo del que se cree que es el líder del nodo i-ésimo. La forma con la que el Maplani trabaja con particiones de red es también diferente: aquí, el primer nodo que detecta la partición se elige a sí mismo como líder del componente (partición) nuevo. En caso de que se junten dos componentes, el líder del componente con el ID más pequeño se convertirá finalmente en el líder único del nuevo componente fusionado. Aparte de los casos de segmentación, la elección de líder básica sigue siendo igual a la del TORA: el líder es el nodo con la mayor altura.

El Derhab se fundamenta en el Maplani e intenta resolver la cuestión en la que los nodos no siempre responden (y se producen cambios simultáneos) designando los tres primeros parámetros en altura como “nivel de referencia” y concediendo al nodo con el nivel de referencia menor una prioridad mayor. El algoritmo Derhab también mantiene información adicional en cada nodo (la primera es el tiempo de inicio del proceso, la segunda el tiempo en el que el nodo recibió el número de referencia especial). Se realiza también una comparación de estos parámetros adicionales. Esto hace que el algoritmo Derhab sea de 10 a 100 veces más rápido que el algoritmo Maplani.

El inconveniente de los algoritmos actuales de elección aleatoria de líderes es que la aleatoriedad se está usando únicamente con vistas a ejercer una función facilitadora para métodos de elección de líderes de orden superior, tal

como la selección de parámetros o la creación de grafos temporales para ellos, o con vistas a crear grupos o subgrupos de nodos dentro de una red. Los métodos actuales, en muchos de sus tipos y variaciones, dejan al descubierto al líder actual como debilidad sistémica y punto único de fallo bajo ataques adversarios.

El documento WO 2018/217804 A divulga una red para obtener una velocidad de verificación mejorada, con datos resistentes a manipulaciones indebidas. Se describe un proceso de elección de líderes, el cual se basa en la determinación de un grafo muestreador en cada nodo localmente. El grafo muestreador se puede determinar sobre la base de los identificadores de nodo y un valor semilla predefinido conocido por todos los nodos de la red informática. Por esta razón, una de las desventajas de este proceso es que cada nodo no determina su propia información aleatoria.

El documento US 2012/124412 A1 divulga un método para llevar a cabo la elección de un nodo líder a partir de múltiples nodos en múltiples procesos dentro de un único sistema informático, o dentro de una red de sistemas informáticos.

Por lo tanto, la presente invención tiene como objetivo mejorar un método para la elección, basada en procesos aleatorios, de líderes en una red distribuida. En particular, la presente invención tiene como objetivo aumentar el nivel de aleatoriedad en el proceso de elección del líder con el fin de conseguir que la predicción de la determinación del líder sea computacionalmente difícil y evitar manipulaciones. Se reducirá al mínimo el riesgo de que un adversario controle la elección del líder manipulando el proceso de manera que actúe en su favor.

Sumario de la invención

Con el fin de poner solución a estos y otros objetivos, la invención proporciona un método implementado por ordenador para una elección de líder basada en procesos aleatorios según la reivindicación 1. En consecuencia, el método se refiere a la elección de líderes basada en procesos aleatorios en una red distribuida de dispositivos de procesado de datos, comprendiendo dicha red distribuida una pluralidad de procesos asíncronos identificados, en el que la totalidad de dichos procesos identificados o un subconjunto de los mismos son procesos en ejecución que participan en la elección del líder, comprendiendo dicho método las siguientes etapas:

a) una información aleatoria (r) es generada por cada proceso en ejecución y compartida con los otros procesos en ejecución, de manera que cada proceso en ejecución mantiene un conjunto de dicha información aleatoria (r),

b) una información aleatoria distribuida (R) es calculada por cada proceso en ejecución a partir del conjunto de información aleatoria (r) aplicando una primera función de transformación compartida (f_1), de manera que se ponga a disposición de cada proceso en ejecución la misma información aleatoria distribuida (R),

c) un designador de uno solo de dichos procesos en ejecución es calculado a partir de la información aleatoria distribuida (R) por medio de una segunda función de transformación compartida (f_2),

d) un líder es elegido entre dichos procesos en ejecución sobre la base de dicho designador.

Por lo tanto, la invención se basa en la idea de deducir la asignación del líder a partir de una información aleatoria distribuida (R) generada conjuntamente por todos los procesos en ejecución en solamente una ronda de comunicación, permitiendo que el líder sea calculado de forma autónoma por cada proceso. En particular, cada proceso en ejecución contribuye con su propia información aleatoria (r) al cálculo de la información aleatoria distribuida (R) de manera que se alcanza un nivel de aleatoriedad muy elevado.

Si, tal como se produce según una forma de realización preferida de la invención, la secuencia de etapas a) - d) se repite a intervalos regulares o irregulares, el líder elegido cambia aleatoriamente.

La información aleatoria (r) generada localmente por cada proceso en ejecución puede ser cualquier elemento de información digital que se genere aleatoriamente, y puede adoptar preferentemente la forma de un número. Para generar dicha información aleatoria (r) en cada proceso en ejecución se puede usar un generador de bits aleatorios no determinista o un generador de bits aleatorios determinista, es decir, un generador de números seudoaleatorios. Según la etapa a) del método de la invención, la información aleatoria (r) que es generada por cada proceso en ejecución se comparte con los otros procesos en ejecución, de manera que cada proceso en ejecución mantiene un conjunto de dicha información aleatoria. En particular, dicha compartición de la información aleatoria (r) entre el grupo de procesos en ejecución se lleva a cabo en una única ronda de comunicación, es decir, el método no es interactivo. Esto significa que únicamente se requiere una comunicación unidireccional para que un proceso en ejecución envíe su información aleatoria (r) a otro proceso en ejecución.

La transmisión de la información aleatoria (r) desde el proceso en ejecución respectivo a los otros procesos puede ser directa o indirecta. No obstante, incluso en el caso de una comunicación indirecta, la información aleatoria (r) permanece preferentemente invariable y/o sin procesar durante su trayecto de comunicación. En particular, la

compartición de información aleatoria (r) se efectúa preferentemente sin usar nodos mediadores para sincronizar información con el fin de acordar un curso aleatorio común, ya que dichos mediadores presentarían una vulnerabilidad particular para el proceso de elección del líder.

Según la etapa b) de la invención, cada proceso en ejecución calcula la información aleatoria distribuida (R) a partir del conjunto de información aleatoria (r) aplicando una primera función de transformación compartida (f_1). Todos los procesos en ejecución conocen la misma primera función de transformación (f_1), de manera que todos los procesos en ejecución pueden calcular la información aleatoria distribuida (R) por sí solos y de forma mutuamente independiente. En general, se puede usar cualquier tipo de función de transformación que establezca que la información aleatoria distribuida (R) es una función de la información aleatoria (r) de todos los procesos en ejecución. La primera función de transformación se puede basar en cualquier operación de cálculo, tal como una concatenación, XOR y *hashing*, o una combinación de las mismas. Según la invención, la primera función de transformación compartida es

$$R = \prod_{i=1}^{n} r_i (mod\ o),$$

en la que

R es la información aleatoria distribuida,

r_i es la información aleatoria,

mod designa la operación módulo, y

o es un número primo de *Mersenne* definido como $o = 2^n-1$, siendo n preferentemente ≥ 31.

Esto significa que el cálculo de R se basa en una multiplicación de los enteros (r) en un campo finito de orden primo. De esta manera, se puede elegir un líder entre un grupo de procesos con la misma probabilidad por proceso.

La primera función de transformación también se puede concebir de tal manera que la información aleatoria distribuida (R) sea una función de la información aleatoria (r) de los procesos en ejecución y, adicionalmente, de otro elemento de información, tal como datos estructurados de una solicitud de cliente.

Para deducir de forma fiable un líder a partir de la información aleatoria distribuida (R) en cada proceso en ejecución, la invención prevé que cada proceso en ejecución mantenga un conjunto ordenado (K) de todos los procesos en ejecución y el número total (k) de procesos en ejecución. El conjunto ordenado de todos los procesos en ejecución contiene todos los procesos en ejecución que participan en la elección del líder, lo cual requiere que todos los procesos en ejecución conocidos sean identificables por cada proceso. En particular, cada proceso en ejecución es identificable por medio de un identificador exclusivo, tal como una dirección de red. Que el conjunto esté ordenado significa que los procesos en ejecución están contenidos en el conjunto en una secuencia definida de manera que cada proceso tiene una posición definida dentro del conjunto. Con respecto a la etapa c) del método inventivo, se usa una segunda función de transformación compartida (f_2) para calcular un designador de uno solo de dichos procesos en ejecución a partir de la información aleatoria distribuida (R). Puesto que la información aleatoria distribuida (R) se calcula en cada proceso en ejecución, el líder se puede elegir en cada proceso en ejecución de manera autónoma, es decir, a partir de información local en su totalidad, sin requerir interacciones adicionales ni entre los procesos en la red distribuida ni con ningún proceso externo.

De acuerdo con la invención, la segunda función de transformación compartida (f_2) se define como

$m = R\ (mod\ k)$ o $m = R\ (mod\ k) + 1$,

en la que

m es el designador del líder elegido,

R es la información aleatoria distribuida,

k es el número total de procesos en ejecución, y

mod designa la operación módulo,

en la que el líder se elige seleccionando el proceso en ejecución que se corresponde con el elemento $m^{ésimo}$ en dicho conjunto ordenado (K) de procesos en ejecución.

Esta función específica, al basarse en la operación módulo del número total de procesos en ejecución, garantiza una distribución uniforme entre los procesos. El incremento de la operación módulo en el número 1 garantiza que

el resultado de la función esté en el intervalo de 1 a k. En este caso, el resultado m apunta directamente al proceso elegido, que se identifica como el elemento m^{ésimo} en el conjunto ordenado (K) de procesos en ejecución, donde el índice del primer elemento en dicho conjunto ordenado (K) comienza con 1.

En la elección del líder es crucial garantizar que la mayoría de todos los procesos identificados participa en el proceso de elección. En particular, debe cumplirse un cuórum específico. Con este fin, según una forma de realización preferida, cada proceso en ejecución mantiene información sobre el número total (n) de todos los procesos identificados y verifica si el número total (k) de procesos en ejecución se corresponde con un cuórum predefinido del número total (n) de todos los procesos identificados, en donde las etapas b), c) y/o d) se llevan a cabo únicamente si se cumple el cuórum. En el caso más sencillo, el cuórum se define como la mayoría simple n/2 + 1 de todos los procesos identificados. Para lograr una tolerancia a fallos, tal como la tolerancia a fallos bizantinos, el cuórum se fija de manera que se considere un número definido de procesos deficientes. Cuando se aplica un cuórum de mayoría bizantina, *f* será el número máximo de procesos deficientes tolerados y la relación entre *n* y *f* se limitará a $n = 3f + 1$. Un cuórum de mayoría bizantina se deduce usando la fórmula de la mayoría simple e ignorando los *f* procesos deficientes, para definir:

$$q_{byzantine} = \frac{n}{2} + f + 1$$

En un entorno dinámico, es preferible actualizar el sistema de manera que incluya procesos de unión que se incorporan al grupo de procesos en ejecución o excluya procesos, tales como procesos fallidos o salientes, del grupo de procesos en ejecución.

Con este fin, según una forma de realización preferida de la invención, el conjunto ordenado (K) de procesos en ejecución se actualiza para incluir un proceso que se incorpora al grupo de procesos en ejecución, en donde cada proceso en ejecución, incluido el proceso de unión, en la etapa a), comparte su conjunto ordenado (K) de todos los procesos en ejecución con los otros procesos, y el conjunto ordenado (K) mantenido en cada proceso en ejecución se fusiona con el conjunto ordenado compartido (K).

Además, según otra forma de realización preferida de la invención, el conjunto ordenado (K) de procesos en ejecución se actualiza de manera que se elimina un proceso que abandona el grupo de procesos en ejecución, en donde el proceso saliente envía un mensaje de salida que comprende un identificador de proceso a los otros procesos en ejecución, y el proceso saliente se elimina del conjunto ordenado (K) de procesos en ejecución.

