Problema del ataque coordinado

Varios generales están planeando un ataque coordinado desde diferentes direcciones, contra un objetivo común. Saben que la única forma de que el ataque tenga éxito es que ataquen todos los generales; si sólo atacan algunos de ellos, sus ejércitos serán destruidos. Cada general tiene una opinión inicial sobre si su ejército está listo para atacar.

Los generales están situados en lugares diferentes. Los generales cercanos pueden comunicarse, pero sólo a través de mensajeros que viajan a pie. Sin embargo, los mensajeros pueden perderse o ser capturados, por lo que sus mensajes pueden perderse. Utilizando sólo este medio de comunicación poco fiable, los generales deben conseguir ponerse de acuerdo sobre si atacar o no. Además, deben atacar si es posible.

Suponemos que la 'gráfica de comunicación' de los generales es no dirigida y conexa, y que todos los generales conocen la gráfica. También suponemos que existe un límite superior conocido del tiempo que tarda un mensajero en entregar un mensaje con éxito.

Si todos los mensajeros son confiables, entonces todos los generales pueden enviar mensajeros a todos los demás generales (posiblemente en varios saltos), diciendo si están o no dispuestos a atacar. Tras un número de 'rondas' igual al diámetro de la 'gráfica de comunicación', todos los generales dispondrán de toda esta información. Entonces, todos podrán aplicar una regla comúnmente acordada para tomar la misma decisión sobre el ataque: por ejemplo, podrán decidir atacar exactamente si todos los generales quieren hacerlo.

En un modelo en el que los mensajeros pueden perderse, este sencillo algoritmo no funciona. Resulta que no se trata sólo de un problema con este algoritmo: realmente no existe ningún algoritmo que resuelva siempre correctamente este problema.

Problema del general

Una versión alternativa del problema de consenso requiere que el valor de entrada de un proceso distinguido (llamado el general) se distribuya a todos los demás procesos (llamados los tenientes); este problema se conoce como consenso de fuente única. Las condiciones que deben cumplirse son:

- Terminación: Cada teniente no fallido debe eventualmente decidir.
- Acuerdo: Todos los tenientes no fallidos deben tener la misma decisión.
- Validez: Si el general es no fallido, entonces el valor de decisión común es la entrada del general.

La diferencia está en la condición de validez: si el general es defectuoso, el proceso no defectuoso no tiene por qué decidir sobre la entrada del general, pero deben estar de acuerdo entre sí.

DEFINICIÓN 1: El problema del ataque coordinado

Consideramos n procesos con su respectivo índice de $1, \ldots, n$, dispuestos en una gráfica no dirigida arbitraria, en el que cada proceso conoce toda la gráfica, incluidos los índices de los procesos. Cada proceso comienza con una entrada en $\{0, 1\}$ en un componente de estado designado. Usamos 1 para denotar 'atacar', o comprometerse, y 0 para denotar 'no atacar', o abortar. Utilizamos el mismo modelo síncrono con el que hemos estado trabajando hasta ahora, excepto que ahora permitimos que se pierda cualquier número de mensajes durante el curso de una ejecución. El objetivo es que todos los procesos acaben produciendo decisiones en $\{0, 1\}$, poniendo los componentes especiales del estado de decisión a 0 o 1. El objetivo es que todos los procesos acaben produciendo decisiones en $\{0, 1\}$. Hay tres condiciones impuestas a las decisiones tomadas por los procesos:

- Acuerdo: No hay dos procesos que decidan valores diferentes.
- Validez:
  1. Si todos los procesos empiezan con 0, entonces 0 es el único valor de decisión posible.
  2. Si todos los procesos comienzan con 1 y se entregan todos los mensajes, entonces 1 es el único valor de decisión posible.
- Terminación: Todos los procesos eventualmente deciden.

DEFINICIÓN 2: Consenso

Consideremos un sistema en el cual cada proceso p_i tiene un componente especial x_i, que es la entrada, y y_i la salida, también conocida como la decisión. Inicialmente, x_i contiene un valor de algún conjunto bien ordenado de posibles entradas y y_i está indefinido. Cualquier asignación a y_i es irreversible. Una solución al problema del consenso debe garantizar lo siguiente:

- Terminación: En cada ejecución admisible, se asigna finalmente un valor a y_i, para cada proceso no fallido p_i.
- Acuerdo: En cada ejecución, si y_i y y_j están asignados, entonces $y_i = y_j$, para todos los procesos no fallidos p_i y p_j. Es decir, los procesos no fallidos no deciden sobre valores contradictorios.
- Validez: En cada ejecución, si, para algún valor v, $x_i = v$ para todos los procesos p_i, y si y_i se asigna para algún proceso no fallido p_i entonces $y_i = v$. Es decir, si todos los procesos tienen la misma entrada, entonces cualquier valor que se decida debe ser esa entrada común.

