

CONSISTÊNCIA E CONSENSO

CONSTRUINDO SISTEMAS DISTRIBUÍDOS TOLERANTES A FALHAS

Aluísio Augusto Silva Gonçalves

17 de maio de 2018

CONTEÚDO

- Recapitulando
- Garantias de consistência
- Linearizabilidade
- Garantias de ordenação
- Transações Distribuídas
- Consenso
- Associação e coordenação
- Sumário
- Questões

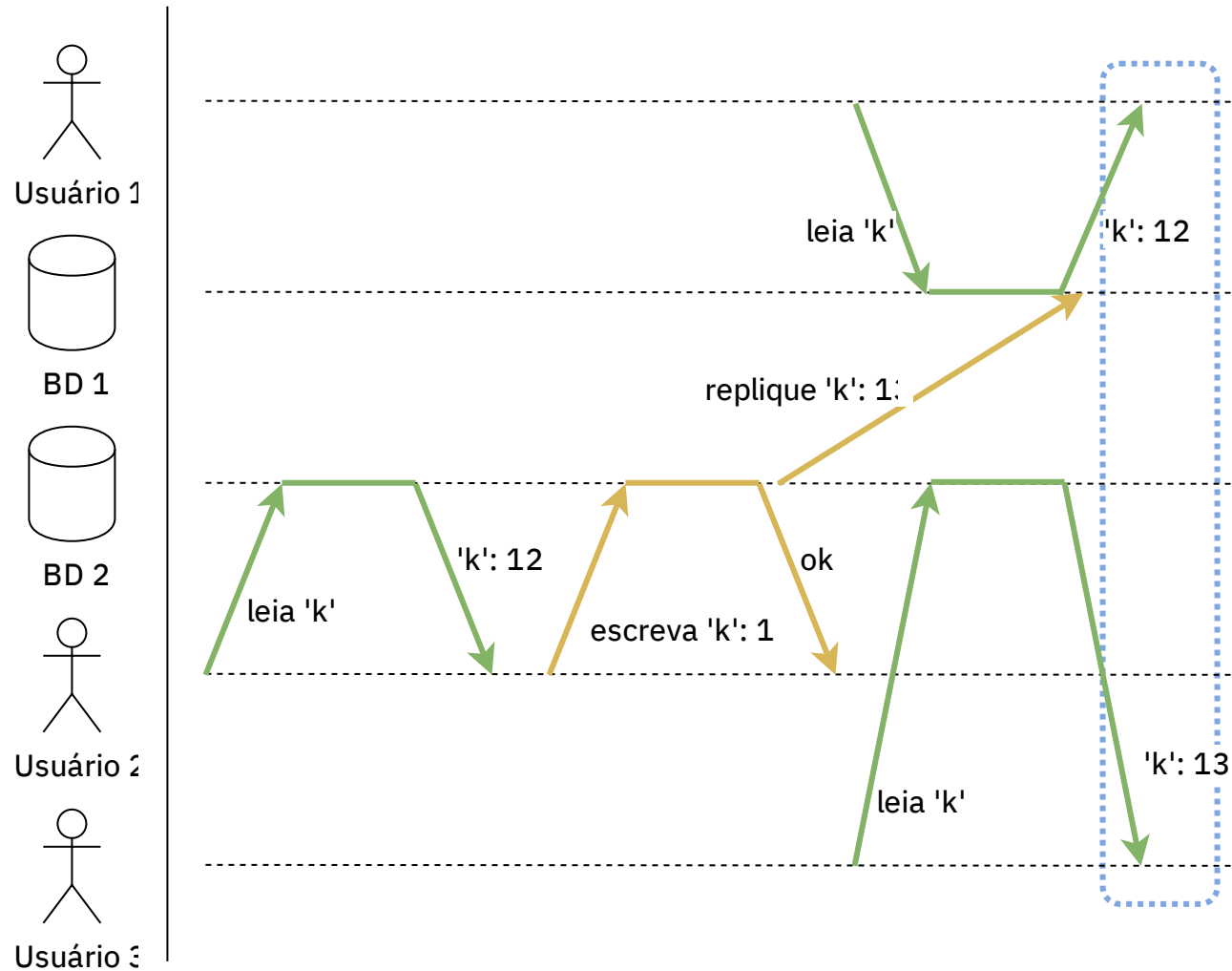
RECAPITULANDO

PROBLEMAS COM SISTEMAS DISTRIBUÍDOS

- Comunicação: atraso/reordenação/duplicação/perda de pacotes
- Ordenação: relógios imprecisos ou dessincronizados
- Disponibilidade: nodos podem pausar ou falhar a qualquer momento

GARANTIAS DE CONSISTÊNCIA

ATRASO DE REPLICAÇÃO



Mesmo instante, nodos diferentes, dados diferentes.

MODELOS DE CONSISTÊNCIA

Consistência eventual

Em *algum momento* após uma escrita, todos os nodos terão o mesmo dado.

Linearizabilidade

Após uma escrita, todas as leituras (em qualquer nodo) retornam o valor escrito.

LINEARIZABILIDADE

OBJETIVO

Fazer parecer que há apenas uma cópia dos dados, abstraindo da aplicação/usuário a existência de réplicas (e os problemas que vêm com elas).

Todas as leituras e escritas podem ser mapeadas para uma única linha do tempo.

UTILIDADE

- *Locks*