Según todavía otra forma de realización preferida de la invención, el conjunto ordenado (K) de procesos en ejecución se actualiza para eliminar un proceso en ejecución fallido, en donde

- cada proceso identifica que no ha recibido ninguna información aleatoria (r) compartida por el proceso fallido,
- cada proceso envía un mensaje de fallo a todos los procesos en ejecución restantes consultando si se ha identificado el proceso fallido en los procesos en ejecución restantes, y
- se elimina el proceso fallido del conjunto ordenado (K) de procesos en ejecución al producirse la recepción de un mensaje de confirmación proveniente de todos los procesos en ejecución restantes.

En referencia a continuación a las posibles formas de compartir la información aleatoria (r) generada en cada proceso en ejecución con los otros procesos en ejecución según la etapa a) del método inventivo, una primera alternativa prevé que la compartición de información aleatoria (r) en la etapa a) comprenda las etapas de:

- cada proceso en ejecución entrega su información aleatoria (r) a un sistema de difusión de orden total,
- el sistema de difusión de orden total difunde la información aleatoria (r) recibida de todos los procesos en ejecución a cada proceso en ejecución en el mismo orden.

La ventaja de usar un sistema de difusión de orden total es que todos los procesos en ejecución reciben los mensajes que contienen la información aleatoria (r) en el mismo orden. Por lo tanto, cada proceso en ejecución tiene el mismo conjunto de información aleatoria (r), estando la información aleatoria (r) individual en el mismo orden, de manera que se puede calcular instantáneamente la información aleatoria distribuida (R).

En una segunda alternativa, en la que los procesos en ejecución reciben la información aleatoria (r) de los otros procesos en un orden arbitrario, la información aleatoria (r) debe ordenarse según criterios definidos para garantizar que el conjunto de información aleatoria mantenido en los procesos en ejecución es idéntico en todos los procesos en ejecución. A este respecto, una de las formas de realización preferidas de la invención prevé que la compartición de información aleatoria (r) en la etapa a) comprenda las etapas siguientes:

- cada proceso en ejecución asigna un identificador de ronda de generador (g) a la información aleatoria generada (r) para obtener una tupla (r,g) que consiste, cada una de ellas, en una información aleatoria (r) y un identificador de ronda de generador (g),

- cada proceso en ejecución envía la tupla, de manera preferente directamente, a todos los otros procesos en ejecución,

- cada proceso en ejecución recopila tuplas recibidas de los otros procesos en ejecución, para obtener dicho conjunto de información aleatoria (r), que adopta la forma de una colección de tuplas que consiste en tuplas (r,g) que tienen el mismo identificador de ronda de generador (g),

- comparar el número de tuplas en dicha colección de tuplas con el número total (k) de procesos en ejecución;

y se inicia la etapa b), si el número de tuplas en la colección local es igual al número total (k) de procesos en ejecución.

Si, tal como ocurre según una forma de realización preferida, los procesos en ejecución generan una información aleatoria (r) nueva a intervalos regulares, la compartición de información aleatoria (r) en la etapa a) comprende las etapas siguientes:

- cada proceso en ejecución asigna un identificador de ronda de generador (g) a cada información aleatoria generada (r) para obtener tuplas (r,g) que consisten, cada una de ellas, en una información aleatoria (r) y un identificador de ronda de generador (g),

- cada proceso en ejecución envía las tuplas directamente a la totalidad del resto de procesos en ejecución,

- cada proceso en ejecución recopila tuplas recibidas de los otros procesos en ejecución, para obtener conjuntos de información aleatoria (r), que adoptan la forma de colecciones de tuplas, consistiendo cada colección en tuplas (r,g) que tienen el mismo identificador de ronda de generador (g),

- una ronda de generador se marca como localmente completa si el número de tuplas en una colección de tuplas es igual al número total (k) de procesos en ejecución;

y se inicia la etapa b) con respecto a la ronda de generador completada.

Para restringir el número de conjuntos de información aleatoria (r) mantenidos en todos los procesos en ejecución, una de las formas de realización preferidas prevé que se defina un número máximo de colecciones de tuplas y se suprima una colección de tuplas, si se supera el número máximo de colecciones.

Para mejorar la prevención contra fraudes, los procesos en ejecución pueden intercambiar un compromiso criptográfico sobre su información aleatoria (r), antes de que cualquiera de los procesos en ejecución comience a intercambiar su información aleatoria (r) en correspondencia con el proceso de determinar una información aleatoria distribuida para una ronda, con el fin de imponer honestidad en la provisión de dicha información aleatoria (r).

Los procesos en ejecución también pueden identificarse entre sí usando una identidad criptográfica, tal como (aunque no de forma exclusiva) claves criptográficas asimétricas y simétricas.

Descripción detallada de la invención

A continuación, se describirá la invención de manera más detallada en referencia a formas de realización preferidas específicas de la invención.

1. Generación de un número aleatorio distribuido

Lo siguiente describe un protocolo no interactivo usado por n procesos en una red distribuida para generar un número aleatorio distribuido R en una ronda. A intervalos periódicos, cada proceso genera un número aleatorio r y lo envía a la totalidad del resto de procesos. Cada proceso recopila los números aleatorios de la totalidad del resto de procesos y, una vez que se completa la recopilación para una ronda dada, genera un número compuesto R según una primera función de transformación. Este envío y recopilación se denomina ronda de elección de líder. Puesto que la recepción de números aleatorios r para una ronda de elección de líder particular se puede producir en diferentes momentos y desordenadamente, cada proceso implementa un vector $\vec{V}$ de tamaño σ para admitir el procesado de σ rondas simultáneas.

La figura 1 ilustra un conjunto de procesos identificados $P = \{p_1, p_2, \ldots p_n\}$ en una red distribuida, de tal manera que el número de procesos $n = |P|$, en donde λ es la latencia máxima de la red en milisegundos entre todos los

procesos ∈ *P*. Sea ϕ la frecuencia de las rondas de elección de líder, es decir, con qué frecuencia por segundo se inicia una ronda nueva de elección de líder, tal que $\phi \gg \lambda$.

Sea ***e*** ∈ $\mathbb{Z}$ un exponente entero de 2, con ***e*** ≥ 31, e idéntico para todo *p* ∈ *P*. Sea $\mathbb{Z}_o^+$ un cuerpo finito de orden primo *o*, tal que *o* es el número primo más grande en 2^e. Por ejemplo, 65521 es el número primo más grande en 2^{16} y $\mathbb{Z}_o^+$ contendría los enteros {0,1,2, ... 65520}. Se usa un número primo de *Mersenne* de la forma $M_e = 2^e - 1$ para eliminar el exceso de valores aleatorios en $\mathbb{Z}_o^+$ más allá de o, lo cual crearía un sesgo hacia algunos primeros elementos de $\mathbb{Z}_o^+$ con una probabilidad de $\frac{e-o}{2^e}$. Los números primos de *Mersenne* preferidos para usarse con la invención son $2^{31} - 1$ y $2^{61} - 1$. Sea *r* un número aleatorio generado sobre $\mathbb{Z}_o^+$ en el intervalo [2, *o* - 1]. Debido a la función de transformación que se esboza a continuación, no se permiten los elementos 0 y 1.

Sea $f_R(r_1, r_2, \ldots r_n) \rightarrow R$ una primera función de transformación que toma números aleatorios *r* para producir un número aleatorio distribuido *R*. Según la invención, la primera función de transformación se basa en multiplicaciones dentro del campo finito $\mathbb{Z}_o^+$ en forma de $(a \cdot b)_o = (a \cdot b) \ (mod\ o)$ y se define como:

$$f_R(r_1, r_2, \ldots r_n) = \prod_{i=1}^{n} r_i \ (mod\ o)$$

En el siguiente ejemplo, la primera función de transformación se usa para transformar 3 números aleatorios r_1, r_2 y r_3 en un número aleatorio distribuido *R* utilizando la multiplicación en el campo finito. Suponiendo r_1 = 58042, r_2 = 41007, r_3 = 27559, *o* = 65521, se lleva a cabo el cálculo de la siguiente manera:

1. $r_1 \cdot r_2\ (mod\ m) = 58042 * 41007\ (mod\ m) = 12448_{65521}$
2. $12448 \cdot r_3\ (mod\ m) = 12448 \cdot 27559\ (mod\ m) = 51997_{65521}$
3. El número aleatorio distribuido calculado es *R* = 51997

La figura 2 muestra la entropía de la función de transformación en 100 millones de rondas de 5 números aleatorios *r* utilizando la anterior primera función de transformación f_R sobre un campo finito de orden primo de *Mersenne* $\mathbb{Z}_{2^{31}-1}^+$ en un histograma de 10,000 intervalos. Los 500 millones de números aleatorios se generaron utilizando el generador Mersenne Twister 19937.

Cada ronda de generador se define como el intercambio entre procesos de números aleatorios *r* atribuibles a la misma ronda de elección de líder. Las rondas de elección de líder se numeran y se designan por medio de *g*, donde *g* ∈ $\mathbb{Z}$, y comienzan en 1 para cada proceso en el *bootstrap* de la red distribuida.

Sea g_i la ronda actual del generador en el proceso p_i. A intervalos periódicos breves (preferentemente cada 1 segundo) cada proceso $p_i \in P$ inicia una ronda de generador nueva incrementando su contador g_i en uno y generando un número aleatorio r_i en $\mathbb{Z}_o^+$. La tupla (g_i, r_i) se difunde a continuación inmediatamente a la totalidad del resto de procesos de $P \backslash \{p_i\}$. Para indicar el orden consecutivo estricto en el que se deben generar números aleatorios, r_i' se define de manera que sea el número aleatorio de $\mathbb{Z}_o^+$ generado por p_i en la ronda previa $g_i' = g_i - 1$. r_i' precederá estrictamente a r_i, lo cual se indica como $r_i' \preccurlyeq r_i$.

Sea C_g una colección de tuplas (g_n, r_n) dentro de un proceso *p* ∈ *P* para su ronda *g*, donde la tupla (g_n, r_n) representa el valor aleatorio r_n creado por el proceso p_n en la ronda g_n tal como lo recibe el proceso p_i. C_g puede existir o no para una ronda *g* dentro de un proceso *p*. De ahí se deduce que C_g se origina para la ronda *g* en el proceso p_i cuando o bien a) p_i genera un valor aleatorio r_i para la ronda *g* y añade la tupla (g_i, r_i) a su C_g , o bien b) el proceso p_i recibe la tupla (g_n, r_n) del proceso p_n y la añade a su $C_g \mid g = g_n$.

Como ejemplo, la figura 3 muestra la ronda de generador 7 que comprende 3 procesos que se envían tuplas entre ellos en una red distribuida, donde cada proceso p_i genera su propio número aleatorio r_i y recibe tuplas (g_n, r_n) de otros procesos p_n.

Sea $\vec{V}$ un vector de colecciones *C* creado por un proceso de manera que C_n es el elemento en la posición *n* en $\vec{V}$. Sea $\mathbb{r}$ el tamaño máximo de ese vector de manera que $\mathbb{r} \geq |\vec{V}|$. En el proceso p_i la tupla (g_i, r_i) es el número aleatorio r_i generado por el proceso local p_i para la ronda g_i y almacenado dentro de C_{gi} en la posición *k* dentro de $\vec{V}$. El orden de los números aleatorios generados se indica como:

$$r_{i-1} \in C_{i-1} \ \prec \ r_i \in C_i \ \forall\ i \ \in \{1,2, \ldots m\}, m = |\vec{V}|.$$

El siguiente ejemplo muestra el vector de colecciones $\vec{V}$ de un proceso durante 7 rondas:

$\vec{V}$

$C_1\{1,872\}_1\ \{1, 283\}_2\ \{1, 924\}_3$
$C_2\{2,276\}_1\ \{2, 982\}_2\ \{2, 124\}_3$
$C_3\{3,842\}_1\ \{3, 294\}_2\ \{3, 628\}_3$
$C_4\{4,824\}_1\ \{4, 877\}_2\ \{4, 482\}_3$
$C_5\{5,926\}_1\ \{\}_2\ \{\}_3$
$C_6\{6,436\}_1\ \{6, 614\}_2\ \{6, 944\}_3$
$C_7\{\}_1\ \{7, 193\}_2\ \{\}_3$

(1) El proceso p_1 mantiene un vector local $\vec{V}$ que comprende las colecciones C_1, C_2, ⋯ C_n.

