ACH 2147 — Desenvolvimento de Sistemas de Informação Distribuídos

Aula 24: Tolerância a Falhas (parte 2)

Prof. Renan Alves

Escola de Artes, Ciências e Humanidades — EACH — USP

07/06/2024

Consenso realista: Paxos

Pressupostos (relativamente fracos e realistas)

- Um sistema parcialmente síncrono (pode ser até mesmo assíncrono).
- Comunicação entre processos pode ser não confiável: mensagens podem ser perdidas, duplicadas ou reordenadas.
- Mensagens corrompidas podem ser detectadas (e, portanto, ignoradas).
- Todas as operações são determinísticas: uma vez que uma a execução seja iniciada, sabe-se exatamente o que ela fará.
- Processos podem apresentar falhas de crash, mas não falhas arbitrárias.
- Processos não agem em conluio.

Entendendo Paxos

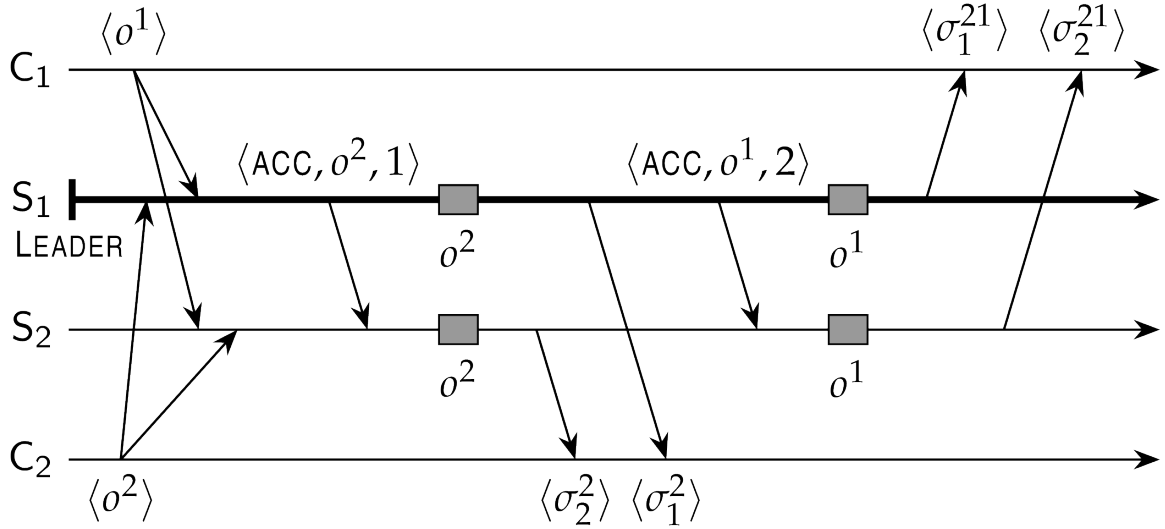
Vamos construir o Paxos do zero para entender de onde vêm muitos algoritmos de consenso.

Fundamentos do Paxos

Ponto de partida

- Assumimos uma configuração cliente-servidor, inicialmente com um servidor primário.
- Para tornar o servidor mais robusto, começamos adicionando um servidor de backup.
- Para garantir que todos os comandos sejam executados na mesma ordem em ambos os servidores, o servidor primário atribui números de sequência únicos a todos os comandos. No Paxos, o primário é chamado de líder.
- Suponha que os comandos possam sempre ser recuperados (seja a partir dos clientes ou dos servidores)

Exemplo com dois servidores



$< \sigma_i^j >$ é o envio do resultado de uma operação pelo Servidor S_i a um cliente, onde j representa o estado do servidor através da sequência de operações já realizadas.