- Restrição de unicidade
- Interação entre sistemas

CONDIÇÕES

- Após a conclusão de uma escrita, todas as leituras retornam o valor novo
 - Mas e *durante* a escrita?
- Após uma leitura do valor novo, todas as leituras seguintes também retornam o valor novo
 - Ou seja, as operações são atômicas

IMPLEMENTANDO SISTEMAS LINEARIZÁVEIS

1. Tenha apenas uma cópia dos dados, isto é, não utilize replicação
2. Utilize um mecanismo de replicação linearizável

REPLICAÇÃO LINEARIZÁVEL

Replicação com líder único (possivelmente linearizável)

- Isolamento por *snapshot* não é linearizável
- Requer replicação síncrona

Replicação por consenso (linearizável)

- Inclui medidas para evitar os problemas da replicação com líder único

REPLICAÇÃO LINEARIZÁVEL

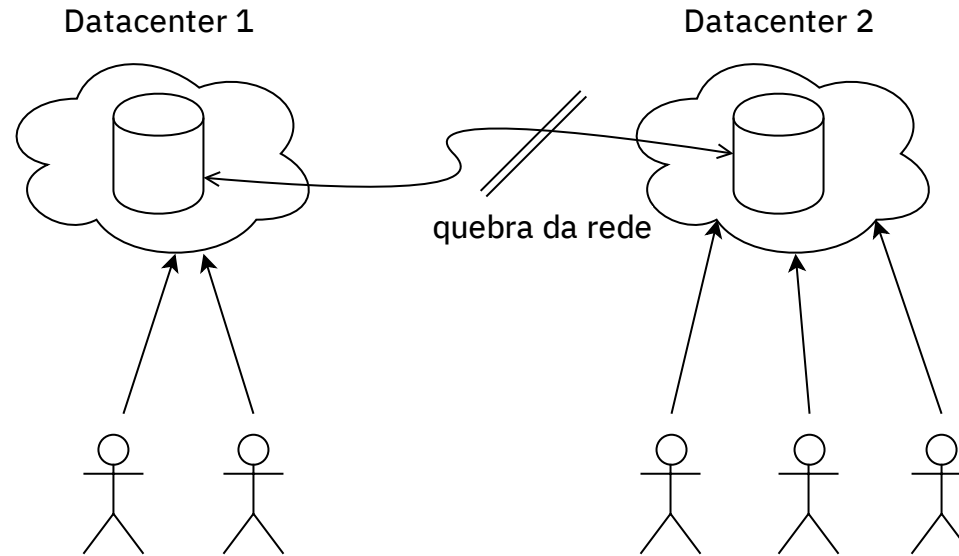
Replicação multi-líder (não linearizável)

- Escritas em paralelo com replicação assíncrona podem levar à conflitos

Replicação sem líder (provavelmente não-linearizável)

- Quórum não é afetado por atraso de replicação, mas leituras são
- Ordenação por relógio físico/*time-of-day* pode não ser a ordem real dos eventos

O TEOREMA CAP



Consistência OU disponibilidade quando particionado.