(2) C_5 para la ronda 5 en p_1 se origina cuando p_1 crea su valor aleatorio para esa ronda; en este momento la colección no contiene todavía ninguna otra tupla recibida por p_1.

(3) Si un proceso p_2 genera un valor aleatorio para la ronda 5 y envía la tupla (g_5, r_5) = (5, 719) al proceso p_1, el proceso p_1 la añadirá a su C_5.

(4) C_7 se origina en p_1, cuando recibe la tupla (7, 193) de p_2. Cabe señalar que en este momento p_1 no ha generado todavía su propio valor aleatorio para la ronda 7.

Como $\mathbb{r}$ define el tamaño máximo de $\vec{V}$, cuando un proceso añade un conjunto para una ronda nueva a su vector y $|\vec{V}| \geq \mathbb{r}$, en primer lugar, debe eliminar la(s) ronda(s) más antigua(s) para lograr $|\vec{V}| \geq \mathbb{r} - 1$ antes de añadir la ronda nueva.

Como P es el conjunto definido estáticamente de todos los procesos identificados en la red distribuida, cada proceso también mantiene un conjunto ordenado (vector) $\vec{K}$ de procesos conocidos (en ejecución) ∈ P.

Por lo tanto, una ronda de generador g se considera localmente completa para un proceso p_i, cuando $|C_g| = |\vec{K}|$, como en el ejemplo mostrado anteriormente para las rondas C_1, C_2, C_3, C_4, C_6.

C_g^p se define como una colección C para la ronda g en el proceso p. Además, C_g^p se define como una colección C para la ronda g en el proceso q. La igualdad para tuplas se define como $(g_p, r_p) = (g_q, r_q) \mid g_p = g_q \wedge r_p = r_q$. La colección C_g^p para la ronda g en el proceso p es congruente con la colección C_g^p para la ronda g en el proceso q cuando todas las tuplas coinciden:

$$C_g^p \equiv C_g^q \mid (g_k^p, r_k^p) = (g_k^q, r_k^q)\ \forall\ (g_k^p, r_k^p) \in C_p^p,\ \ (g_k^q, r_k^q) \in C_g^p,$$
$$k\ \in \{1,2,\ldots i\}, i = |\vec{K}|.$$

Finalmente, una ronda de generador g se define como totalmente completa, cuando las C_g en todos los procesos p_n son congruentes para una ronda g, es decir:

$$C_g^k \equiv C_g^l\ \mid p_k\ \neq\ p_l\ \forall\ k, l\ \in \{1,2,\ldots i\}, i = |\vec{K}|.$$

Cada vez que una colección C_g correspondiente a un proceso p llega a completarse localmente, el proceso calcula el número aleatorio distribuido R_g aplicando la primera función de transformación previamente definida:

$$f_R(r_1, r_2, \ldots r_n)\ \rightarrow R_g \mid r_i \in\ C_g\ \forall\ i\ \in \{1,2,\ldots i\}, i = |\vec{K}|.$$

Dada la ordenación de los números de las rondas, se deduce que:

$$R_x \preccurlyeq\ R_y \mid x\ <\ y\ \forall\ x, y\ \in \{1,2,\ldots m\} \mid m\ =\ |\vec{V}|.$$

2. *Bootstrapping* y cuórum

A continuación, se describirá un método preferido sobre cómo hacer *bootstrap* en una red distribuida de procesos de manera que lleguen a un cuórum, y cómo los procesos llegan a considerar que una ronda de elección de líder se ha completado localmente con el fin de dar comienzo al funcionamiento normal. Igual que antes, un conjunto completo estático de procesos identificados se designa *P*.

Sea *q* el cuórum mínimo necesario para ejecutar una ronda de elección de líder, y sea $n = |P|$ y $P_q \subset P$ un subconjunto con cuórum de *P*, donde $|P_q| = q, q < |P|$. Además, P_0 se define como el subconjunto resto $P_0 = P \backslash P_q$ de procesos que se incorporan a la red distribuida en un momento posterior, y limitan P_0 con $P_q \cap P_0 = \emptyset$.

El ejemplo mostrado en la figura 4 Ilustra un cuórum y 2 subconjuntos resto. La figura 4 ilustra un total de 5 procesos |P| = *n* = 5.

- El cuórum *q* se define como una mayoría simple de 3 de entre 5 procesos (círculo interior).
- Los conjuntos resto definen los dos procesos p_4 y p_5 que se incorporan más tarde.
- El círculo exterior ilustra un conjunto de futuras bajas compuesto por un proceso p_5 que podría abandonar el conjunto de procesos posteriormente.

En el contexto de la invención, se pueden aplicar diferentes tipos de cuórum. Un cuórum de mayoría simple se define como $q_{simple} = \frac{n}{2} + 1$. No obstante, para lograr una tolerancia a fallos bizantinos, se puede usar un cuórum de mayoría bizantina en lugar de un cuórum de mayoría simple. A este respecto, *f* será el número máximo de nodos deficientes tolerados y la relación entre *n* y *f* se limitará de manera que sea *n* = 3*f* + 1. Un cuórum de mayoría bizantina se deduce usando la fórmula de la mayoría simple e ignorando los *f* procesos deficientes, y se define:

$$q_{byzantine} = \frac{n}{2} + f + 1$$

Ejemplo de cálculo para el número de nodos y sus tipos de cuórum, para las tuplas: (f, n, q_{simple}, $q_{byzantine}$)

(1, 4, 3, 4)
(2, 7, 4, 6)
(3, 10, 6, 9)
(4, 13, 7, 11)
(5, 16, 9, 14)
(6, 19, 10, 16)
(7, 22, 12, 19)
(8, 25, 13, 21)
(9, 28, 15, 24)

Para cada proceso se pueden definir los siguientes estados: incorporación, ejecución, saliente, fallido. Cuando un proceso se inicia por primera vez, su estado es “incorporación”. Inicializa su ronda en curso a *g* = 1 y comienza a crear y emitir tuplas (*g*, *r*) hacia otros procesos ∈ *P*. Estos otros procesos pueden existir y, si existen, finalmente reciben la tupla enviada.

Cuando un proceso p_i en estado “incorporación” recibe una tupla $(g_k, r_k)_m$ del proceso p_m para la ronda *k*, y $g_k > g_i$, donde g_i es la ronda en curso del proceso, el proceso adopta el número de ronda más alto, añade $(g_k, r_k)_m$ a su colección C_k, genera inmediatamente un valor aleatorio para la ronda *k* y envía su tupla $(g_k, r_k)_i$ a la totalidad del resto de procesos ∈ *P*. C_k cumple el cuórum en el proceso p_i, cuando $|C_k| \geq q$, y en tal caso el proceso cambia al estado “ejecución”.

La figura 5 es una ilustración del proceso p_1 que se incorpora al subconjunto con cuórum ⊂ *P*, donde *q* = 3.

(1) El proceso p_1 comienza en el estado “incorporación”.

(2) p_1 comienza a emitir tuplas, a partir de la ronda *r* = 1.

(3) p_1 recibe una tupla de p_2 para la ronda 14 y la añade a su C_{14} local.

(4) Puesto que esta ronda 14 está más adelantada que su ronda local, p_1 adopta esta ronda, genera un valor aleatorio para ella y envía la tupla a otros procesos ∈ *P* (que podrían no estar en ejecución en ese momento). También puede abandonar de forma segura todas las rondas creadas previamente $C_k \mid k < 14$.

(5) A continuación p_1 recibe de p_2 y p_3 tuplas correspondientes a la ronda 17. Se puede concluir que p_2 y p_3 de alguna manera intercambiaron tuplas en paralelo y realizaron un *bootstrap* hasta la ronda 17 como un conjunto de dos sin alcanzar un cuórum mientras p_1 todavía estaba ocupado arrancando.

(6) p_1 genera un valor aleatorio para la misma y envía la tupla a otros procesos $\in P$ de entre los cuales ya tiene p_2 y p_3 en su vector $\vec{K}$.

(7) Después de que p_1 añada su tupla para la ronda 17 a su colección correspondiente a esa ronda, entonces C_{17} llega a completarse localmente. p_1 también puede suponer legítimamente que p_2 y p_3 podrán añadir en breve su tupla a su C_{17} consiguiendo que la misma también se complete localmente.

(8) Habiendo determinado su primera ronda completa localmente, p_1 cambia al estado "ejecución" y continúa con el funcionamiento normal. Asimismo se puede concluir que p_2 y p_3 cambiarán también al estado "ejecución" si no estuvieran ya en ese estado. Esto dará como resultado un estado de compleción global para la red distribuida, por lo que se puede considerar lista y preparada para solicitudes de los clientes.

3. Incorporación de un proceso

El método para incorporar un proceso p_{new} a los vectores ordenados locales de procesos conocidos (en ejecución) $\vec{K}$ en una red distribuida es similar al proceso de *bootstrap* antes descrito. En la medida en la que p_{new} se incorpora en algún instante de tiempo posterior, $|\vec{K}|$ tendrá por lo menos el tamaño de cuórum mínimo definido para la red en los otros procesos en ejecución. Cuando se incorpore el nuevo proceso, el mismo se añadirá al vector de procesos conocidos $\overrightarrow{K'} = \vec{K} \cup \{p_{new}\}$. Para permitir la adición y sustracción de procesos de los vectores $\vec{K}$ respectivos en cada proceso, cuando se envían tuplas cada proceso también envía conjuntamente su vector $\vec{K}$ de todos los procesos conocidos. Cada vez que un proceso nuevo se inicia y comienza a enviar sus tuplas, también empezará a recibir tuplas de otros procesos y construirá su $\vec{K}$ fusionando su vector ordenado local con el recibido $\overrightarrow{K'}_{local} = \vec{K}_{local} \cup \vec{K}_{received}$. Los otros procesos añadirán este proceso nuevo a su $\vec{K}$ y cada proceso de la red distribuida, a partir de la siguiente ronda, basará sus consideraciones en $\overrightarrow{K'}$.

Como ejemplo, la figura 6 es una ilustración de procesos p_1 que se incorporan a p_2 y p_3, y a continuación p_4 se incorpora a los tres.

(1) Los procesos p_2 y p_3 son los procesos iniciales que se ejecutan en un conjunto de dos. El cuórum se define como q = 2. Intercambian sus vectores $\vec{K} = \{2,3\}$ entre sí. El tamaño en curso del conjunto es $|\vec{K}| = 2$.

(2) El proceso p_1 se inicia y se incorpora a la red distribuida.

(3) p_1 envía su vector $\vec{K} = \{1\}$ junto con sus tuplas a otros procesos, todavía desconocidos (en ejecución o no), de la red.

(4) p_2 se entera de la existencia de p_1 y lo añade a su $\vec{K}$. Envía el vector nuevo en el mensaje de la siguiente tupla.

(5) p_1 obtiene una respuesta de p_2 y fusiona su vector local con el vector que envió conjuntamente p_2.

(6) p_3 recibe una tupla de p_1 y fusiona su vector local con el vector que envió conjuntamente p_1.

(7) En todos los mensajes los procesos ahora tienen incluido el vector nuevo $\overrightarrow{K'} = \{1,2,3\}$. Ahora toda la red distribuida tiene $|\overrightarrow{K'}| = 3$ como criterio de compleción local.

(8) El proceso p_4 se inicia y envía su vector $\vec{K} = \{4\}$ conjuntamente en sus mensajes a otros procesos, todavía desconocidos, de la red distribuida.

(9) p_3 se entera de la existencia de p_4 y fusiona el vector enviado conjuntamente con su propio vector $\overrightarrow{K'} = \{1,2,3\} \cup \{4\}$.

(10) Envía conjuntamente este vector actualizado en su mensaje de la siguiente tupla de manera que los procesos p_1 y p_2 también fusionen sus vectores.