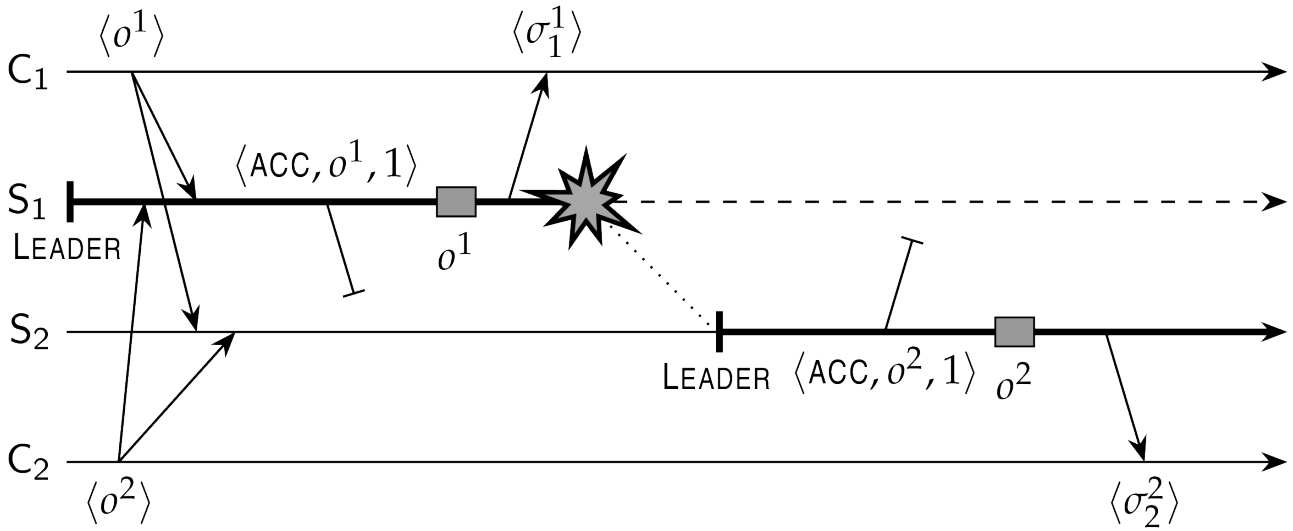
Observação: os clientes enviam as requisições para todos os servidores e são responsáveis por eliminar respostas duplicadas

Lidando com mensagens perdidas

Um pouco de terminologia do Paxos

- O líder envia uma mensagem de accept ACCEPT(o, t) para os backups ao atribuir um timestamp t ao comando o.
- Um servidor de backup responde enviando uma mensagem learn: LEARN(o, t)
- Quando o líder percebe que a operação o ainda não foi aprendida, ele retransmite ACCEPT(o, t) com o timestamp original.

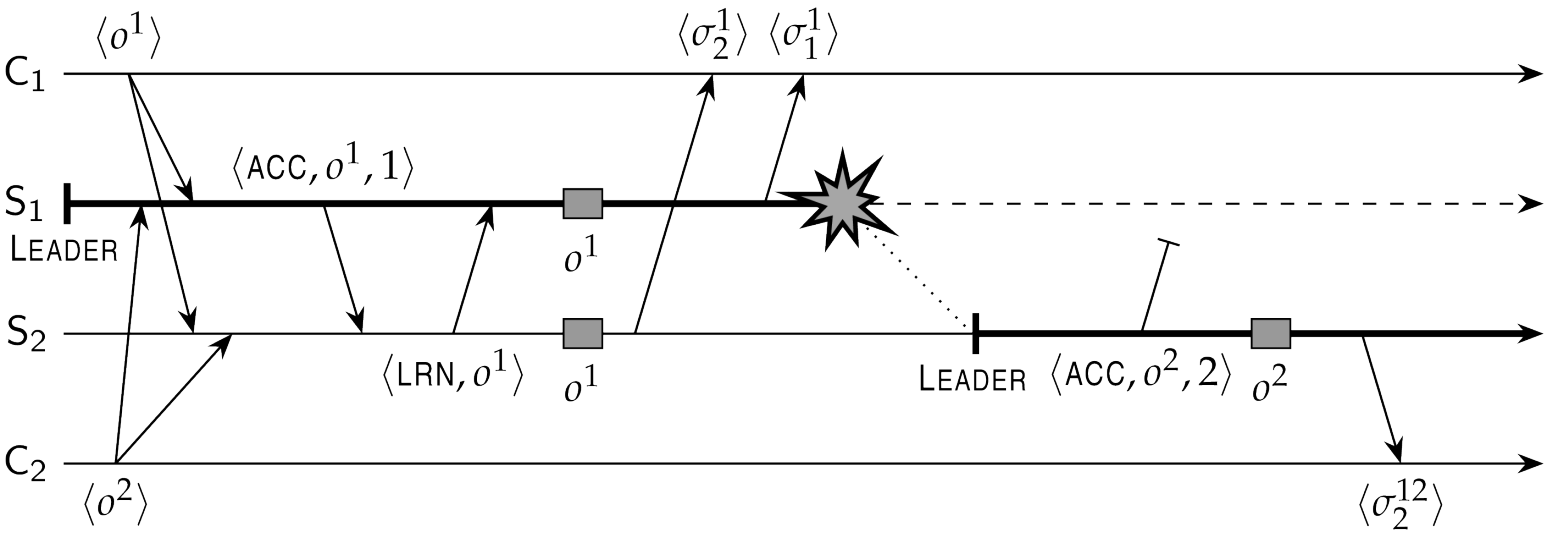
Dois servidores e um crash: problema



Problema

O primário falha após executar uma operação, mas o backup nunca recebeu a mensagem de accept.