O TEOREMA CAP

- Se a aplicação *requer* linearizabilidade, e algumas réplicas perdem acesso às outras, as réplicas se tornam indisponíveis para o processamento de requisições.
- Se a aplicação *não requer* linearizabilidade, e algumas réplicas perdem acesso às outras, requisições ainda podem ser processadas independentemente, ao custo da linearizabilidade.

LINEARIZABILIDADE E REDES INSTÁVEIS

- Linearizabilidade reduz o desempenho de um sistema
- O tempo de resposta em um sistema linearizável é proporcional à incerteza sobre os atrasos na rede
- Alta variabilidade \Rightarrow maior tempo de resposta

GARANTIAS DE ORDENAÇÃO

ORDEM É IMPORTANTE

- O líder na replicação com líder único determina a *ordem* na qual seus seguidores executam operações de escrita
- Serializabilidade garante que transações são executadas *como-se* fossem executadas em alguma *ordem* sequencial
- Relógios são um modo de *ordenar* escritas

ORDEM E CAUSALIDADE

- Causalidade é uma ordenação *parcial*
 - Uma pergunta *acontece antes* de sua resposta
 - A criação de um registro *acontece antes* de sua modificação
 - Duas mensagens simultâneas de usuários diferentes não podem ser ordenadas (são *concorrentes*)
- Sistemas que respeitam essa ordenação causal são *causalmente consistentes*

ORDEM E LINEARIZABILIDADE

- Linearizabilidade é uma ordenação *total*
 - Todos os eventos ocorrem em uma única linha do tempo
 - Não existem eventos *concorrentes* (ou seja, não-ordenáveis)
- Linearizabilidade implica em consistência causal

CAUSALIDADE E LINEARIZABILIDADE

- Consistência causal é uma garantia mais fraca que linearizabilidade
- O teorema CAP não se aplica à sistemas apenas causalmente consistentes
- O tempo de resposta de sistemas apenas causalmente consistentes não depende diretamente da qualidade da rede

RASTREANDO DEPENDÊNCIAS CAUSAIS COM NÚMEROS DE SEQUÊNCIA

- Associa-se um número de sequência à cada chave, que é incrementado com cada modificação
- Números de sequência impõem uma ordenação total:
 - Operações ordenáveis causalmente mantêm esta ordem
 - Operações concorrentes são ordenadas em *alguma* ordem