(11) Los procesos p_1, p_2, p_3 ahora reconocen a p_4 como miembro nuevo en P y usan su vector actualizado a partir del mensaje de la siguiente tupla en adelante.

(12) p_4 encuentra el vector completo en los mensajes recibidos y se fusiona con su vector local. Ahora todos

los procesos saben que la red distribuida ha crecido hasta 4 miembros y usan su $\vec{K}, |\vec{K}| = 4$ actualizado a partir de ahora en adelante para desencadenar la compleción local.

4. Procesos salientes

Los procesos ∈ *P* pueden decidir dejar de funcionar en cualquier momento. Cuando un proceso p_s se detiene, envía el mensaje *leave*(p_s, r_s) a la totalidad del resto de procesos, donde r_s es la siguiente ronda que usaría normalmente p_s para enviar tuplas. p_s también deja de enviar y recibir tuplas.

Cuando un proceso p_i recibe un mensaje *leave*(p_s, r_s), saca p_s de su vector local $\vec{K}$ de procesos en ejecución conocidos y comprueba si sigue estando en vigor el número mínimo de procesos con cuórum. En caso de que el número de procesos en ejecución restantes fuera menor que el cuórum requerido mínimo, el proceso cambia de nuevo al estado de *bootstrap*, pero en este caso no reinicializará su número de ronda a 1.

La figura 7 muestra un ejemplo de proceso p_1 que abandona el conjunto de p_1, p_2, p_3

(1) Los procesos p_1, p_2, p_3, p_4, p_5 funcionan en modo normal “ejecución” con un cuórum q = 3.

(2) Los procesos intercambian tuplas durante 18 rondas, enviando conjuntamente su vector ordenado de procesos en ejecución conocidos $\vec{K} = \{p_1, p_2, p_3, p_4, p_5\}$.

(3) p_1 decide dejar de ejecutarse. En ese momento está en la ronda 18, por lo que envía el mensaje *leave*(1,19) a la totalidad del resto de procesos de *P*. Este es el último mensaje que envía p_1 a la red.

(4) Los procesos que están en ese momento en ejecución p_2, p_3, p_4, p_5 eliminan la baja de su vector $\overrightarrow{K'} = \vec{K} \backslash \{p_1\}$.

(5) p_2, p_3, p_4, p_5 comprueban si siguen teniendo cuórum. Dado que q = 3 sí lo tienen, y por tanto continúan con su funcionamiento normal, ahora usando $\vec{K} = \{p_2, p_3, p_4, p_5\}$ junto con sus tuplas.

5. Procesos fallidos

Los procesos pueden fallar sin opciones de enviar un mensaje *leave*. En esta sección se presenta un método preferido para detectar procesos fallidos y eliminarlos del conjunto de procesos que están en ejecución en ese momento, lo cual afecta a la determinación de compleción local para las rondas de elección de líder.

Sea $p_i \in P$ un proceso en ejecución que recoge tuplas recibidas para la ronda k en su colección local C_k. Sea $\vec{V}$ su vector de colecciones C y defina $\mathbb{r}$ el tamaño máximo de $\vec{V}$. Un proceso p_a se considera en vigor para la colección C_k, si $(g_k, r_k)_a \in C_k$ y, en cualquier otro caso, se considera que un proceso ha fallado para C_k.

Sea $\vec{A} \subseteq \vec{V}$ el conjunto de colecciones en vigor en $\vec{V}$ para un proceso p_f. Se define que el proceso p_f falla cuando $|\vec{A}| \ll \mathbb{r} \wedge |\overrightarrow{V}| = \mathbb{r}$. Se puede deducir que la detección de fallos únicamente puede estar disponible una vez que los procesos han pasado por $\mathbb{r}$ rondas en estado de ejecución.

Cuando un proceso p_i detecta un proceso fallido p_f en el inicio de una ronda nueva, envía el mensaje *fail*(g_i,p_f) a la totalidad del resto de procesos.

Cuando un proceso p_j recibe *fail*(g_j ,p_f), comprueba la condición $|\vec{A}| \ll \mathbb{r} \wedge |\overrightarrow{V}| = \mathbb{r}$ para su $\vec{V}$ local. Si la condición se cumple, envía *confirm*(g_j,p_f) a la totalidad del resto de procesos. Si la condición no se cumple, envía *alive*(g_j,p_f) a la totalidad del resto de procesos.

Cuando el proceso p_i recibe un mensaje *alive*(g_j,p_f) de cualquier otro proceso, continúa con el funcionamiento normal y deja de emitir mensajes *fail*. Se puede concluir que p_i debe haberse perdido algunos mensajes de p_f en el pasado, y su $\vec{K}$ ó $\vec{V}$ local podría no ser congruente con los otros procesos. En particular, esto puede producirse cuando la red se segmenta, lo cual se aborda más adelante.

Un proceso p_i recopila mensajes *confirm*(g_k,p_f) correspondientes a un *fail*(p_i,p_f) enviado previamente hasta que haya recopilado confirmaciones de todos los procesos conocidos ∈ $\vec{K} \backslash \{p_f\}$. A continuación elimina p_f de su $\vec{K}$. p_i sigue enviando *fail*(g_j ,p_f) en cada ronda hasta que o bien reciba un mensaje *alive* o bien pueda eliminar finalmente p_f de su $\vec{K}$. Después de la eliminación de p_f, los procesos deben verificar si siguen teniendo cuórum o, de lo contrario, volverán de nuevo al estado de *bootstrap*.

La figura 8 ilustra un ejemplo de proceso p_1 fallido, y su detección y eliminación por parte de otros.

(1) Los procesos p_1, p_2, p_3, p_4, p_5 forman un conjunto de procesos en ejecución con un cuórum $q = 3$.

(2) Los procesos intercambian mensajes de tupla durante el funcionamiento normal.

(3) El proceso p_1 falla. Los otros procesos dejan de recibir mensajes de tupla de él.

(4) p_2 observa que se cumple $|\vec{A}| \ll \mathbb{r} \wedge |\overrightarrow{V}| = \mathbb{r}$ para p_1 y envía *fail*(23,1) a los otros procesos (23 es su ronda en curso, 1 el proceso).

(5) p_2 obtiene la respuesta *confirm*(24,1) de p_3 y p_4. Registra el mensaje y puede ignorar la diferencia en el número de ronda.

(6) p_2 obtiene la respuesta *confirm*(24,1) de p_5 y, finalmente, ahora ha recopilado las retroalimentación completa.

(7) A continuación p_2 elimina p_1 de su vector ordenado de procesos en ejecución $\vec{K}$. Dada la simetría del intercambio de mensajes, se concluye que los otros procesos han pasado por el mismo flujo de trabajo.

(8) Puesto que p_2, p_3, p_4, p_5 siguen teniendo cuórum, pueden continuar en estado de ejecución.

6. Segmentación de la red

Una partición de red se define como una división de la red debido a algún fallo de conexiones en la red entre procesos. La siguiente sección describe un comportamiento tolerante a particiones preferido, de subredes, para el método de elección de líder.

La figura 9 ilustra un ejemplo de una red distribuida configurada para 5 procesos y un cuórum $q = 3$.

(1) Algunos componentes fallan de manera que la comunicación se corta parcialmente y la red se divide en dos segmentos en los lados izquierdo y derecho de la barra negra. Las líneas de puntos representan los canales de comunicación interrumpidos.

(2) Los procesos p_1, p_4, p_5 están en la primera subred. Permanecen en el estado “ejecución” ya que forman un cuórum de 3.

(3) Los procesos p_2, p_3 forman la segunda subred. Puesto no pueden tener cuórum, cambian su estado a “bootstrap”.

Sea P el conjunto de todos los procesos y q un cuórum simple, tal como se ha usado anteriormente. Sea P_1 el conjunto de procesos de la subred 1 y P_2 el conjunto de procesos de la subred 2. Una partición de red se define como la formación de subconjuntos exclusivos $P_k \subset P$, de manera que $P_i \cap P_j = \emptyset \mid i \neq j \; \forall i,j \; \in \{1,2,\dots k\}$.

Sea $n_1 = |P_1|$ y $n_2 = |P_2|$ de una red distribuida segmentada de 2 segmentos. La mayoría simple para formar cuórum es $q = \frac{n_1+n_2}{2} + 1$, donde $n_1 + n_2 = |P|$. Puesto que el cuórum q debe ser mayor que la mitad del número total de procesos de manera que $q > \frac{n_1+n_2}{2}$, se puede deducir que o bien $n_1 \geq q$ o $n_2 \geq q$ ó bien ninguno de los dos lo es. Se puede generalizar: sean $S = \{P_1, P_2, \cdots P_k\}$ subconjuntos exclusivos de P y $n_i = |P_i| \mid \forall \; P_i \in S$. Entonces $q = \frac{n_1+n_2+\cdots n_k}{2} + 1$ y o bien exactamente un segmento forma un cuórum $n_i \geq q$, $n_j < q \mid i \neq j \; \forall \; j \in \{1,2, i - 1, i + 2, \dots |S|\}$ ó bien ningún segmento lo hace $n_j < q \; \forall \; j \in \{1,2, \dots |S|\}$.

Se puede concluir que siempre que una red se segmenta en 2 o más partes, un máximo de 1 parte puede permanecer en el estado “ejecución”.

La figura 10 ilustra un ejemplo de la segmentación de una red durante el funcionamiento normal.

(1) Una red distribuida de 5 procesos está en ejecución con $q = 3$.

(2) Todos los procesos intercambian tuplas.

(3) Se produce una segmentación de la red, que divide el conjunto de procesos en un segmento izquierdo p_1, p_4, p_5 y uno derecho p_2, p_3.

(4) Los procesos del segmento izquierdo ven que p_2 y p_3 fallan después de algunas rondas y llevan a cabo el procedimiento para procesos fallidos descrito anteriormente. p_2 y p_3 ven que los procesos p_1, p_4, p_5 fallan y

ejecutan el mismo flujo de trabajo.

(5) En los segmentos izquierdo y derecho, los procesos confirman mutuamente su visión sobre los procesos fallidos.

(6) Cuando se ha producido la confirmación, los procesos de ambos segmentos eliminan de su $\vec{K}$ los procesos que han fallado.

(7) Durante la eliminación de los procesos que han fallado, los procesos del segmento izquierdo siguen formando un cuórum de 3 y mantienen su estado “ejecución”, mientras que los procesos del segmento izquierdo pierden el cuórum y cambian al estado “bootstrap”.

(8) Los procesos del segmento derecho continúan enviándose entre sí mensajes de *bootstrap*. Envían estos mensajes a todos los procesos ∈ *P*. En algún momento posterior, cuando finalice la partición y los segmentos se vuelvan a juntar, los procesos del primer segmento izquierdo recibirán procesos de *bootstrap* del segmento derecho, y todos los procesos se añadirán mutuamente de nuevo a su $\vec{K}$.

7. Determinación de un líder que actúa como maestro de transacciones

En cualquier momento dado, una red distribuida tendrá $\vec{K}$ procesos en ejecución conocidos. Una solicitud es un mensaje proveniente de un proceso externo en la misma red (tal como un cliente, véase más adelante) a procesos ∈ $\vec{K}$. Para cada solicitud se determina un proceso maestro, utilizando rondas de elección de líder para una elección de líder según se ha descrito anteriormente.

Sea *request*(*m,r,data*) un mensaje de solicitud enviado por un proceso externo, donde *m* ∈ {1,2, … $|\vec{K}|$} se designa como “maestro” y especifica un índice en $\vec{K}$, *r* es la ronda de elección del líder y *data* indica algún conjunto de datos estructurado suministrado por el cliente, para su procesado por parte del líder.

El líder se selecciona calculando un designador de uno solo de los procesos en ejecución a partir de la información aleatoria distribuida por medio de una segunda función de transformación compartida que, según la presente invención, se define como $m = \left(R_r\left(mod|\vec{K}|\right)\right) + 1$.