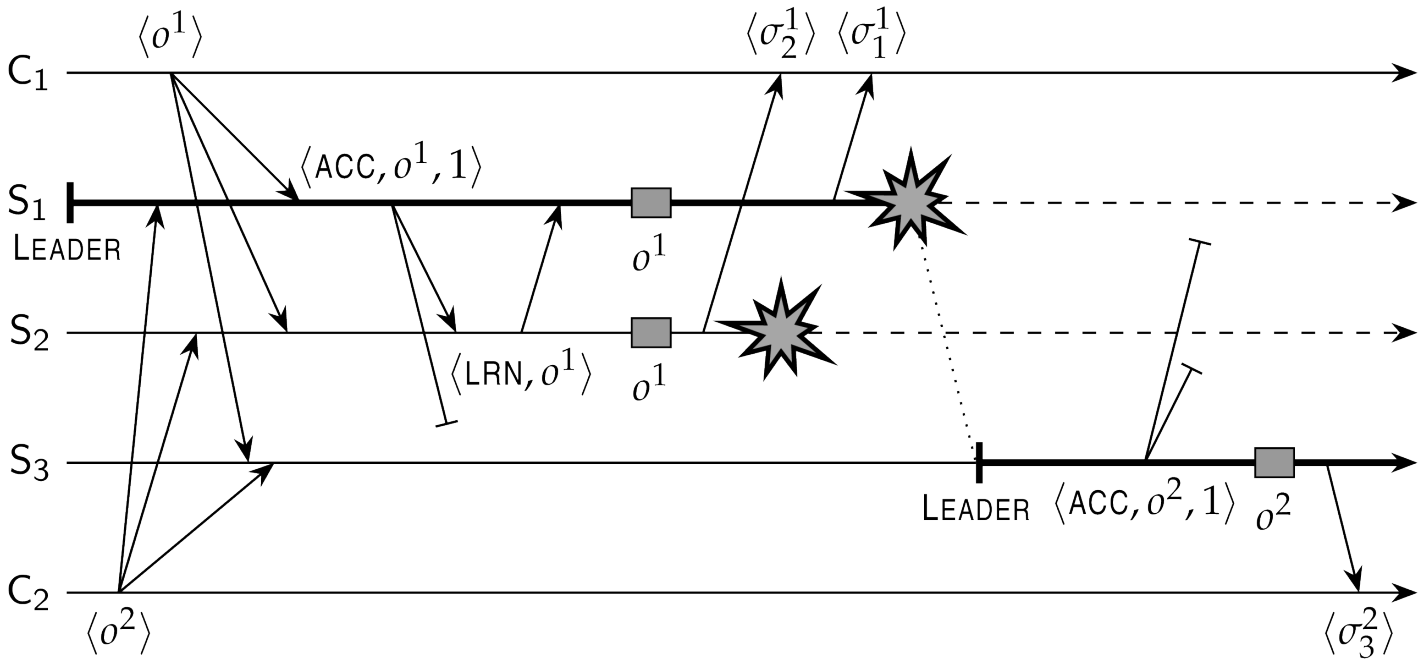
Dois servidores e um crash: solução



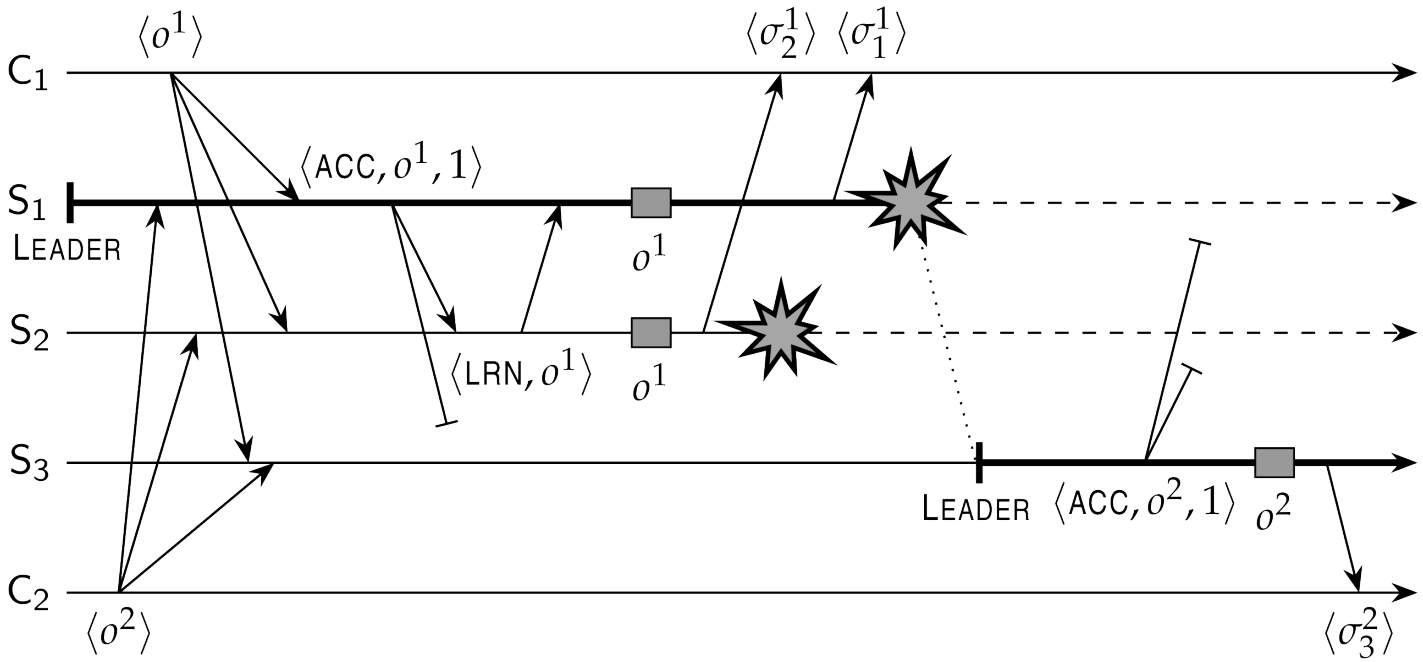
Solução

Nunca execute uma operação antes que esteja