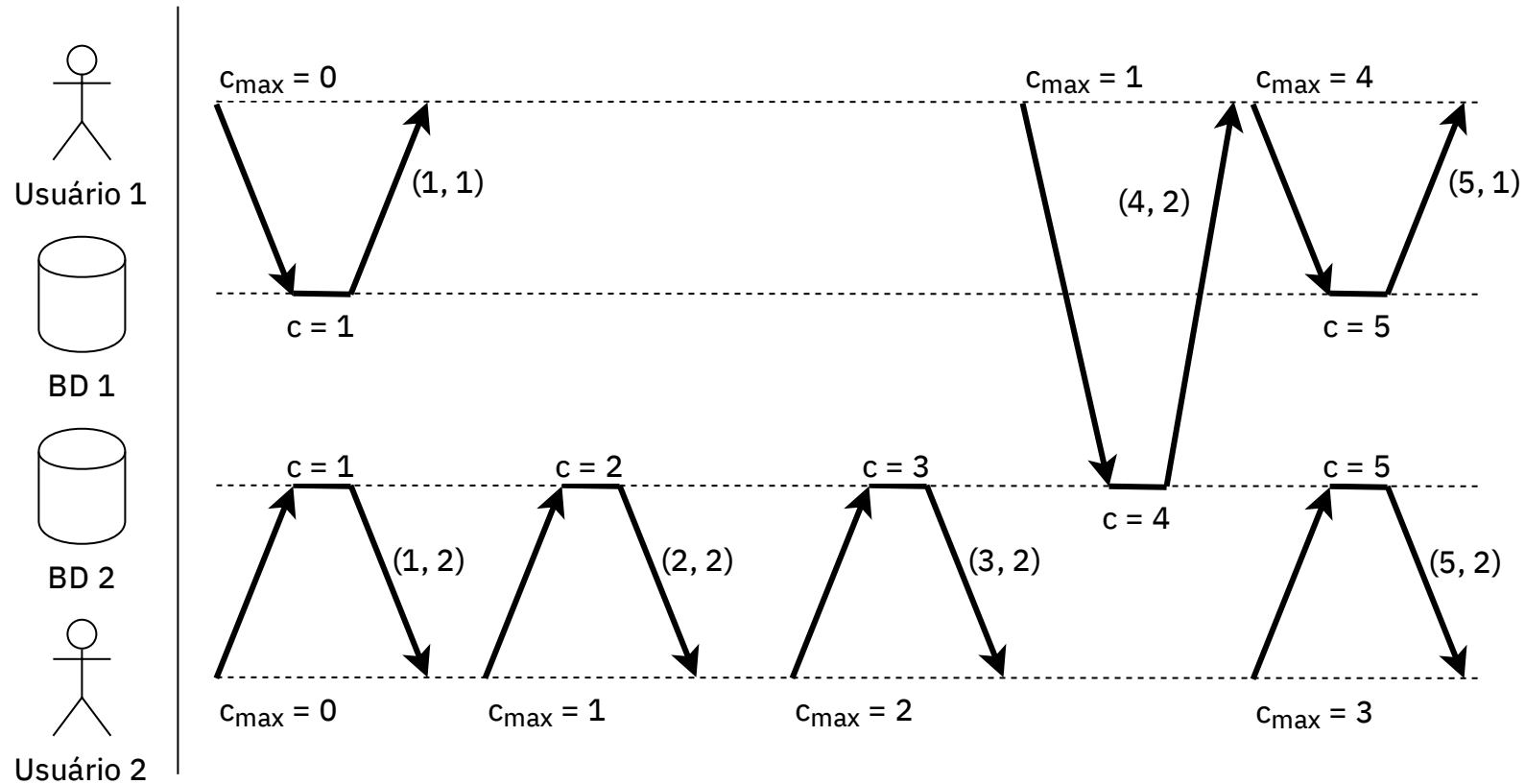
GERANDO NÚMEROS DE SEQUÊNCIA

- Quando há um líder único: o líder incrementa o número associado à chave após uma escrita
- Em outros casos:
 - Geração de sequências disjuntas de números por nodo
 - *Timestamps* de alta resolução
 - Pré-alocação de lotes de números de sequência por nodo
 - Nenhum destes é causalmente consistente!

ORDEM TOTAL DE LAMPORT

- Nodos e clientes mantêm um contador individual
- Cada requisição inclui o contador do cliente
- Cada resposta inclui um par (contador do nodo, ID do nodo) \leftarrow *timestamp* da operação
- Ao receber uma requisição ou resposta com contador maior que o seu próprio, este é incrementado para além do que foi recebido

ORDEM TOTAL DE LAMPORT



Dependências causais resultam em um incremento do contador

ORDENAÇÃO DE *TIMESTAMPS* DE LAMPOR

- Ordenação total é possível através do par (contador do nodo, ID do nodo)
- A ordenação só pode ser feita com conhecimento das operações de todos os nodos
 - Não é o bastante para garantir unicidade

BROADCAST DE ORDENAÇÃO TOTAL / BROADCAST ATÔMICO

Entrega confiável

Se uma mensagem é entregue à qualquer nodo, então ela é entregue à todos os nodos

Entrega totalmente ordenada

Mensagens são entregues à todos os nodos na mesma ordem

Estas propriedades devem ser sempre mantidas, ou atingidas eventualmente no caso de falhas na rede ou em algum nodo.