Un proceso p_i que recibe un mensaje *request* determina si es el maestro para dicha solicitud comprobando si se trata del elemento m$^{\text{ésimo}}$ en su $\vec{K}$. Verifica $p_i = \vec{K}[m]$ y $m = \left(R_r\left(mod|\vec{K}|\right)\right) + 1$. Si la primera ecuación coincide, p_i es el destinatario de *request*; si coincide la segunda ecuación, p_i es legítimamente el líder para la ronda *r*. Si $p_i \neq \vec{K}[m]$, entonces p_i puede ignorar de manera segura *request*. Para $m = \left(R_r\left(mod|\vec{K}|\right)\right) + 1$ son posibles dos condiciones de error:

1) Si los lados izquierdo y derecho de la ecuación no coinciden, entonces *request* se dirigió a p_i bajo suposiciones erróneas por parte del proceso externo (error, ataque adversario). En este caso, p_i devuelve el mensaje de rechazo *error*(*notmaster*) al proceso externo.

2) si R_r o K_r no existe en p_i, entonces el proceso externo podría estar usando una ronda que es demasiado avanzada, demasiado antigua o podría referirse a una ronda que está incompleta. En ese caso p_i devuelve el mensaje de rechazo *error*(*noround*) al proceso externo.

La figura 11 ilustra un proceso externo que envía solicitudes a una red distribuida.

(1) Los procesos $p_1 \cdots p_5$ forman una red distribuida con cuórum $q = 3$. Tienen conocimiento de las rondas 6,7,8 y mantienen los respectivos valores aleatorios distribuidos R_k y $\overrightarrow{K_k}$.

(2) Un proceso externo envía una solicitud a la red, dirigiéndose a p_1 en la ronda 7, que verifica que su posición en $\overrightarrow{K_7}$ se encuentra de hecho en el índice 1. Adicionalmente p_1 verifica $R_7\left(mod|\overrightarrow{K_7}|\right) + 1 = 860\ (mod\ 5) + 1 = 1$. Se cumple esto, por lo que p_1 puede procesar la solicitud.

(3) El proceso externo envía una solicitud a la red, dirigiéndose a p_1 en la ronda 8, que verifica que su posición en $\overrightarrow{K_8}$ se encuentra de hecho en el índice 1. Adicionalmente, p_1 comprueba $R_8\left(mod|\overrightarrow{K_8}|\right) + 1 = 131(mod\ 5) + 1 = 2 \neq 1$. Por lo tanto p_1 devuelve un error.

(4) El proceso externo envía una solicitud a la red, dirigiéndose a p_1 en la ronda 9 el cual no puede comprobar su posición puesto que $\overrightarrow{K_9}$ todavía no existe en p_1. Por lo tanto p_1 devuelve un error. p_2 no puede comprobar su posición tampoco, pero debido a que su última posición conocida en $\vec{K}_{max} = \overrightarrow{K_8}$ era 2 rechazará la solicitud.

8. Clientes

Un proceso-cliente (cliente) es un proceso externo que envía solicitudes a procesos en ejecución, es decir, procesos que participan en la elección del líder. Un maestro es el líder dentro de la red distribuida, responsable de la solicitud de un proceso-cliente, y se espera que envíe una respuesta al cliente.

En este ejemplo, los clientes se incorporan a la red distribuida y adoptan un papel de solo lectura en las rondas de elección de líder, es decir, tienen sus propios vectores $\vec{V}$ y $\vec{K}$ junto con las colecciones $C_k \in \vec{V}$ para tener un conocimiento continuado sobre procesos, rondas, números aleatorios distribuidos, altas y bajas en curso. Así, un cliente podrá determinar el maestro para cada ronda.

Cuando un cliente prepara una solicitud para su envío, comienza con la última ronda localmente completa g de $\vec{V}$ y calcula el maestro m a partir del número aleatorio distribuido R_g como $m = R_g\left(mod\left|\overrightarrow{K_g}\right|\right) + 1$. A continuación usa *request*(*m,g,data*) para enviar su mensaje o bien a todos los procesos en la red distribuida, o bien directamente al proceso maestro $p_n = \vec{K}_g[m]$.

Cuando se usa el modo de difusión, los nodos que no son maestros – al tiempo que ignoran la solicitud maestra en el nivel de elección de líder – podrían lanzar una funcionalidad de nivel superior dentro de un proceso de optimización, por ejemplo, los procesos que no son maestros pueden enviar su voto sobre un consenso al maestro inmediatamente, ahorrándole al maestro la emisión de la solicitud en primer lugar.

Cuando un cliente honesto recibe un error o no recibe ninguna respuesta antes de un tiempo límite determinado, crea una *request* nueva utilizando los datos de una ronda localmente completa previa: *request*(m_i,*i,data*), donde $i = max(j) \mid j < g \ \wedge locallyComplete(C_j) = 1 \wedge C_j \in \vec{V}$ hasta que se tenga éxito o se haya probado la ronda más antigua dentro de $\vec{V}$. Si todos los intentos siguen siendo erróneos, el cliente debe ceder.

Interpretación de condiciones de error para el cliente:

error(*notmaster*) el $\vec{V}$ y $\vec{K}$ locales parecen no estar sincronizados con la red distribuida, probar con la ronda anterior.

error(*noround*) el $\vec{V}$ local resulta estar más avanzado que el $\vec{V}$ en el p_m maestro, por lo que probar con la ronda anterior podría tener más éxito.

tiempo límite cuando el cliente no ha obtenido ninguna respuesta del maestro p_m, se puede suponer que p_m probablemente está fallando y prueba con la ronda anterior.

La figura 12 ilustra un proceso de cliente que envía una solicitud utilizando una ronda demasiado avanzada.

(1) Los procesos $p_1 \cdots p_5$ forman una red distribuida con cuórum $q = 3$. Tienen conocimiento de las rondas 6,7,8 y mantienen los respectivos valores aleatorios distribuidos R_k y $\overrightarrow{K_k}$.

(2) El cliente envía una solicitud para la ronda 9, pero puesto que la ronda 9 todavía no se conoce o no se ha completado localmente en p_1, el cliente recibe de vuelta un *error*(*noround*).

(3) El cliente puede decrementar su ronda puesto que $C_8 \in \vec{V}$ y prueba nuevamente con esa ronda.

9. Fragmentación [*sharding*] del maestro

Según otro ejemplo de una forma de realización de la invención, las solicitudes para una red distribuida se pueden fragmentar de manera que existan varios maestros en la misma ronda de elección de líder. Para lograr esto, los mensajes de cliente deben usar conjuntos de datos estructurados y un elemento de ese conjunto de datos debe ser $\in \mathbb{Z}_o^+$. Para impedir que el cliente seleccione contenido arbitrario para manipular al maestro a su favor, ese elemento debe o bien seguir algún orden o bien tener un significado intrínseco para las rutinas de nivel superior asociadas a la solicitud, como un número de serie, tal como un UUID versión 1 con incremento de tiempo monótono, o un número de cuenta con el que está vinculada una solicitud en particular.

Usando el elemento $\varepsilon \in \mathbb{Z}_o^+$ de un conjunto de datos de un cliente, ε se multiplica por el número aleatorio distribuido de la ronda real R_g en el campo finito $\mathbb{Z}_o^+$ para obtener el maestro:

$$m = ((R_g \cdot \varepsilon)\ (mod\ o))\ (mod\ |\vec{K}|)\ + 1$$

10. Forma de realización preferida de la invención cuando se usa en sistemas de difusión de orden total

La complejidad del procesado de mensajes de orden arbitrario en sistemas distribuidos se reduce considerablemente al basarse en primitivas de comunicación de grupo que proporcionan mayores garantías que la comunicación de multidifusión estándar. Una de estas primitivas se denomina difusión de orden total. De manera informal, la primitiva garantiza que mensajes enviados a un conjunto de procesos sean recibidos en el mismo orden por cada miembro del conjunto. Además, cuando se diseña una variante del método inventivo de elección de líder, se requiere que estos sistemas de difusión de orden total entreguen cada mensaje de forma fiable y exactamente una vez a todos y cada uno de los procesos. Por ello, en este tipo de sistema de difusión de orden total, los números aleatorios generados por los procesos en ejecución son recibidos por todos los procesos en el mismo orden, intercalándose mensajes de transporte de números aleatorios entre el flujo continuo totalmente ordenado de mensajes arbitrarios.

Como se ha descrito anteriormente, cada proceso genera un número aleatorio nuevo a intervalos regulares, pero esta vez lo entrega al sistema de difusión de orden total. Para garantizar que un proceso usa su propio número aleatorio en la posición correcta dentro del orden total, ese proceso debe colocar su propio número aleatorio en su conjunto de colecciones local en el momento que lo recibe de vuelta del sistema de difusión de orden total.

Para esta variante de la invención, cada proceso mantiene exactamente un conjunto de colecciones de números aleatorios con elementos de todos los procesos en ejecución. Cuando un proceso realiza un *bootstrap*, espera hasta que su conjunto de colecciones se complete localmente y a continuación cambia al modo de funcionamiento normal.

Cada vez que un proceso p_i recibe un r_j aleatorio del proceso p_j, sustituye el elemento en curso de su conjunto de colecciones en la posición j por r_j.

Como opción, antes de procesar cualquier mensaje sucesivo recibido, p_i calcula un nuevo número aleatorio distribuido aplicando la primera función de transformación $f_R(r_1,r_2, \ldots r_n) \mapsto R$. Puesto que todos los procesos llevan a cabo el mismo cálculo en el mismo orden, se considera que R se completa totalmente con respecto al conjunto de procesos P. Como consecuencia, cada proceso $\in P$ conoce el líder en curso en cualquier posición del flujo continuo de mensajes. Se puede deducir que no hay ningún requisito para rondas de elección de líder independientes.

Como opción alternativa, para cada solicitud de cliente recibida se determina un líder utilizando una primera función de transformación ampliada $f_R(r_1,r_2, \ldots r_n,C) \mapsto R$ con el fin de generar un valor aleatorio por solicitud, donde C indica algunos datos tomados de la solicitud de cliente, por ejemplo su firma. Además de la multiplicación de enteros en un campo de orden primo como se ha explicado anteriormente, para generar R se pueden usar otras funciones de transformación tales como (sin carácter limitativo) concatenación, xor y *hashing*, y una combinación de las mismas. Sea l el número máximo de bits necesarios para representar todos los procesos (por ejemplo, 8 bits = 256 procesos) y sea L la longitud de R en bits. Se designa que β es la máxima probabilidad de sesgo hacia los primeros i procesos de P en forma de $\beta = 1 / 2^{(L-l)}$, donde $i = 2^l \ (mod\ |P|)$. Para seguir la recomendación de la página 46 de la Norma Federal de Procesado de Información [Federal Information Processing Standard] 186-4 del Instituto Nacional de Normas y Tecnología (NIST) con vistas a trabajar con la probabilidad de sesgo en un cálculo con módulo bajo, con $i > 0$, se recomienda utilizar una función de transformación que da como resultado R con $L \geq l + 64$.

Puesto que la elección de líder en estos sistemas de difusión de orden total garantiza que R se complete totalmente cada vez que se reciba cualquier otro mensaje, se deduce que ya no se aplica más el requisito de que un cliente se involucre en los mensajes de elección de líder y que seleccione una ronda y un maestro. Los procesos de cliente envían su mensaje al sistema de difusión de orden total, en lugar de a un proceso en ejecución particular (o muchos). El maestro correspondiente a la solicitud se retransmite a los clientes al producirse la recepción de los mensajes por parte del líder apropiado.