claro que ela foi aprendida.

Três servidores e duas falhas: ainda é problema?



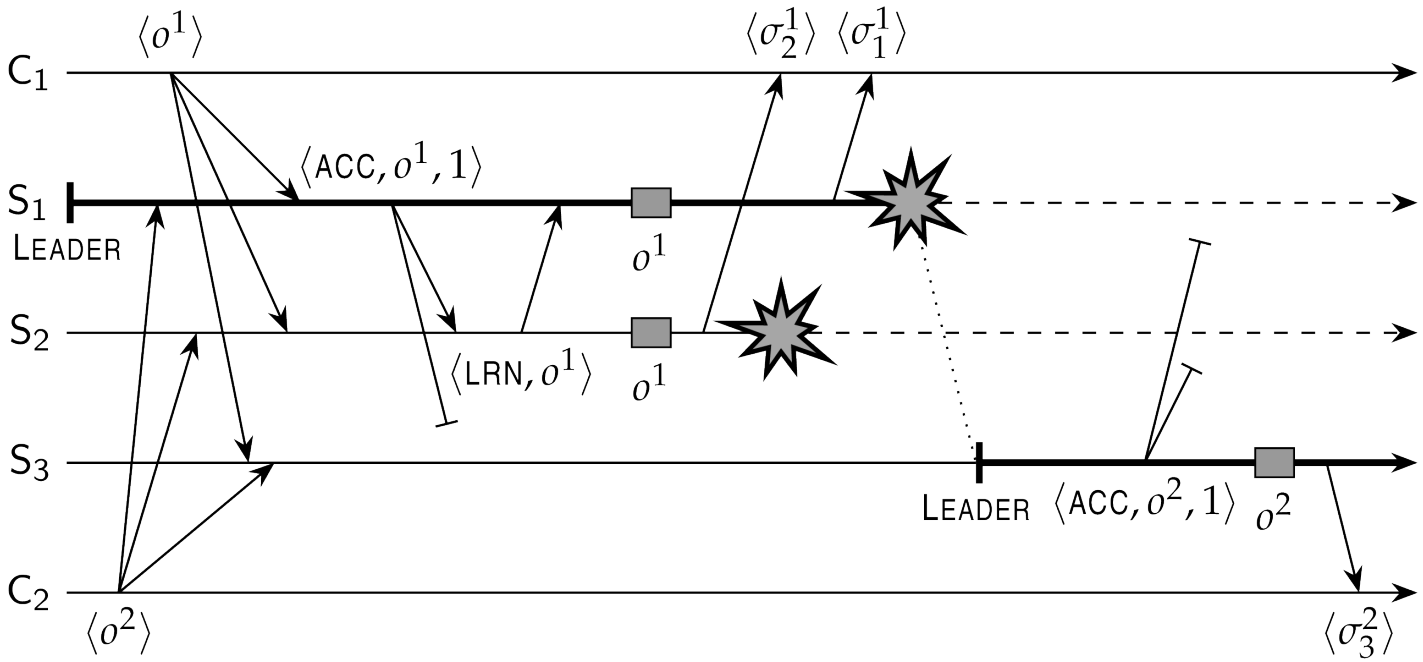
Três servidores e duas falhas: ainda é problema?