OBSERVACIÓN.

- La condición de validez infiere que si todos los procesos tienen la misma entrada x, entonces los procesos tienen que decidir sobre x. Tenga en cuenta que el consenso no es democrático, puede ocurrir que los procesos decidan sobre un valor de entrada promovido por una pequeña minoría.
- Que el consenso sea posible depende de muchos parámetros del sistema distribuido, en par-

ticular si el sistema es síncrono o asíncrono, o qué significa 'fallido'.

- El consenso es una primitiva poderosa. Con un consenso establecido se puede calcular casi todo en un sistema distribuido, por ejemplo, un líder.
- Dado un sistema distribuido asíncrono de paso de mensajes con $n \geq 2$ procesos. Todos los procesos pueden comunicarse directamente con todos los demás procesos, simplemente enviando un mensaje. En otras palabras, la gráfica de comunicación es la gráfica completa. ¿Puede resolverse el problema del consenso? Sí.

Algorithm 1 Algoritmo de consenso trivial

1: Cada nodo tiene una entrada
2: Tenemos un líder, por ejemplo, el nodo con el ID más alto
3: **if** proceso p es el lider **then**
4: El líder simplemente decidirá sobre su propia entrada
5: **else**
6: Envia un mensaje al líder solicitando su entrada
7: Esperar el mensaje de respuesta del líder, y decidir con la respuesta

OBSERVACIÓN. Sin embargo, el algoritmo no es en absoluto tolerante a fallas. Si el líder falla antes de poder responder a todas las peticiones, hay nodos que nunca terminarán y, por tanto, la condición de terminación no se cumpliría.

Algorithm 2 Algoritmo de consenso en presencia de fallas
Código para el proceso p_i, $0 \leq i \leq n-1$

1: Initially $V = \{x\}$

2: **round** k, $1 \leq k \leq f+1$**:**
3: send $\{v \in V : p_i$ has not already sent $v\}$ to all processors
4: receive S_j from p_j, $0 \leq j \leq n-1$, $j \neq i$
5: $V := V \cup \bigcup_{j=0}^{n-1} S_j$
6: **if** $k = f+1$ **then**
7: $y := min(V)$
8: **end**

OBSERVACIÓN. El problema del consenso requiere que un conjunto de procesos se pongan de acuerdo en un único valor, dado que cada proceso comienza con un valor inicial. El problema se complica por el hecho de que algunos procesos pueden fallar durante la ejecución del algoritmo. El objetivo del algoritmo es garantizar que, a pesar de la posibilidad de que fallen los procesos, todos los procesos que no fallen se pongan de acuerdo sobre el mismo valor.

TEOREMA 1
El algoritmo 1 resuelve el problema del consenso en presencia de f fallas en $f+1$ rondas.

Demostración. Demostraremos por inducción sobre el número de rondas.

Caso base: En la primera ronda, cada proceso envía su valor inicial a todos los demás procesos. Como hay a lo más f procesos fallidos, al menos $n-(f+1)$ procesos recibirán el mismo conjunto de valores, y la unión de estos conjuntos contendrá todos los valores iniciales de los procesos no fallidos. Así, todos los procesos sin fallas tendrán el mismo conjunto de valores al final de la primera ronda.

Hipótesis de inducción: Supongamos que, después de k rondas, todos los procesos no fallidos tienen el mismo conjunto de valores, V.

Paso inductivo: En la ronda $(k+1)$, cada proceso envía los valores de V que aún no ha enviado a todos los demás procesos. Cada proceso no fallido recibirá al menos $n-(f+1)$ de estos conjuntos, ya que hay a lo más f procesos fallidos. Por tanto, la unión de estos conjuntos contendrá todos los valores actuales de los procesos no fallidos. Así, tras la ronda $(k+1)$, todos los procesos no fallidos tendrán el mismo conjunto de valores.