REIVINDICACIONES

1. Método implementado por ordenador para la elección, basada en procesos aleatorios, de un líder en una red distribuida de dispositivos de procesado de datos, comprendiendo dicha red distribuida una pluralidad de procesos asíncronos identificados, en el que la totalidad de dichos procesos identificados o un subconjunto de los mismos son procesos en ejecución que participan en la elección de líder, comprendiendo dicho método las siguientes etapas:

a) una información aleatoria (r) es generada por cada proceso en ejecución y compartida con los otros procesos en ejecución, de manera que cada proceso en ejecución mantenga un conjunto de dicha información aleatoria (r),

b) una información aleatoria distribuida (R) es calculada por cada proceso en ejecución a partir del conjunto de información aleatoria (r) aplicando una primera función de transformación compartida (f_1), de manera que se ponga a disposición de cada proceso en ejecución la misma información aleatoria distribuida (R),

c) un designador de uno solo de dichos procesos en ejecución es calculado a partir de la información aleatoria distribuida (R) por medio de una segunda función de transformación compartida (f_2),

d) un líder es elegido entre dichos procesos en ejecución sobre la base de dicho designador,

en el que cada proceso en ejecución mantiene un conjunto ordenado (K) de todos los procesos en ejecución y el número total (k) de procesos en ejecución,

caracterizado por que

- la primera función de transformación compartida es

$R = \prod_{i=1}^{n} r_i (mod\ o)$,

en la que

R es la información aleatoria distribuida,

r_i es la información aleatoria,

mod designa la operación módulo, y

o es un número primo de *Mersenne* definido como $o = 2^n-1$.

- la segunda función de transformación compartida (f_2) se define como

m = R (mod k) o *m = R (mod k + 1)*,

en la que

m es el designador del líder elegido,

R es la información aleatoria distribuida,

k es el número total de procesos en ejecución, y

mod designa la operación módulo,

siendo el líder elegido al seleccionar el proceso en ejecución que se corresponde con el elemento $m^{ésimo}$ en dicho conjunto ordenado (K) de procesos en ejecución.

2. Método según la reivindicación 1, en el que la secuencia de etapas a) - d) se repite a intervalos regulares o irregulares para cambiar aleatoriamente el líder elegido.

3. Método según la reivindicación 1 o 2, en el que cada proceso en ejecución mantiene información sobre el número total (n) de todos los procesos identificados y verifica si el número total (k) de procesos en ejecución se corresponde con un cuórum predefinido del número total (n) de todos los procesos identificados, en el que las etapas b), c) y/o d) se llevan a cabo únicamente si se cumple el cuórum.

4. Método según una cualquiera de las reivindicaciones 1 a 3, en el que el conjunto ordenado (K) de procesos en

ejecución se actualiza para incluir un proceso que se incorpora al grupo de procesos en ejecución, en el que cada proceso en ejecución, incluyendo el proceso de unión, en la etapa a), comparte su conjunto ordenado (K) de todos los procesos en ejecución con los otros procesos y el conjunto ordenado (K) mantenido en cada proceso en ejecución se fusiona con el conjunto ordenado (K) compartido.

5. Método según una cualquiera de las reivindicaciones 1 o 4, en el que el conjunto ordenado (K) de procesos en ejecución se actualiza para eliminar un proceso que abandona el grupo de procesos en ejecución, en el que el proceso saliente envía un mensaje de salida que comprende un identificador de proceso a los otros procesos en ejecución, y el proceso saliente es eliminado del conjunto ordenado (K) de procesos en ejecución.

6. Método según una cualquiera de las reivindicaciones 1 a 5, en el que el conjunto ordenado (K) de procesos en ejecución se actualiza para eliminar un proceso en ejecución fallido, en el que

- cada proceso identifica que no ha recibido ninguna información aleatoria (r) que está siendo compartida por el proceso fallido,
- cada proceso envía un mensaje de fallo a todos los procesos en ejecución restantes consultando si se ha identificado el proceso fallido en el proceso restante, y
- se elimina el proceso fallido del conjunto ordenado (K) de procesos en ejecución al producirse la recepción de unos mensajes de confirmación procedentes de todos los procesos en ejecución restantes.

7. Método según una cualquiera de las reivindicaciones 1 a 6, en el que compartir la información aleatoria (r) en la etapa a) comprende las etapas siguientes:

- cada proceso en ejecución entrega su información aleatoria (r) a un sistema de difusión de orden total,
- el sistema de difusión de orden total difunde la información aleatoria (r) recibida de todos los procesos en ejecución a cada proceso en ejecución en el mismo orden.

8. Método según una cualquiera de las reivindicaciones 1 a 7, en el que compartir la información aleatoria (r) en la etapa a) comprende las etapas siguientes:

- cada proceso en ejecución asigna un identificador de ronda de generador (g) a la información aleatoria generada (r) para obtener una tupla (r,g) que consiste, cada una de ellas, en una información aleatoria (r) y un identificador de ronda de generador (g),
- cada proceso en ejecución envía la tupla a la totalidad del resto de procesos en ejecución,
- cada proceso en ejecución recopila las tuplas recibidas de los otros procesos en ejecución, para obtener dicho conjunto de información aleatoria (r), que adopta la forma de una colección de tuplas que consiste en tuplas (r, g) que tienen el mismo identificador de ronda de generador (g),
- comparar el número de tuplas en dicha colección de tuplas con el número total (k) de procesos en ejecución;

y la etapa b) se inicia si el número de tuplas en la colección local es igual al número total (k) de procesos en ejecución.

9. Método según una cualquiera de las reivindicaciones 1 a 8, en el que compartir información aleatoria (r) en la etapa a) comprende las etapas siguientes:

- cada proceso en ejecución asigna un identificador de ronda de generador (g) a cada información aleatoria generada (r) para obtener unas tuplas (r,g) que consisten, cada una de ellas, en una información aleatoria (r) y un identificador de ronda de generador (g),
- cada proceso en ejecución envía las tuplas directamente a todos los otros procesos en ejecución,
- cada proceso en ejecución recopila las tuplas recibidas de los otros procesos en ejecución, para obtener conjuntos de información aleatoria (r), que adoptan la forma de colecciones de tuplas, consistiendo cada colección en unas tuplas (r,g) que tienen el mismo identificador de ronda de generador (g),
- una ronda de generador se marca como localmente completa si el número de tuplas en una colección de tuplas es igual al número total (k) de procesos en ejecución;

y se inicia la etapa b) con respecto a la ronda de generador completada.

10. Método según la reivindicación 9, en el que se define un número máximo de colecciones de tuplas y se suprime una colección de tuplas si se supera el número máximo de colecciones.

11. Producto de programa informático que comprende instrucciones que, cuando el programa es ejecutado por dispositivos de procesado de datos, tales como ordenadores, dispuestos en una red distribuida, consiguen que los dispositivos de procesado de datos lleven a cabo el método según una cualquiera de las reivindicaciones 1 a 10.

12. Sistema de procesado de datos que comprende una pluralidad de dispositivos de procesado de datos que comprenden unos medios para llevar a cabo el método según una cualquiera de las reivindicaciones 1 a 10.

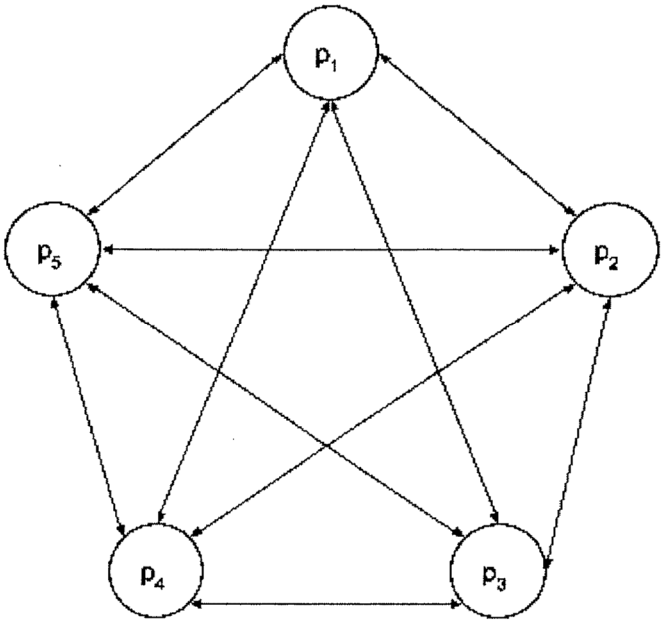
p1
p2
p3
p4
p5

Fig. 1

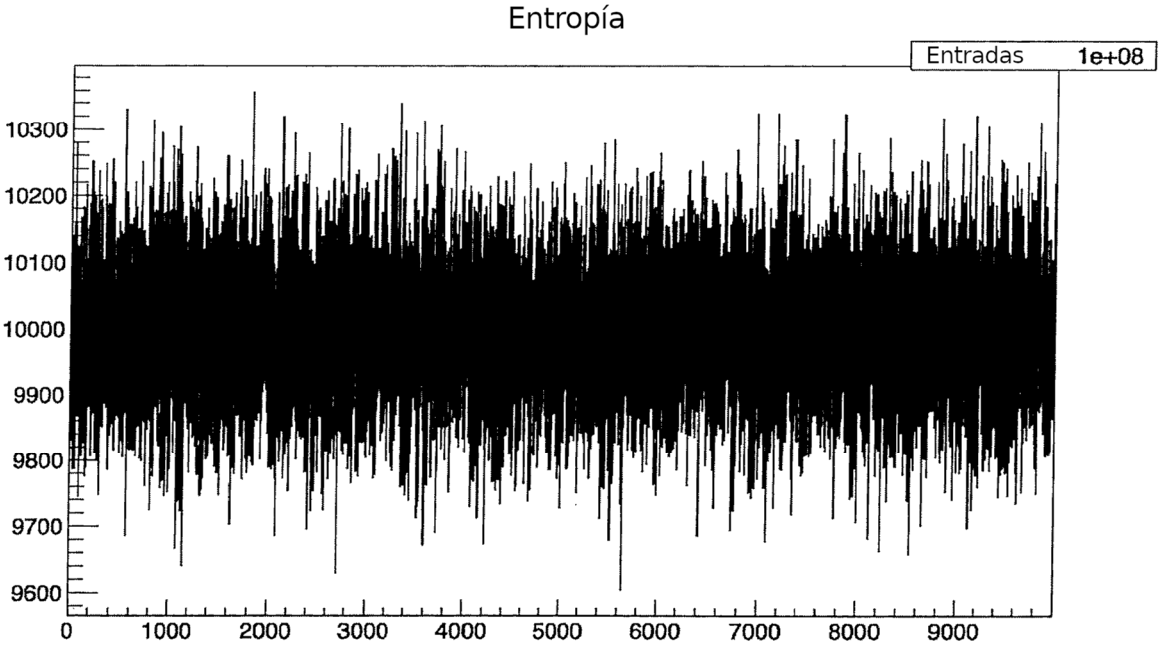
Entropía
Entradas 1e+08
10300
10200
10100
10000
9900
9800
9700
9600
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