Cenário

O que acontece quando LEARN(o^1) enviado por S_2 para S_1 é perdido?

Três servidores e duas falhas: ainda é problema?



Cenário

O que acontece quando LEARN(o^1) enviado por S_2 para S_1 é perdido?

Solução

S_2 também terá que esperar até saber que S_3 aprendeu o^1.

Paxos: regra fundamental

Regra geral

No Paxos, um servidor S não pode executar uma operação o até que tenha recebido uma LEARN(o) de todos os outros servidores não defeituosos.

Detecção de falhas

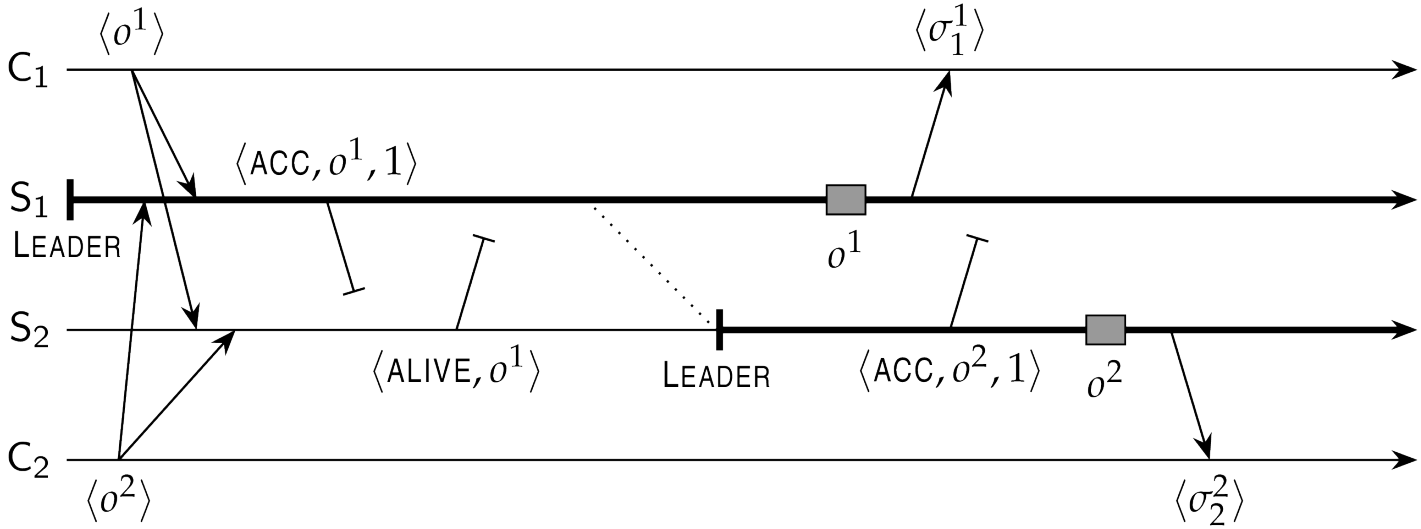
Prática

Detecção de falhas confiável é praticamente impossível. Uma solução é definir timeouts, mas considere que uma falha detectada pode ser falsa.

Detecção de falhas

Prática

Detecção de falhas confiável é praticamente impossível. Uma solução é definir timeouts, mas considere que uma falha detectada pode ser falsa.



Número necessário de servidores

Observação

Paxos precisa de pelo menos três servidores

Número necessário de servidores

Observação

Paxos precisa de pelo menos três servidores

Regra fundamental adaptada

No Paxos com três servidores, um servidor S não pode executar uma operação o até que tenha recebido pelo menos uma (outra) mensagem LEARN(o), para saber que a maioria dos servidores executará o.

Número necessário de servidores

Pressupostos antes de dar os próximos passos

- Inicialmente, S_1 é o líder.
- Um servidor pode detectar de forma confiável que perdeu uma mensagem e se recuperar dessa perda.
- Quando um novo líder precisa ser eleito, os servidores restantes seguem um algoritmo estritamente determinístico, e.g. $S_1 \rightarrow S_2 \rightarrow S_3$.
- Um cliente não pode ser solicitado a ajudar os servidores a resolver uma situação.

Número necessário de servidores

Pressupostos antes de dar os próximos passos

- Inicialmente, S_1 é o líder.
- Um servidor pode detectar de forma confiável que perdeu uma mensagem e se recuperar dessa perda.
- Quando um novo líder precisa ser eleito, os servidores restantes seguem um algoritmo estritamente determinístico, e.g. $S_1 \rightarrow S_2 \rightarrow S_3$.
- Um cliente não pode ser solicitado a ajudar os servidores a resolver uma situação.