Por inducción, después de $f+1$ rondas, todos los procesos no fallidos tendrán el mismo conjunto de valores. Si hay al menos $f+1$ procesos no fallidos, entonces ejecutarán la sentencia condicional en la última ronda y acordarán el valor mínimo en V. Si hay menos de $f+1$ procesos no fallidos, entonces todos tendrán el mismo valor en la última ronda, lo cual está garantizado por el algoritmo.

Por lo tanto, el algoritmo resuelve el problema de consenso en presencia de f fallas en $f+1$ rondas.

□

TEOREMA 2
Sea G la gráfica formada por los nodos n_1 y n_2 conectados por una única arista. Entonces no existe ningún algoritmo que resuelva el problema del ataque coordinado en G.

TEOREMA 3
Cualquier algoritmo de consenso para n procesos que sea resistente a f fallas requiere de al menos $f+1$ rondas en alguna ejecución admisible, para todo $n \geq f+2$.

TEOREMA 4
Cualquier algoritmo de consenso para n procesos que sea resistente a f fallos requiere al menos $f+1$ rondas en alguna ejecución admisible, para todo $n \geq f+2$.

DEFINICIÓN 3: Reliable Broadcast (Difusión confiable)
Consideremos un sistema distribuido asíncrono con n procesos que pueden fallar. Dos nodos cualesquiera pueden intercambiar mensajes, es decir, la gráfica de comunicación está completa. Queremos que el proceso p envíe una difusión confiable a los $n-1$ procesos restantes. Confiable significa que, o bien nadie recibe el mensaje, o bien todo el mundo lo recibe.

OBSERVACIÓN. El principal problema es que el emisor puede bloquearse al enviar el mensaje a los $n-1$ procesos restantes, de modo que algunos de ellos reciban el mensaje y otros no. Necesitamos una técnica que se ocupe de este caso

Algorithm 3 Algoritmo de difusión confiable (Reliable Broadcast)

if Proceso p es el origen del mensaje m **then**
    Envía el mensaje m a cada uno de los $n-1$ procesos restantes
    Al recibir m de cualquier otro proceso: la difusión es exitosa
else
    Al recibir el mensaje m por primera vez:
    Envía el mensaje m a cada uno de los $n-1$ procesos restantes

TEOREMA 5
El algoritmo 3 resuelve la difusión confiable.

Demostración. En primer lugar, debemos señalar que no nos preocupan los procesos que fallan durante la ejecución: si reciben o no el mensaje es irrelevante, ya que fallan de todos modos. Si un único proceso no defectuoso p recibió el mensaje (no importa cómo, puede ser que lo recibiera a través de una ruta de nodos accidentados) todos los nodos no fallidos recibirán el mensaje a través de p. Si ningún nodo no fallido recibe el mensaje, se resuelve la difusión confiable.

□

DEFINICIÓN 4: Univalente, Bivalente
Un sistema distribuido se denomina x-valente si el resultado de un cálculo será x. Un sistema x-valente también se denomina univalente. Si, dependiendo de la ejecución, todavía es factible más de un resultado posible, el sistema se denomina multivalente. Si aún son posibles exactamente dos resultados, el sistema se denomina bivalente.

Modelo Bizantino

Varias divisiones del ejército bizantino acampan frente a una ciudad enemiga. Cada división está comandada por un general. Los generales sólo pueden comunicarse entre sí mediante mensajeros fiables. Los generales deben decidir un plan de acción común, es decir, deben decidir si atacan la ciudad o no (cf. acuerdo), y si los generales son unánimes en su opinión inicial, entonces esa opinión debe ser la decisión (cf. validez). La novedad es que algunos de los generales pueden ser traidores (por eso están en el ejército bizantino) e intentar impedir que los generales leales se pongan de acuerdo. Para ello, los traidores envían mensajes contradictorios a diferentes generales, informan falsamente de lo que han oído a otros generales e incluso conspiran y forman una coalición.

Modelo Formal

En una ejecución de un sistema bizantino resistente a fallas, existe un subconjunto de a lo más f procesos, los procesos fallidos. Cuando un proceso fallido realiza un cálculo local, el nuevo estado del proceso y el contenido de los mensajes enviados no tienen ninguna restricción. Como en el caso confiable, cada proceso realiza un cálculo local en cada ronda y cada mensaje enviado se entrega en esa ronda.