Fig. 2

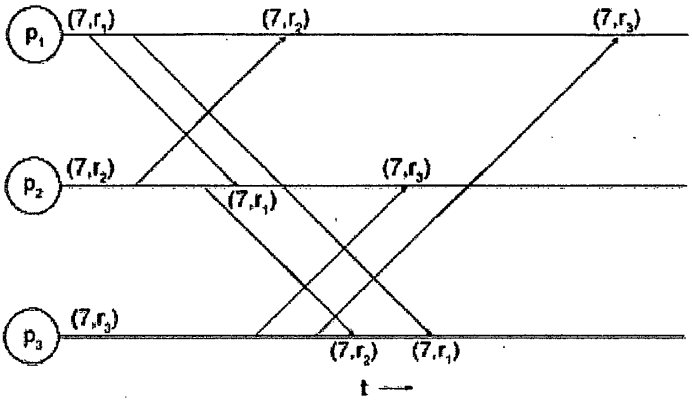

Fig. 3

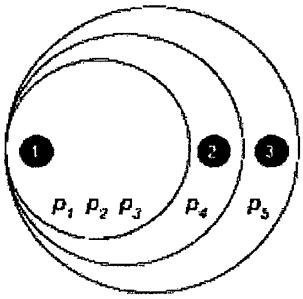

Fig. 4

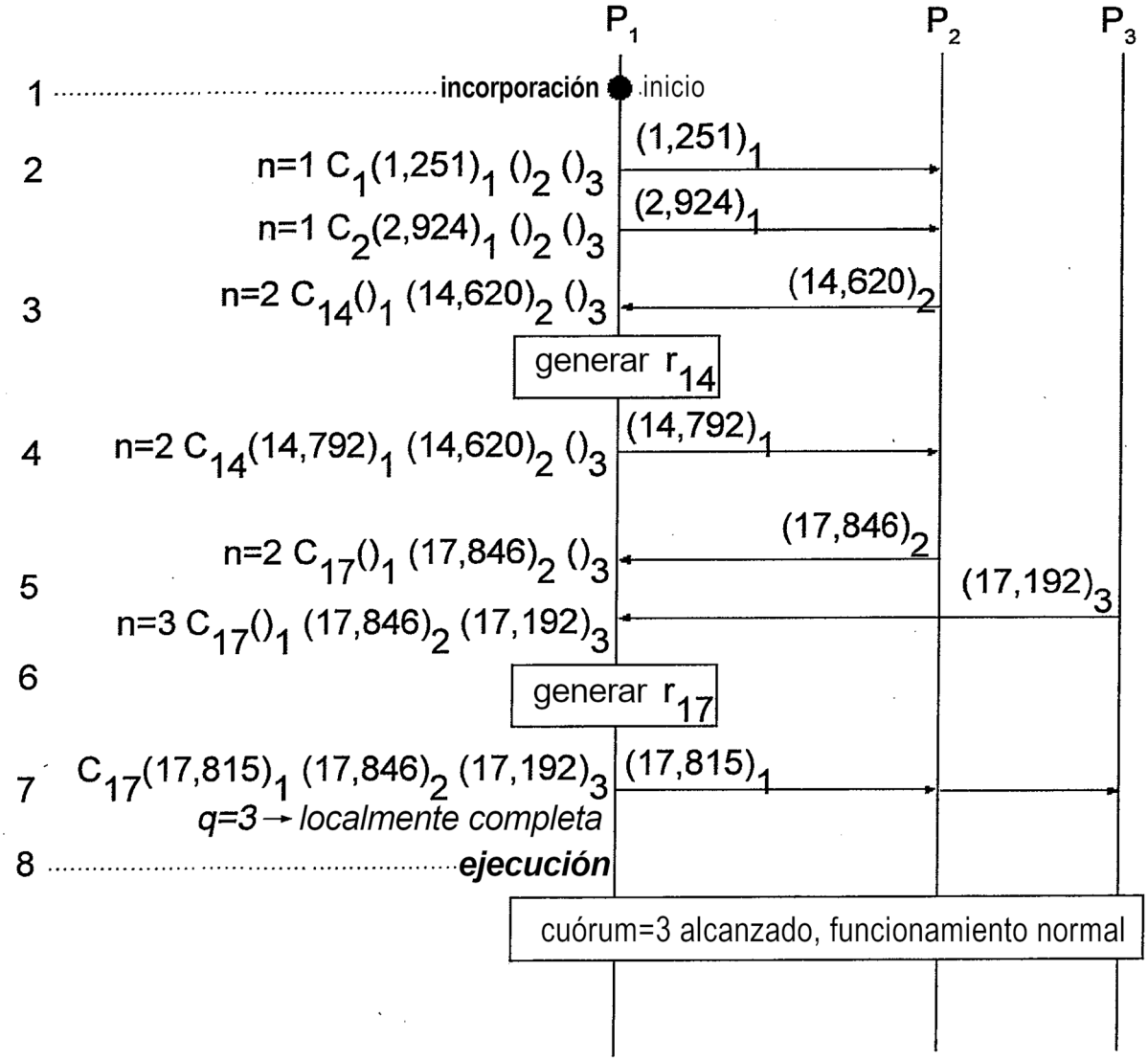
P1
P2
P3
1
incorporación
inicio
2
n=1 C1(1,251)1 ()2 ()3
(1,251)1
n=1 C2(2,924)1 ()2 ()3
(2,924)1
3
n=2 C14()1 (14,620)2 ()3
(14,620)2
generar r14
4
n=2 C14(14,792)1 (14,620)2 ()3
(14,792)1
n=2 C17()1 (17,846)2 ()3
(17,846)2
5
n=3 C17()1 (17,846)2 (17,192)3
(17,192)3
6
generar r17
7
C17(17,815)1 (17,846)2 (17,192)3
(17,815)1
q=3 → localmente completa
8
ejecución
cuórum=3 alcanzado, funcionamiento normal

Fig. 5

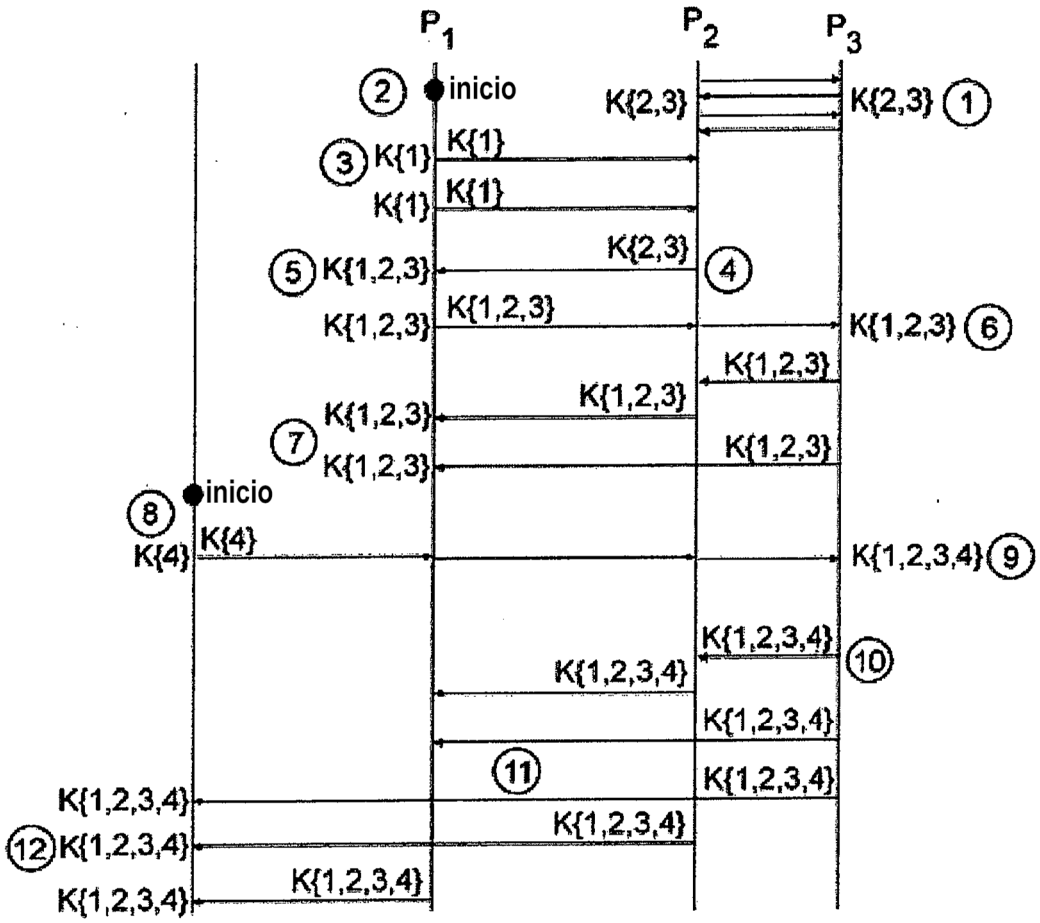
P1
P2
P3
2
inicio
K{2,3}
K{2,3}
1
3
K{1}
K{1}
K{1}
K{1}
K{2,3}
5
K{1,2,3}
4
K{1,2,3}
K{1,2,3}
K{1,2,3}
6
K{1,2,3}
K{1,2,3}
K{1,2,3}
7
K{1,2,3}
K{1,2,3}
inicio
8
K{4}
K{4}
K{1,2,3,4}
9
K{1,2,3,4}
10
K{1,2,3,4}
K{1,2,3,4}
11
K{1,2,3,4}
K{1,2,3,4}
K{1,2,3,4}
12
K{1,2,3,4}
K{1,2,3,4}
K{1,2,3,4}

Fig. 6

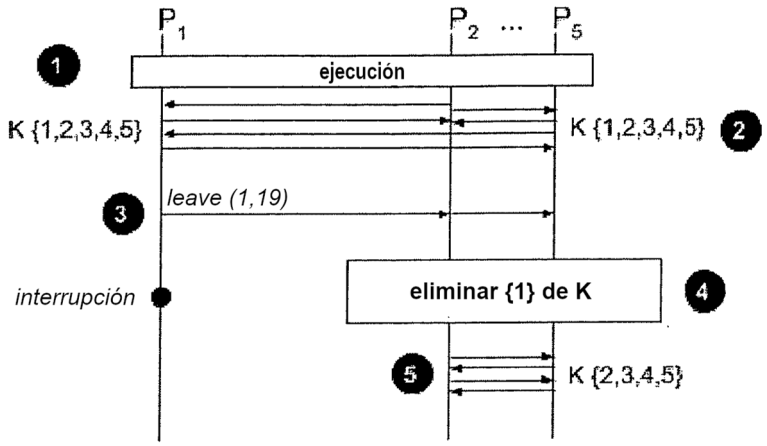
P1
P2 ... P5
1
ejecución
K {1,2,3,4,5}
K {1,2,3,4,5}
2
3
leave (1,19)
interrupción
eliminar {1} de K
4
5
K {2,3,4,5}

Fig. 7

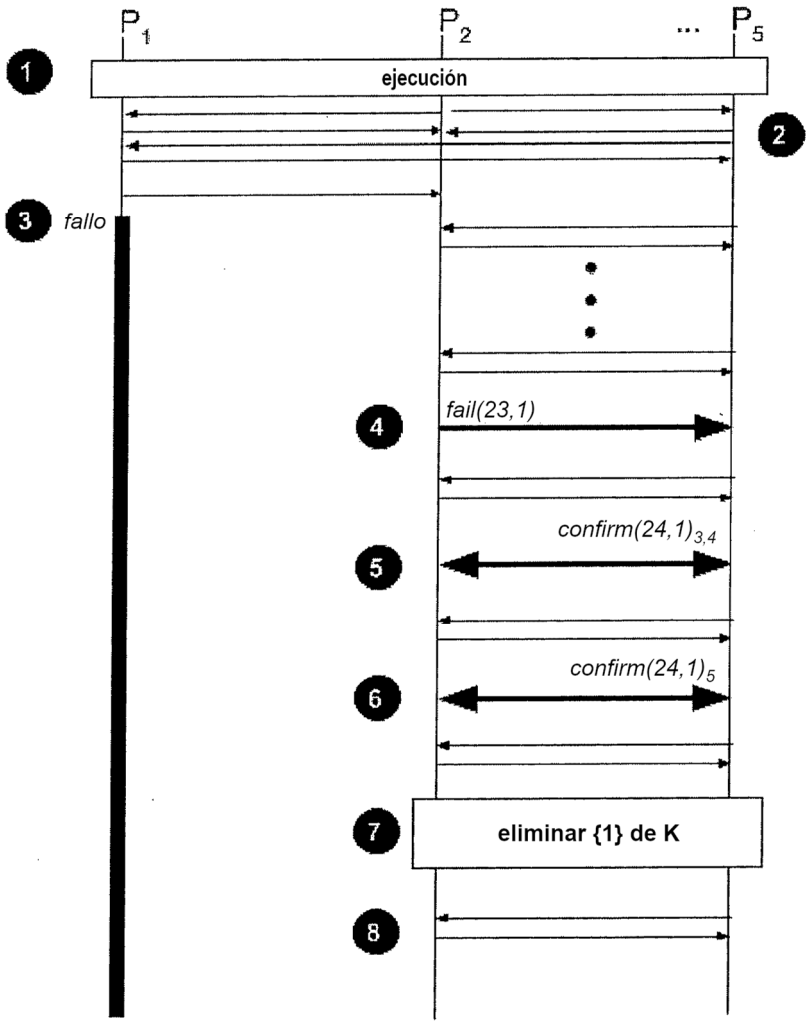
P1
P2
... P5
1
ejecución
2
3
fallo
4
fail(23,1)
5
confirm(24,1)3,4
6
confirm(24,1)5
7
eliminar {1} de K
8