Observação

Se qualquer um dos backups (S_2 ou S_3) falhar, o Paxos se comportará corretamente: operações nos servidores não defeituosos são executadas na mesma ordem.

Líder falha após executar o^1

Líder falha após executar o^1

S_3 está completamente ignorante de qualquer atividade de S_1

- S_2 recebeu ACCEPT$(o, 1)$, detecta falha e se torna líder.
- S_3 nem sequer recebeu ACCEPT$(o, 1)$.
- Se S_2 enviar ACCEPT$(o^2, 2) \Rightarrow S_3$ vê timestamp inesperado e informa S_2 que perdeu o^1.
- S_2 retransmite ACCEPT$(o^1, 1)$, permitindo que S_3 se atualize.

Líder falha após executar o^1

S_3 está completamente ignorante de qualquer atividade de S_1

- S_2 recebeu ACCEPT$(o,1)$, detecta falha e se torna líder.
- S_3 nem sequer recebeu ACCEPT$(o,1)$.
- Se S_2 enviar ACCEPT$(o^2,2) \Rightarrow S_3$ vê timestamp inesperado e informa S_2 que perdeu o^1.
- S_2 retransmite ACCEPT$(o^1,1)$, permitindo que S_3 se atualize.

S_2 perdeu ACCEPT$(o^1,1)$

- S_2 detectou a falha e se tornou o novo líder
- Se S_2 enviar ACCEPT$(o^1,1) \Rightarrow S_3$ retransmite LEARN(o^1).
- Se S_2 enviar ACCEPT$(o^2,1) \Rightarrow S_3$ informa S_2 que aparentemente perdeu ACCEPT$(o^1,1)$ de S_1, para que S_2 possa se atualizar.

Líder falha após enviar ACCEPT(o^1, 1)

S_3 está completamente ignorante de qualquer atividade de S_1

Assim que S_2 anunciar que o^2 deve ser aceito, S_3 perceberá que perdeu uma operação e pode pedir a S_2 ajuda para recuperar.

S_2 perdeu ACCEPT(o^1, 1)

Assim que S_2 propuser uma operação, usará um timestamp desatualizado, permitindo que S_3 informe S_2 que perdeu a operação o^1.

Líder falha após enviar ACCEPT(o^1, 1)

S_3 está completamente ignorante de qualquer atividade de S_1

Assim que S_2 anunciar que o^2 deve ser aceito, S_3 perceberá que perdeu uma operação e pode pedir a S_2 ajuda para recuperar.

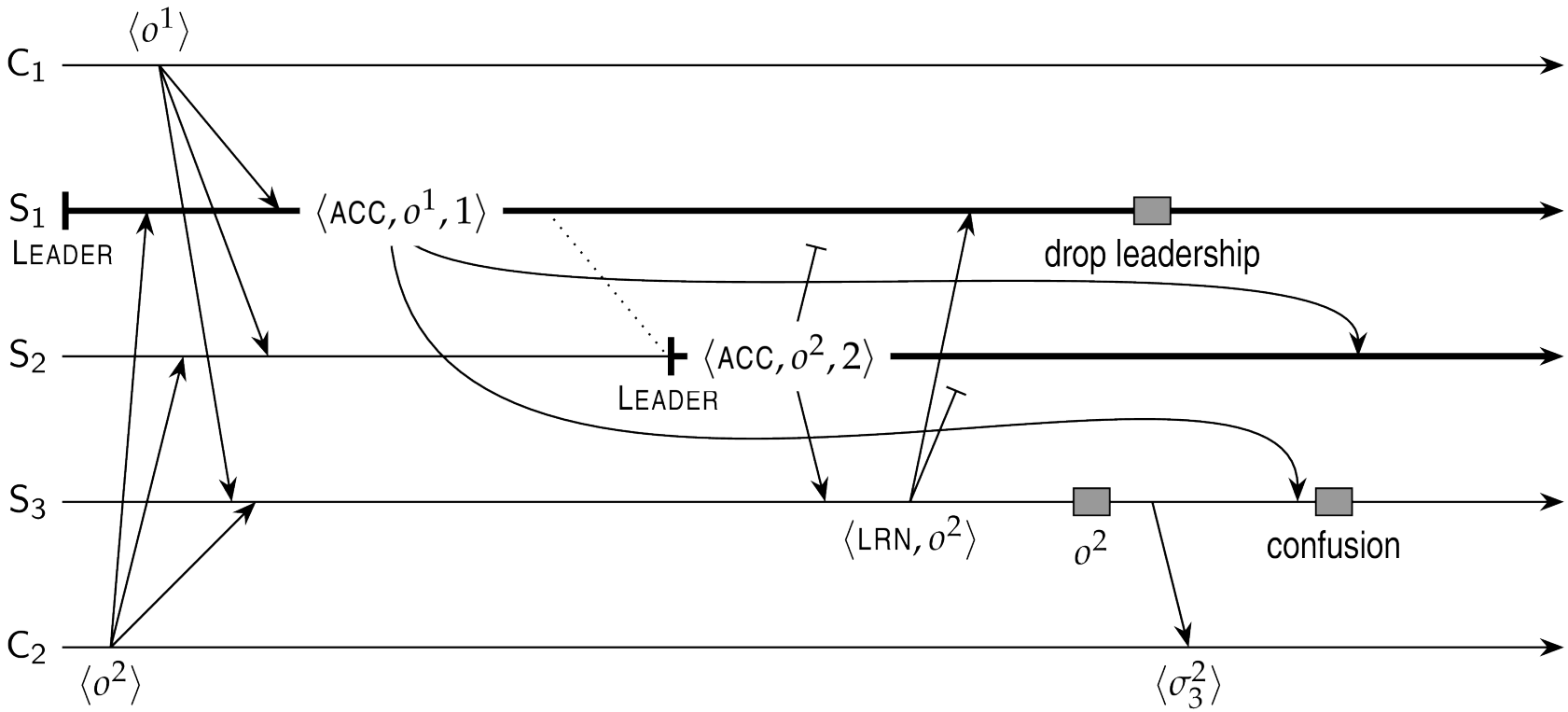
S_2 perdeu ACCEPT(o^1, 1)