Así, un proceso fallido puede comportarse de forma arbitraria e incluso maliciosa, por ejemplo, puede enviar mensajes diferentes a procesos diferentes (o no enviar mensajes en absoluto) cuando se supone que debe enviar el mismo mensaje. Los procesos fallidos pueden parecer que se coordinan entre sí. En algunas situaciones, el receptor de un mensaje procedente de un proceso defectuoso puede detectar que el emisor es defectuoso, por ejemplo, si el mensaje tiene un formato incorrecto. Las dificultades surgen cuando el mensaje recibido es plausible para el receptor, pero no

es correcto. Un proceso fallido también puede imitar el comportamiento de un proceso bloqueado y no enviar ningún mensaje a partir de cierto momento.

DEFINICIÓN 5: El problema del consenso en presencia de fallos bizantinos

La definición del problema de consenso en presencia de fallos bizantinos es la misma que para los fallos de colisión (crash failures) y se repite aquí. Cada proceso p_i tiene componentes de estado de entrada y salida, x_i y y_i; inicialmente, x_i, tiene un valor de algún conjunto bien ordenado y y_i está indefinido. Cualquier asignación a y_i es irreversible. Una solución al problema de consenso debe garantizar lo siguiente:

- Terminación: En cada ejecución admisible, se asigna finalmente un valor a y_i para cada uno de los procesos no defectuosos p_i.
- Acuerdo: En cada ejecución, si y_i y y_j son asignados, entonces $y_i = y_j$, para todos los procesos no defectuosos p_i y p_j. Es decir, los procesos no defectuosos no deciden sobre valores en conflicto.
- Validez: En cada ejecución, si, para algún valor v, $x_i = v$ para todos los procesos p_i, y si y_i se asigna para algún proceso no defectuoso p_i, entonces $y_i = v$. Es decir, si todos los procesos tienen la misma entrada, entonces cualquier valor decidido por un proceso no defectuoso debe ser esa entrada común.

TEOREMA 6

En un sistema con tres procesos y un proceso bizantino, no existe un algoritmo que resuelva el problema del consenso.

Demostración. Supongamos, por contradicción, que existe un algoritmo para alcanzar el consenso en un sistema con tres procesos, p_0, p_1 y p_2, conectados por una gráfica de comunicación completa. Sea A el algoritmo local (máquina de estados) para p_0, B el algoritmo local para p_1, y C el algoritmo local para p_2.

Consideremos un sistema síncrono de anillo con seis procesos en el que p_0 y p_3 tienen A para su algoritmo local, p_1 y p_4 tienen B para su algoritmo local, y p_2 y p_5 tienen C para su algoritmo local, como se representa en la siguiente figura. No podemos suponer que un sistema así resuelva consensos, ya que la combinación de A, B y C sólo tiene que funcionar correctamente en un triángulo. Sin embargo, este sistema tiene un comportamiento particular y bien definido, cuando cada proceso comienza con un valor de entrada y no hay procesos defectuosos.

La ejecución particular del anillo de interés es cuando los valores de entrada son 1 para p_0, p_1 y p_2 y 0 para p_3, p_4 y p_5. Llame a esta ejecución β. Esta ejecución se utilizará para especificar el comportamiento de los procesos fallidos en algunos triángulos.

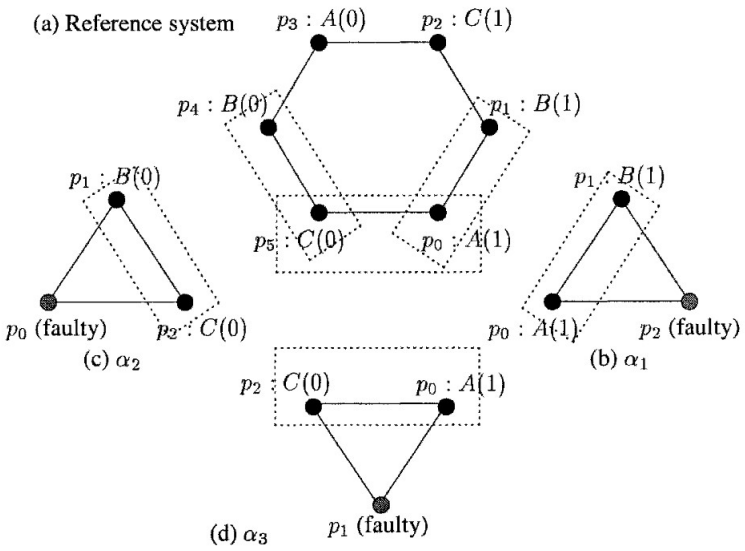
Consideremos una ejecución α_1 del algoritmo en un triángulo en el que todos los procesos comienzan con la entrada 1 y el proceso p_2 es defectuoso. Además, supongamos que el proceso p_2 está enviando a p_0 los mensajes enviados en β por p_5 a p_0 y a p_1 los mensajes enviados en β por p_2 a p_1. Por la condición de validez, tanto p_0 como p_1 deben decidir 1 en α_1.