Fig. 8

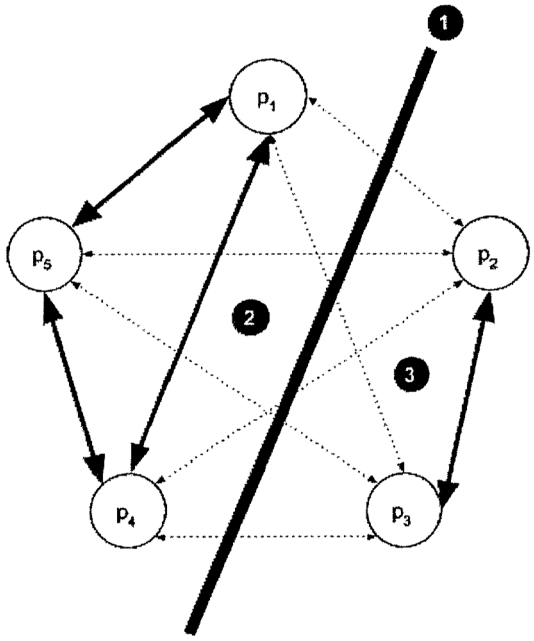
1
p1
p5
p2
2
3
p4
p3

Fig. 9

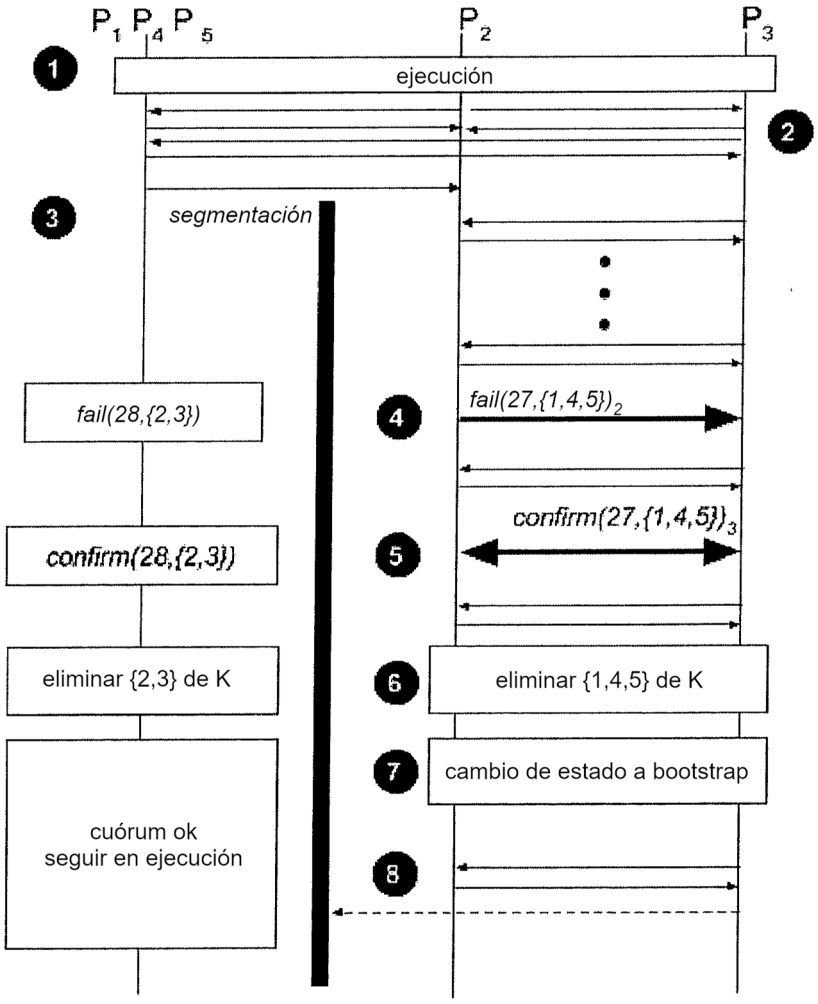
P1 P4 P5
P2
P3
1
ejecución
2
3
segmentación
fail(28,{2,3})
4
fail(27,{1,4,5})2
confirm(27,{1,4,5})3
confirm(28,{2,3})
5
eliminar {2,3} de K
6
eliminar {1,4,5} de K
7
cambio de estado a bootstrap
cuórum ok
seguir en ejecución
8

Fig. 10

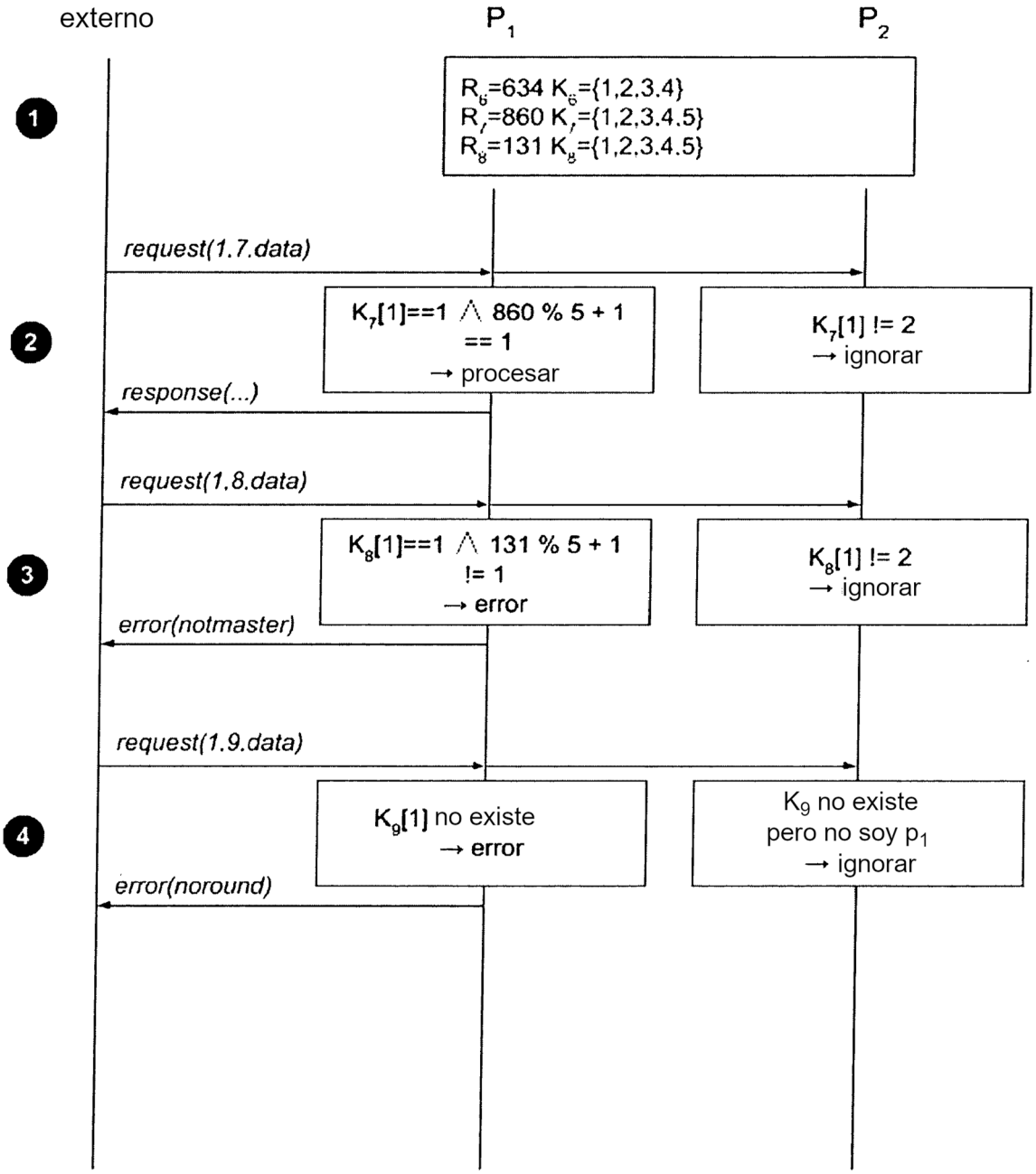
externo
P1
P2
1
R6=634 K6={1,2,3.4}
R7=860 K7={1,2,3.4.5}
R8=131 K8={1,2,3.4.5}
request(1.7.data)
2
K7[1]==1 ∧ 860 % 5 + 1 == 1 → procesar
K7[1] != 2 → ignorar
response(...)
request(1.8.data)
3
K8[1]==1 ∧ 131 % 5 + 1 != 1 → error
K8[1] != 2 → ignorar
error(notmaster)
request(1.9.data)
4
K9[1] no existe → error
K9 no existe pero no soy p1 → ignorar
error(noround)

Fig. 11

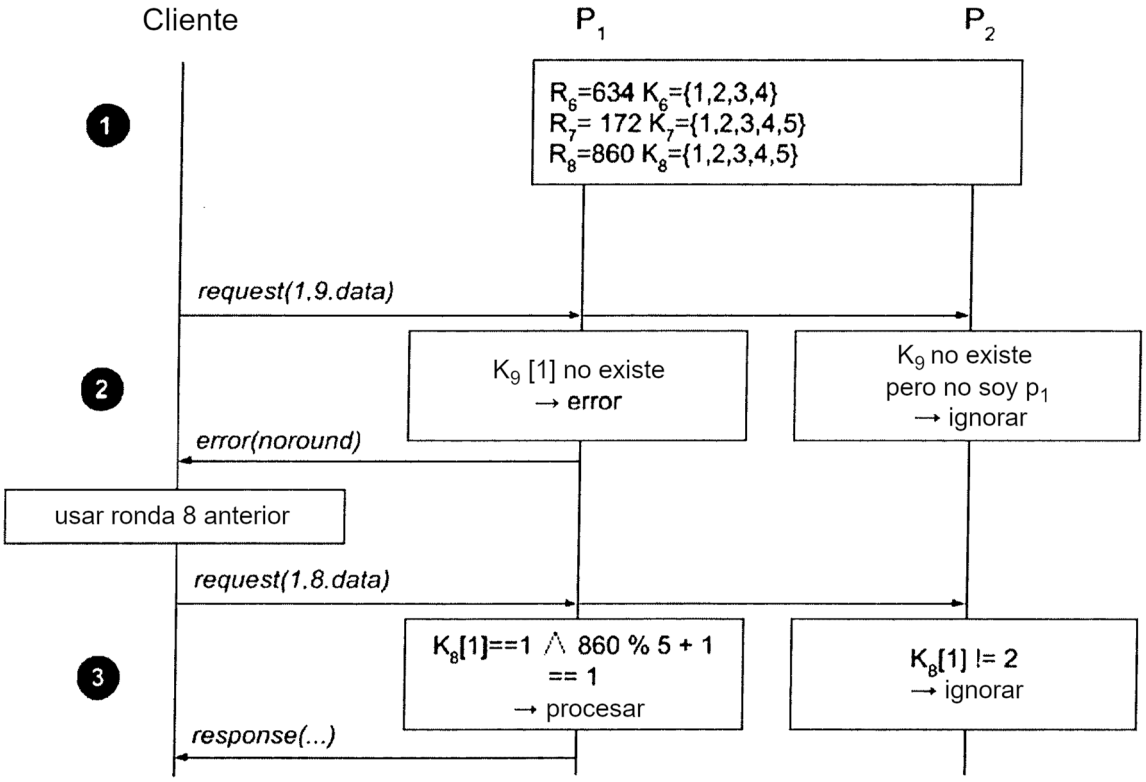
Cliente
P1
P2
1
R6=634 K6={1,2,3,4}
R7= 172 K7={1,2,3,4,5}
R8=860 K8={1,2,3,4,5}
request(1,9.data)
2
K9 [1] no existe
→ error
K9 no existe
pero no soy p1
→ ignorar
error(noround)
usar ronda 8 anterior
request(1.8.data)
3
K8[1]==1 ∧ 860 % 5 + 1
== 1
→ procesar
K8[1] != 2
→ ignorar
response(...)

Fig. 12