BROADCAST ATÔMICO - UTILIDADE

- Consenso
- Consistência entre réplicas (salvo atraso de replicação)
- Transações serializáveis
- *Log* de operações/replicação/transações/*write-ahead*
- Serviço de *lock*

LINEARIZAÇÃO DE ESCRITAS COM *BROADCAST ATÔMICO*

Estudo de caso: reivindicando um nome de usuário

1. Envie uma mensagem reivindicando o nome de usuário desejado.
2. Aguarde o recebimento desta mesma mensagem.
3. Caso alguma outra reivindicação seja recebida nesse ínterim, desista.
4. Do contrário, a operação foi concluída com sucesso.

Todos os nodos concordam sobre quais operações ocorreram e sua ordem.

LINEARIZAÇÃO DE LEITURAS COM *BROADCAST* ATÔMICO

Opção 1, usando *broadcast* atômico para marcar uma posição no tempo:

1. Envie uma mensagem qualquer.
2. Aguarde a mensagem retornar.
3. Efetue a leitura.

LINEARIZAÇÃO DE LEITURAS COM *BROADCAST ATÔMICO*

Opção 2, caso as mensagens sejam indexadas:

1. Obtenha o índice da última mensagem recebida.
2. Aguarde até que todas as mensagens com índice menor sejam recebidas.
3. Efetue a leitura.

LINEARIZAÇÃO DE LEITURAS COM *BROADCAST ATÔMICO*

Opção 3, faça a leitura em uma réplica síncrona.

BROADCAST ATÔMICO COM LINEARIZAÇÃO

1. Execute uma operação incremente-e-leia em um contador linearizado.
2. Use o valor lido como número de sequência.
3. Envie a mensagem para todos os nodos, re-enviando em caso de perda.
4. Ao receber uma mensagem, aguarde até ter entregue à aplicação a mensagem com contador imediatamente anterior antes de entregar a mensagem atual.

TRANSAÇÕES DISTRIBUÍDAS

COMMIT ATÔMICO DISTRIBUÍDO

Em um único nodo, o SGBD escreve as alterações em um log e marca a transação como finalizada no log. Caso haja uma falha *antes* da marcação ser gravada com sucesso, a transação é revertida.

Mas e quando múltiplos nodos estão envolvidos? Por exemplo, nodos com partes distintas dos dados (banco de dados particionado)?

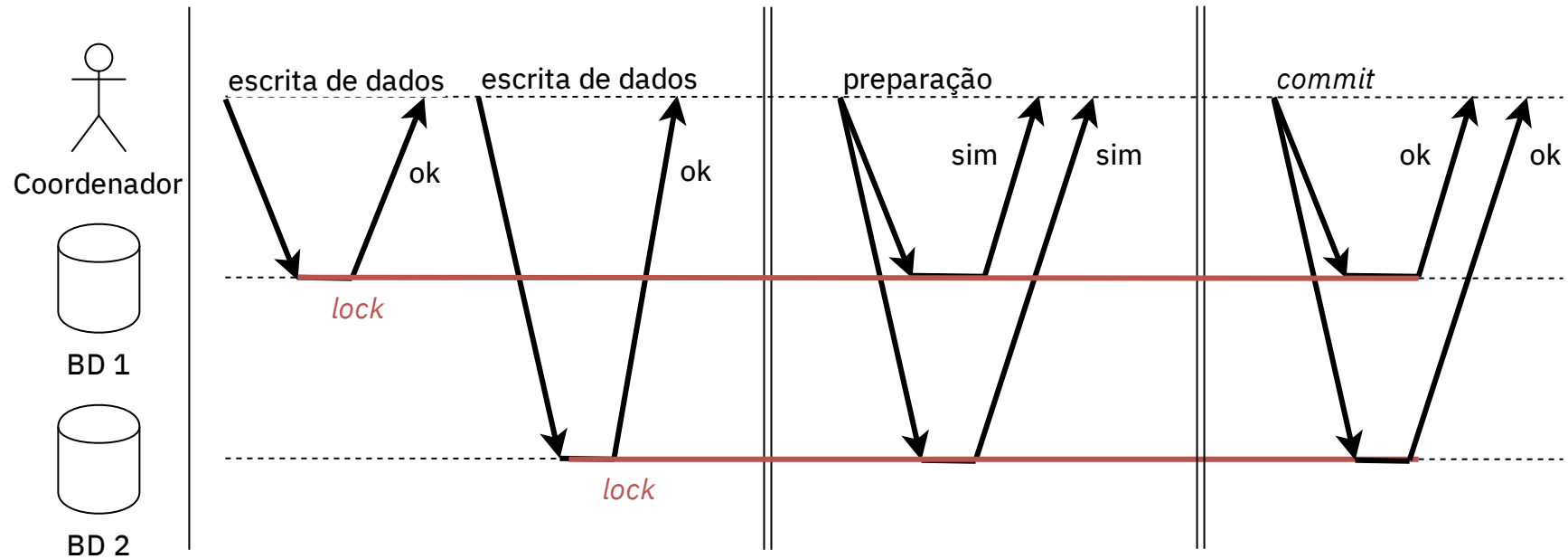
COMMIT EM DUAS FASES (2PC)