Consideremos ahora una ejecución α_2 del algoritmo en un triángulo en el que todos los procesos comienzan con la entrada 0, y el proceso p_0 está defectuoso. Además, supongamos que el proceso p_0 está enviando a p_1 los mensajes enviados en β por p_3 a p_4 y a p_2 los mensajes enviados en β por p_0 a p_5. Por la condición de validez, tanto p_1 como p_2 deben decidir 0 en α_2.

Por último, considere una ejecución α_3 del algoritmo en un triángulo donde el proceso p_0 comienza con la entrada 1, el proceso p_2 comienza con la entrada 0, y el proceso p_1 es defectuoso. Además, supongamos que el proceso p_1 está enviando a p_2 los mensajes enviados en β por p_4 a p_5 y a p_0 los mensajes enviados en β por p_1 a p_0.

Ahora argumentamos que $\alpha_1 \overset{p_0}{\sim} \alpha_3$. Dado que los mensajes enviados por el proceso fallido p_2 se definen con referencia a β, una simple inducción sobre el número de rondas verifica que p_0 tiene la misma vista en α_1 así como en β y que p_1 tiene la misma vista en α_1 que en β. De forma similar, la inducción sobre el número de rondas verifica que p_0 tiene la misma vista en β que en α_3 y p_5 tiene la misma vista en β que p_2 en α_3. Por lo tanto $\alpha_1 \overset{p_0}{\sim} \alpha_3$, y en consecuencia, p_0 decide 1 en α_3.

Pero como $\alpha_2 \overset{p_2}{\sim} \alpha_3$, p_2 decide 0 en α_3, violando la condición de acuerdo, una contradicción.



□

OBSERVACIÓN. Algunos de los resultados clave en computación distribuida prometen proporcionar un sistema distribuido que pueda tolerar fallos arbitrarios. Estos fallos suelen denominarse bizantinos. Las máquinas (nodos) de un sistema distribuido comprueban periódicamente si se encuentran en el mismo estado. Cuando los nodos no están de acuerdo, ejecutan un protocolo de acuerdo bizantino para eliminar las meteduras de pata de algunos nodos y reforzar así el acuerdo. Si los nodos coinciden continuamente en su estado, el sistema distribuido en su conjunto es correcto.

Para que un sistema distribuido sea aplicable en la práctica, un protocolo de acuerdo bizantino debe ser rápido y garantizar que todos los nodos coinciden en un estado común. Estos criterios se conocen como tiempo de terminación y propiedad de acuerdo, respectivamente. Sin embargo, ambas propiedades no son suficientes. Por ejemplo, un protocolo puede simplemente acordar borrar toda la información del sistema, cumpliendo tanto la terminación como el acuerdo, pero potencialmente destruyendo también datos valiosos. Para evitar estas 'soluciones' absurdas, necesitamos una tercera propiedad, conocida como propiedad de validez. Informalmente, la propiedad de validez debe garantizar que la decisión 'tiene sentido'. En concreto, si todos los nodos de un sistema distribuido proponen el mismo estado, deben conformarse con ese estado.

Observación. Consideramos un adversario computacionalmente ilimitado e incluso suponemos que el adversario bizantino puede ver la información que se transmite por todos los canales de comunicación. Se ha demostrado que es imposible resolver el acuerdo bizantino en un sistema de este tipo si un tercio de los participantes son bizantinos. Por tanto, supondremos que menos de un tercio de las partes son bizantinos y nos centraremos en derivar condiciones de validez significativas. En concreto, queremos ver hasta qué punto pueden ser fiables los sistemas distribuidos si tienen que ponerse de acuerdo sobre valores reales, como el valor verdadero de un sensor en un coche. También vamos un paso más allá y consideramos un entorno multidimensional en el que la decisión debe basarse en clasificaciones. Estas aplicaciones pueden encontrarse, por ejemplo, en sistemas de aprendizaje automático a gran escala, en los que la decisión final debe tomarse a partir de un conjunto de clasificaciones. Aunque los nodos bizantinos pueden impedir que los sistemas se pongan de acuerdo sobre el valor verdadero, sigue siendo posible acordar un valor o una clasificación bastante buenos.