Assim que S_2 propuser uma operação, usará um timestamp desatualizado, permitindo que S_3 informe S_2 que perdeu a operação o^1.

Observação

Paxos (com três servidores) se comporta corretamente quando um único servidor falha, independentemente de quando essa falha ocorreu.

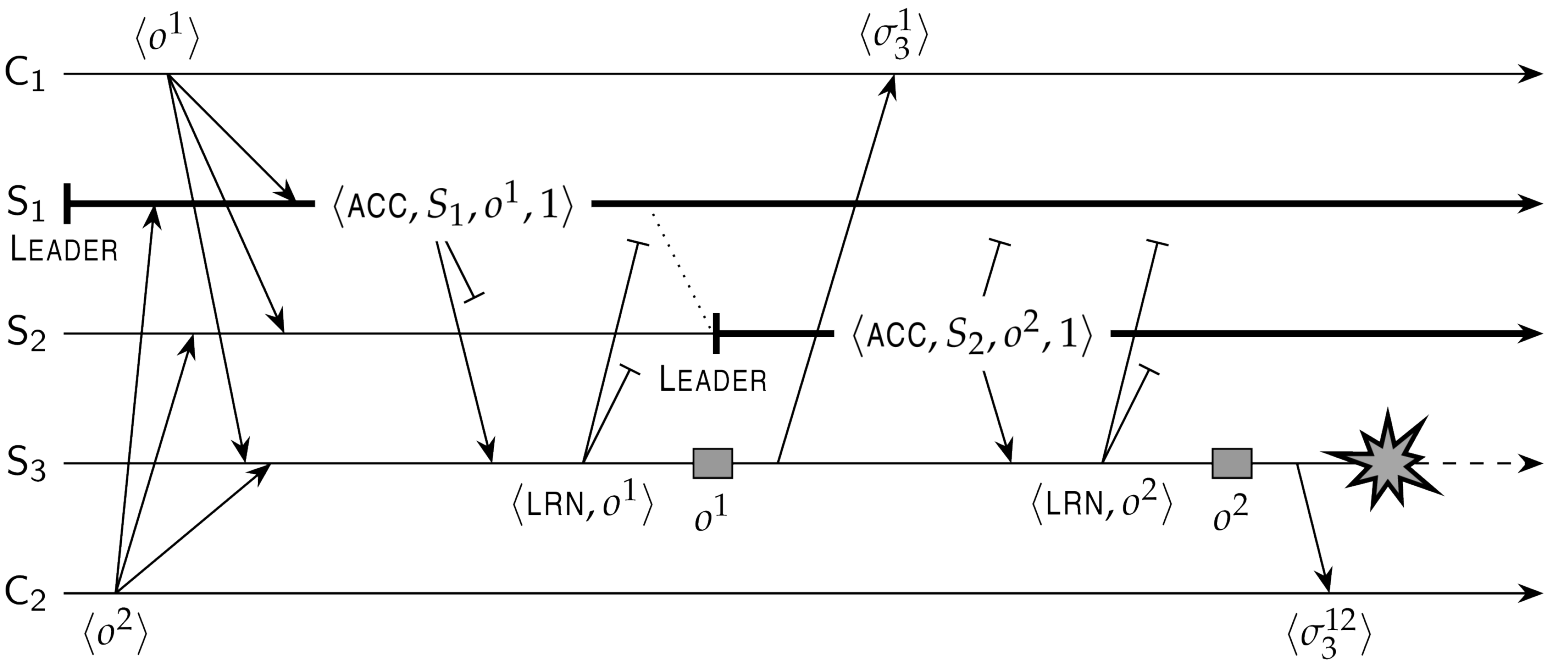
Falsos positivos na detecção de falha



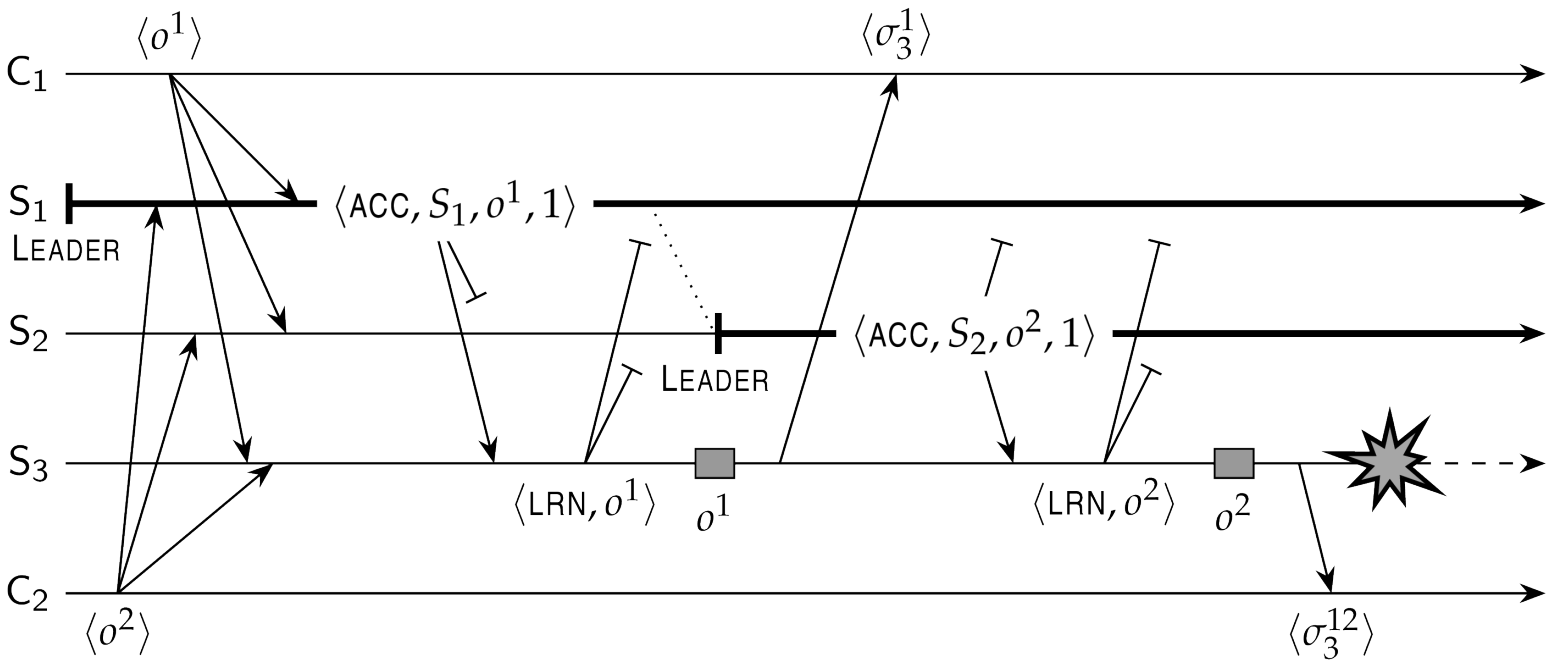
Problema e solução

S_3 recebe ACCEPT(o^1,1), mas muito mais tarde do que ACCEPT(o^2,1). Se S_3 soubesse quem era o líder atual, poderia rejeitar com segurança a mensagem de accept atrasada $\Rightarrow$ líderes devem incluir sua ID nas mensagens.

Situação em que não é possível progredir



Situação em que não é possível progredir



Essência da solução

Quando S_2 assume o controle, precisa garantir que quaisquer operações pendentes iniciadas por S_1 tenham sido devidamente finalizadas, ou seja, executadas por servidores suficientes. Isso requer uma tomada de liderança explícita pela qual os outros servidores são informados antes de enviar novas mensagens de accept.