- Objetivo: garantir que ou todos os nodos fazem *commit*, ou todos abortam a transação
- Requer um novo componente da aplicação: o *coordenador* ou *gerenciador de transações*
 - Pode ser parte do cliente ou um serviço separado

2PC - ALGORITMO

1. A aplicação solicita ao coordenador uma transação.
2. A aplicação escreve os dados, então solicita ao coordenador que faça *commit* da transação.
3. O coordenador envia um aviso de *preparação* aos nodos envolvidos (*participantes*) na transação.
4. O coordenador aguarda as respostas de todos os nodos participantes quanto a ser possível prosseguir.
5. Caso todos indiquem ser possível o *commit*, uma mensagem de *commit* é enviada. Do contrário, a transação é abortada.

2PC - EXEMPLO



Exemplo de uma execução com sucesso do *commit* em duas fases

2PC - PONTOS SEM VOLTA

1. Um nodo participante que responda Sim ao coordenador após uma solicitação de preparação garante que pode fazer *commit* da transação (mesmo que o nodo venha a falhar);
2. A decisão tomada pelo coordenador após a fase de preparação é imutável.

2PC - FALHA DO COORDENADOR

- Caso o coordenador falhe antes da fase de preparação, os participantes podem abortar a transação
- Caso o coordenador falhe antes da fase de *commit*, os participantes **devem** aguardar a volta do coordenador
 - A transação é mantida em um estado *indeciso* até lá, sem ser abortada nem ser feito *commit*
 - Quaisquer *locks* obtidos para a transação precisam ser mantidos

2PC - FALHA DO COORDENADOR

Caso o coordenador não consiga decidir se deve prosseguir uma transação (p.ex. se o log for perdido ou corrompido), as transações não-decidas se tornam *órfãs*, requerendo intervenção manual.

COMMIT EM TRÊS FASES

- 2PC tem no coordenador um ponto único de falha
- 3PC introduz uma fase de *pré-commit* e *timeouts* de operação
 - Uso de *timeouts* impede a garantia de atomicidade

TRANSAÇÕES DISTRIBUÍDAS HETEROGÊNEAS

- Transações distribuídas internas a um SGBD podem utilizar protocolos e otimizações específicos àquela tecnologia
- Transações distribuídas envolvendo sistemas distintos são mais complicadas

EXEMPLO: PROCESSAMENTO ÚNICO DE MENSAGENS

Em uma mesma transação:

- Marque a mensagem como processada na fila de mensagens
- Processe a mensagem
 - Requer que todos os sistemas envolvidos suportem 2PC

Caso haja algum erro, a transação é abortada e o processamento revertido. Assim, garante-se que cada mensagem é processada *exatamente* 1 vez.