Observación. Además de las condiciones de validez, es necesario desarrollar modelos transparentes para los algoritmos de acuerdo distribuidos. Los algoritmos de concordancia suelen desplegarse como subrutinas y permiten resolver de forma fiable tareas más complicadas. Un modelo complicado para el acuerdo podría conducir a sistemas que hacen falsas suposiciones sobre sus subrutinas. Con este espíritu, desarrollamos un modelo novedoso para analizar los protocolos de blockchain que se basa en una memoria compartida en lugar de una comunicación peer-to-peer. Este modelo puede simularse en un entorno peer-to-peer y satisface las mismas propiedades. La ventaja de la memoria compartida es que se hace más transparente por qué no es posible un acuerdo bizantino asíncrono en la blockchain cuando el adversario es computacionalmente ilimitado. De este modo, la comunicación restringida nos permite derivar un análisis de corrección más sencillo de los protocolos de cadena de bloques y compararlos entre sí. Consideramos además el modelo de acuerdo bizantino asíncrono, en el que los mensajes entre los pares de un sistema pueden retrasarse arbitrariamente. Nuestro análisis muestra que los complicados algoritmos de acuerdo bizantino tienden a hacer casi imposible el análisis de todas las estrategias bizantinas concebibles. Por lo tanto, investigamos métodos de aprendizaje profundo por refuerzo para comprender el comportamiento bizantino y podríamos permitir el desarrollo de algoritmos de acuerdo bizantino sencillos y eficientes en el futuro.

Definición 6: Comunicación síncrona

La comunicación se divide en rondas discretas. En una ronda, cada nodo puede enviar un valor, recibir valores de todos los demás nodos correctos y realizar un cálculo local. Todos los mensajes enviados por un nodo correcto en una ronda serán recibidos por sus destinatarios en la misma ronda.

Definición 7: Comunicación asíncrona

No existe un límite superior para el retardo entre el envío y la recepción de un mensaje. Todo mensaje enviado por un nodo correcto acabará llegando a su destino.

Definición 8: Nodos síncronos

Existe una constante $\Delta > 0$ tal que cualquier intervalo entre dos operaciones ejecutadas localmente por un único nodo está acotado desde arriba por Δ. Todos los nodos conocen el límite superior Δ.

DEFINICIÓN 9: Nodos asíncronos

El tiempo entre dos operaciones de un nodo no está acotado. Sin embargo, en una ejecución infinita del protocolo, cada nodo correcto debe realizar infinitas operaciones. De lo contrario, el nodo se denomina defectuoso.

OBSERVACIÓN. Por lo general, el protocolo de acuerdo bizantino debe cumplir las siguientes condiciones estándar:

- Acuerdo: Todos los nodos correctos acuerdan el mismo valor a la terminación.
- Terminación: Todos los nodos correctos deben terminar después de ejecutar un número finito de operaciones.
- Validez de todos iguales: Si todos los nodos correctos tienen el mismo valor de entrada b, deben estar de acuerdo en b al final del protocolo.
- Validez de entrada correcta: Los nodos coinciden en el valor que ha propuesto al menos uno de los nodos correctos.

La condición de validez 'todos iguales' es la condición de validez más débil posible, ya que sólo impide que un sistema se ponga de acuerdo sobre un valor predefinido. Por lo tanto, en esta disertación, a veces nos referiremos a la condición de validez 'todo igual' simplemente como la condición de validez. Obsérvese además que la validez de entrada correcta es equivalente a la validez de todos iguales si los valores de entrada de los nodos son binarios. Sin embargo, en el escenario multivaluado, en el que todos los nodos tienen valores de entrada diferentes, no se puede satisfacer la validez de entrada correcta. Esto se debe a que cualquier nodo correcto no es diferenciable de un nodo bizantino que sigue el protocolo utilizando su propio valor de entrada. Por tanto, introducimos una variante relajada de la validez de la entrada correcta:

- Validez de cualquier entrada: Los nodos están de acuerdo en el valor que al menos uno de los nodos ha propuesto. No es necesario que este valor sea propuesto por un nodo correcto.

