



SISTEMAS DISTRIBUTIVOS

PRINCIPIOS FUNDAMENTALES DEL ALGORITMO PAXOS

PRINCIPIOS FUNDAMENTALES DEL ALGORITMO PAXOS

El algoritmo Paxos, propuesto por el científico informático Leslie Lamport, es uno de los pilares teóricos y prácticos más importantes en el campo de los sistemas distribuidos. Diseñado para resolver el problema del consenso en entornos donde los procesos pueden fallar o quedar suspendidos, Paxos ofrece una solución robusta que garantiza que un grupo de nodos pueda acordar de forma segura un único valor, incluso en presencia de fallos parciales o de red (Takada, 2013).

A pesar de su fama por ser complejo en su formulación, los principios fundamentales de Paxos se basan en una lógica clara: permitir que los procesos distribuidos progresen de forma segura hacia un consenso, garantizando consistencia sin necesidad de confiar en un nodo específico.

Escenario general del algoritmo

Paxos se aplica en escenarios donde múltiples nodos deben ponerse de acuerdo sobre una acción o valor (por ejemplo, el contenido de un registro, el líder del sistema o una decisión de configuración), y en los cuales no se puede garantizar que todos estén en línea o que los mensajes lleguen en orden (Takada, 2013).

Para esto, Paxos define un conjunto de roles básicos que los procesos participantes pueden desempeñar:

- **Proposers (proponentes):** proponen valores a ser aceptados.
- **Acceptors (aceptantes):** votan por propuestas. Son el "corazón del consenso".
- **Learners (aprendices):** aprenden el valor aceptado y lo aplican en el sistema.

Un mismo nodo puede desempeñar más de un rol en una implementación práctica.

Principios fundamentales del algoritmo Paxos

1. Propuestas con identificadores únicos y ordenados.

Cada propuesta enviada por un proposer tiene un número único (normalmente un número natural creciente). Esto permite a los acceptors distinguir entre propuestas nuevas y antiguas y asegurar que el valor final aceptado sea el más reciente aprobado por mayoría (Takada, 2013).

📌 **Ejemplo.** Si un nodo propone el valor "X" con número B y luego otro propone "Y" con número B, los acceptors darán preferencia al número más alto.

2. Proceso en dos fases (Prepare y Accept).

Fase 1: Prepare (Preparar)

El proposer envía un mensaje a los acceptors con una propuesta preliminar:
"¿Aceptarán una propuesta con número N?"

Los acceptors responden si no han aceptado otra propuesta con un número mayor, y en su respuesta, indican el valor que hayan aceptado anteriormente (si existe).

Fase 2: Accept (Aceptar)

Si el proposer recibe una mayoría de respuestas positivas, entonces envía un nuevo mensaje con una propuesta definitiva (valor y número N).


Los acceptors aceptan esta propuesta si no han prometido aceptar una con número superior en el intervalo.

Estas dos fases aseguran que:

- No se pierde un valor ya aceptado.
- Solo se acepta un único valor por consenso.
- Se puede seguir intentando consenso sin romper la consistencia.

3. Mayoría de acceptors para decidir.

Para garantizar seguridad, Paxos requiere que al menos una mayoría (quorum) de acceptors coincida en una propuesta. Como todas las mayorías posibles se solapan, este mecanismo evita decisiones contradictorias, incluso si algunos nodos fallan (Muñoz Escoí, 2013).

 **Ejemplo.** En un sistema con 5 acceptors, es necesario que al menos 3 acepten un valor. Si otra propuesta se inicia, también necesitará al menos 3, lo cual asegura que al menos un nodo común compare el nuevo valor con el anterior.

4. Tolerancia a fallos.

Paxos está diseñado para funcionar correctamente incluso si algunos nodos fallan o se desconectan, siempre que haya una mayoría funcional. Este diseño permite que el sistema siga operando de forma coherente sin esperar a todos los participantes (Muñoz Escoí, 2013).

El algoritmo no depende de un nodo único, lo que evita puntos únicos de falla.

5. Seguridad garantizada, pero no liveness asegurado.

Paxos siempre garantiza la seguridad: nunca dos nodos aceptarán valores diferentes como resultado final del consenso. Sin embargo, no garantiza la progresión (liveness) si hay conflictos continuos entre proposers o condiciones extremas de red. En la práctica, se diseñan variantes como Multi-Paxos para mejorar su desempeño.


 **Ejemplo.** Supongamos que tres servidores (A, B, C) están tratando de acordar qué configuración activar. El servidor A propone la configuración “ModoSeguro” con número 1. Los tres acceptors responden positivamente y aceptan esa propuesta. Luego, el servidor B propone “ModoAltoRendimiento” con número 2. Debido a que ya se aceptó una propuesta con número 1, los acceptors comparan y priorizan la nueva, asegurando que todos terminen aceptando solo una opción final, sin duplicidades ni contradicciones.

Figura 1. Tolerancia a fallos