TRANSAÇÕES XA

- X/Open *eXtended Architecture*
- Especificação para 2PC sobre sistemas heterogêneos
 - Menor denominador comum: sem detecção de *deadlocks*, sem suporte a isolamento por *snapshot* serializável...
- API para interface com coordenadores de transação
 - Para uso por *drivers* ou bibliotecas

CONSENSO

DESCRIÇÃO

- Um ou mais nodos *propõem* um valor
- O algoritmo de consenso garante que *todos* os nodos escolham *um mesmo valor*

UTILIDADE

- Eleição de um líder
- *Commit* atômico de transação

PROPRIEDADES

Concordância uniforme

Dois nodos quaisquer não tomam decisões diferentes.

Integridade

Nenhum nodo decide duas vezes.

Validade

Se um nodo decide por um valor v , então v foi proposto por algum nodo.

Terminação

Todo nodo não-falho eventualmente decide por algum valor.

MODELO DE FALHAS

- Falhas em nodos são assumidas *crash-stop* (o nodo para de responder) e permanentes (o nodo não se recupera)
- Pelo menos metade dos nodos devem estar em funcionamento correto para garantia de terminação

CONSENSO E *BROADCAST* ATÔMICO

- Muitas implementações de consenso decidem sobre uma *sequência* de valores
- Esta sequência pode ser vista como uma ordem de entrega de mensagens:
 - Concordância \Rightarrow todos os nodos decidem entregar mensagens na mesma ordem
 - Integridade \Rightarrow mensagens não são duplicadas
 - Validade \Rightarrow mensagens não são corrompidas ou “inventadas”
 - Terminação \Rightarrow mensagens não são perdidas
- \therefore consenso \Rightarrow *broadcast* atômico

CONSENSO E ELEIÇÃO DE LÍDERES

10. Eleição de um líder único requer consenso
11. O algoritmo de consenso é um *broadcast* atômico
12. *Broadcast* atômico é similar a replicação com líder único
13. Replicação com líder único requer um líder
14. GOTO 10

CONSENSO E ELEIÇÃO DE LÍDERES

1. Cada eleição de líder recebe um *número de época*
2. Quando o líder atual aparenta estar falho, uma nova eleição é realizada
3. Para tomar uma decisão, o líder envia uma proposta e seu número de época aos outros nodos e aguarda as respostas
4. Um nodo responde a favor somente se não tiver visto um líder com número de época maior
5. Um quórum de nodos deve responder a favor para confirmar a decisão

LIMITAÇÕES

- Similar à replicação síncrona
- Requer a maioria dos nodos não-falhos
- Assume (em geral) um conjunto estático de nodos participantes
- Utilizam (em geral) *timeouts* para detectar líderes falhos

ASSOCIAÇÃO E COORDENAÇÃO

SERVIÇOS DE COORDENAÇÃO

- Concentram tarefas como consenso, ordenação, e detecção de falhas
- Úteis para delegação de consenso em sistemas com *muitos* nodos, sem reduzir a um único ponto de falha

SERVIÇOS DE ASSOCIAÇÃO

- Determinam quais nodos estão ativos e em funcionamento
- Combinam detecção de falhas e consenso
- Podem declarar um nodo ativo como falho, mas são úteis mesmo assim