OBSERVACIÓN. Es imposible lograr el consenso de forma determinista en un modelo de comunicación asíncrona. Por lo tanto, las propiedades de consenso presentadas pueden debilitarse de forma que cada una de ellas sólo se satisfaga con una probabilidad alta. Llamamos a las propiedades debilitadas acuerdo débil, terminación débil y validez débil (para validez débil de todos iguales) respectivamente.

OBSERVACIÓN. Tres nodos no pueden establecer un acuerdo en presencia de un nodo bizantino, incluso si el sistema de comunicación es síncrono. Dados n nodos, se demostró para el modelo síncrono que se requieren al menos $t + 1$ rondas para establecer el acuerdo, donde $t < n/3$ es el número de nodos bizantinos en el sistema. Los algoritmos Phase Queen y Phase King que cumplen este límite inferior para valores de entrada binarios.

DEFINICIÓN 10: Difusión Confiable en presencia de procesos bizantinos

En la primera ronda de este protocolo, un nodo -el emisor- emite un valor binario. Todos los nodos que han recibido un valor en la primera ronda emiten un mensaje de eco para este valor en la segunda ronda. Si algún nodo no ha recibido un valor, pero sí suficientes ecos para este valor, emite un eco para el mensaje reenviado en la tercera ronda. Tenga en cuenta que los nodos podrían hacerse eco de dos valores diferentes de esta forma, ya que un remitente bizantino podría enviar valores diferentes a nodos diferentes. Por último, si un nodo recibe suficientes ecos para un valor en ambas rondas, aceptará este valor. Si hay $n/4 < t < n/3$ nodos bizantinos participando en el protocolo, los nodos correctos podrían aceptar más de un valor del remitente.Para $t < n/4$ nodos bizantinos, se aceptará como máximo un valor del remitente. La propiedad de aceptar como máximo un valor por remitente es deseable en el entorno bizantino, ya que no se desea aceptar demasiados valores bizantinos en total. Supondremos que la difusión fiable se aplica con $t < n/4$ nodos bizantinos.

Algorithm 2.2 Synchronous Reliable Broadcast (code for node u)

```
1:  Broadcast own input bit msg(u)
2:  for all received msg(v) do
3:      Broadcast echo(u, msg(v))
4:  end for
5:  for all echo(w, msg(v)) received from at least n − 2t nodes w do
6:      if not echoed bit msg(v) before then
7:          Broadcast echo(u, msg(v))
8:      end if
9:  end for
10: for all echo(w, msg(v)) received from at least n − t nodes w do
11:     Accept bit msg(v)
12: end for
```

OBSERVACIÓN. La difusión fiable garantiza las siguientes propiedades para cada valor de entrada de difusión:

- Si un nodo correcto no ha emitido un mensaje, éste no será aceptado.
- Todos los mensajes correctos serán aceptados en una ronda de difusión fiable.

En el modelo de comunicación síncrona, las líneas 5 a 12 pueden repetirse varias veces. En este caso, el protocolo también cumpliría la siguiente propiedad:

- Si un nodo correcto acepta un mensaje de forma fiable, todos los demás nodos correctos aceptarán el mismo mensaje al final de la siguiente ronda.

Las propiedades anteriores también pueden satisfacerse con protocolos ligeramente simplificados en el modelo síncrono. El motivo de presentar esta versión es que puede utilizarse directamente en el modelo de comunicación asíncrono. El único ajuste necesario es dejar que los nodos ejecuten una operación de eco al recibir $n - 2t$ y $n - t$ mensajes para cada valor respectivamente en lugar de depender de rondas síncronas. La repetición de las líneas 5 a 12 no es necesaria en el modelo asíncrono.

DEFINICIÓN 11: Phase King

$t+1$ nodos pueden ser elegidos como 'líderes' en el algoritmo, cada líder asociado con una fase del algoritmo. El algoritmo puede utilizar entradas multivaluadas y sólo se requiere que satisfaga la condición de validez todos iguales. Cada fase del algoritmo se divide en tres rondas de comunicación: en la primera ronda, los nodos comparten sus valores de entrada y comprueban si se cumple la validez 'todo igual'; en la segunda ronda, los nodos que satisfacen la validez 'todo igual' transmiten este conocimiento, y otros nodos ajustan sus valores si es necesario; en la última ronda sólo se permite comunicar al líder que se denomina 'rey' en el protocolo, que simplemente comparte su propio valor. La idea es que después de una ronda con un nodo correcto como rey, todos los nodos tendrán el mismo valor de salida. En el caso de un rey bizantino, se mantiene la validez de todos iguales, pero los nodos correctos podrían tener valores diferentes al final de la ronda si sus valores de entrada fueran distintos.