SUMÁRIO

CONCEITOS

- Linearizabilidade
- Causalidade
- Consenso

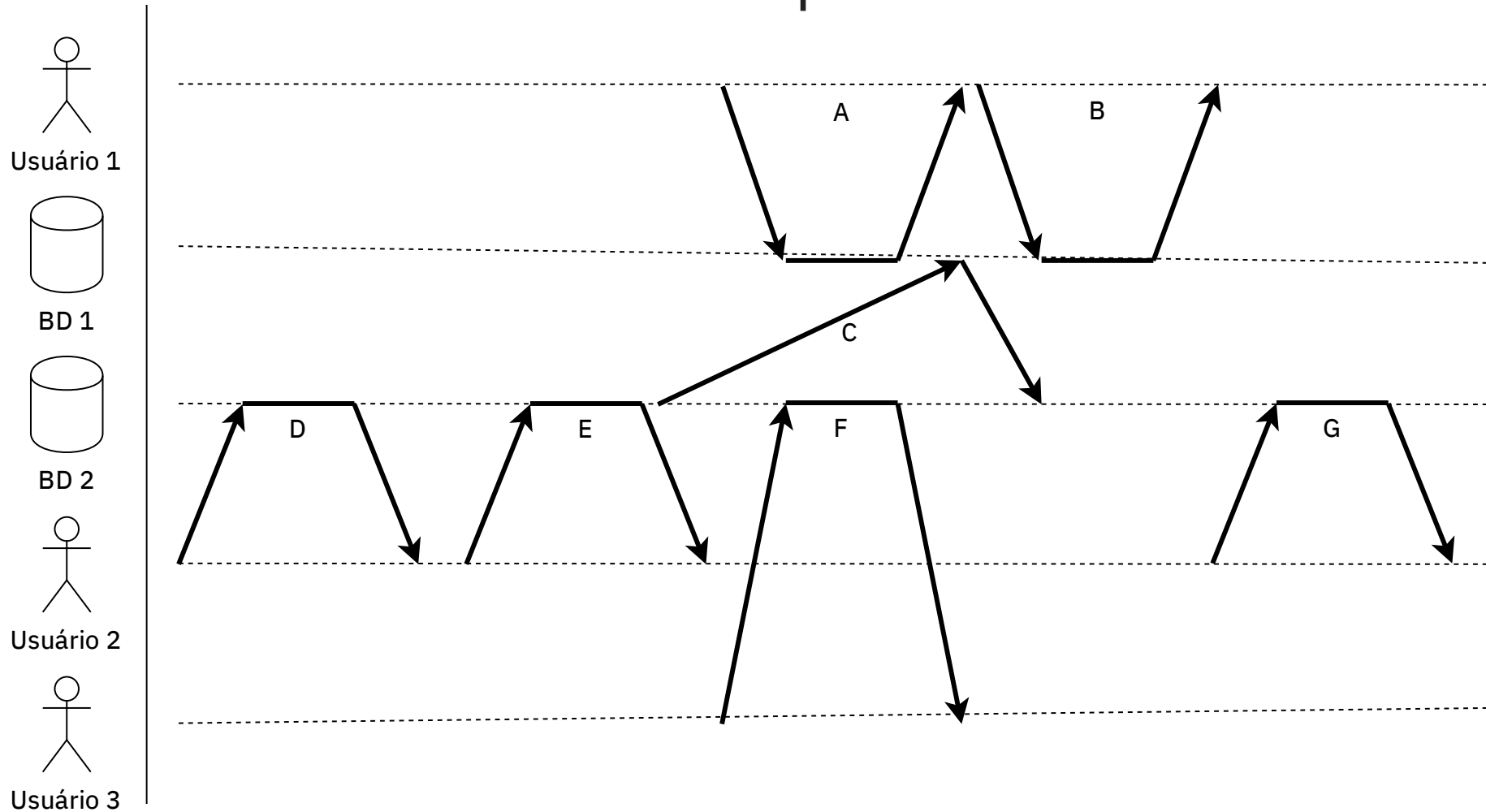
PROBLEMAS REDUTÍVEIS À CONSENSO

- Registradores CAS linearizáveis
- *Commit* atômico
- *Broadcast* atômico
- *Locks*
- Serviços de coordenação e associação
- Restrição de unicidade

QUESTÕES

QUESTÃO 1

Atribua *timestamps* de Lamport e ordene as operações identificadas por letras.



QUESTÃO 2

Explique a relação entre consenso, *broadcast* atômico, e linearizabilidade. Quais as consequências dessa relação?

DÚVIDAS?

`aasg13@inf.ufpr.br`