Algorithm 2.1 Phase King Protocol for $t < n/3$ (code for node i)

```
1:  for phase i = 1 to t + 1 do
        Communication Round:
2:      Broadcast input value v_i
3:      receive guesses v_j from all other nodes
4:      if some value v is received ≥ n − t times then
5:          Broadcast("propose v")
6:      end if
7:      if some "propose v" received > t times then
8:          Adopt input value v_i := v
9:      end if
        King Round (only the King node executes this round):
10:     kingValue = v_king
11:     Broadcast("suggest kingValue")
        Decision Round:
12:     if "propose v" received < n − t times in Line 5 then
13:         v_i = kingValue
14:     end if
15: end for
```

TEOREMA 7: Un algoritmo polinomial

El siguiente algoritmo utiliza mensajes de tamaño constante, tarda $2(f+1)$ rondas y asume que $n > 4f$. Demuestra que es posible resolver el problema del consenso con mensajes de tamaño constante, aunque con un aumento del número de rondas y una disminución de la resiliencia. Cada uno de los procesos mantiene una matriz local *pref* con n entradas.

Algorithm 16 A polynomial consensus algorithm in the presence of Byzantine failures: code for p_i, $0 \leq i \leq n-1$.

```
Initially pref[i] = x                          // initial preference for self is for own input
and pref[j] = v⊥, for any j ≠ i                // default for others

1:  round 2k − 1, 1 ≤ k ≤ f + 1:               // first round of phase k
2:      send ⟨pref[i]⟩ to all processors
3:      receive ⟨v_j⟩ from p_j and assign to pref[j], for all 0 ≤ j ≤ n − 1, j ≠ i
4:      let maj be the majority value of pref[0], ..., pref[n − 1] (v⊥ if none)
5:      let mult be the multiplicity of maj

6:  round 2k, 1 ≤ k ≤ f + 1:                   // second round of phase k
7:      if i = k then send ⟨maj⟩ to all processors   // king of this phase
8:      receive ⟨king-maj⟩ from p_k (v⊥ if none)
9:      if mult > n/2 + f
10:       then pref[i] := maj
11:       else pref[i] := king-maj
12:     if k = f + 1 then y := pref[i]         // decide
```

Demostración. El algoritmo contiene $f+1$ fases, cada una de las cuales dura dos rondas. Cada uno de los procesos tiene una decisión preferida (en resumen, preferencia) para cada fase, inicialmente su valor de entrada. En la primera ronda de cada fase, todos los procesos se envían sus preferencias. Sea v_i^k el valor mayoritario en el conjunto de valores recibidos por el proceso p_i al final de la primera ronda de la fase k. Si no hay mayoría, se utiliza un valor por defecto, v_t. En la segunda ronda de la fase, el proceso p_k, llamado el rey de la fase, envía su valor mayoritario v_k^k a todos los procesos. Si p_i recibe más de $\frac{n}{2} + f$ copias de v_i^k (en la primera ronda de la fase), entonces establece que su preferencia para la siguiente fase sea v_i^k; de lo contrario, establece que su preferencia sea la preferencia del rey de la fase, v_k^k, recibida en la segunda ronda de la fase. Después de $f+1$ fases, los procesos deciden su preferencia. □

LEMA 8. Si todos los procesos no defectuosos prefieren v al principio de la fase k, entonces todos prefieren v al final de la fase k, para toda k, $1 \leq k \leq f+1$.

Demostración. Puesto que todos los procesos no defectuosos prefieren v al principio de la fase k, cada proceso recibe al menos $n-f$ copias de v (incluida la suya propia) en la primera ronda de la fase k. Puesto que $n > 4f$, $n - f > \frac{n}{2} + f$, lo que implica que todos los procesos no defectuosos preferirán v al final de la fase k.

□

- M. Raynal, Fault-Tolerant Message-Passing Distributed Systems

- H. Attiya, Distributed Computing, Cap. 5