Replicação

Cleide Luzia Bonfim Possamai
03/05/2018

Agenda

- Conceito
- Motivação
- Principais abordagens
  - Replicação *single-leader*
  - Replicação *multi-leader*
  - Replicação *leaderless*
- Modelos de consistência
- Conclusão

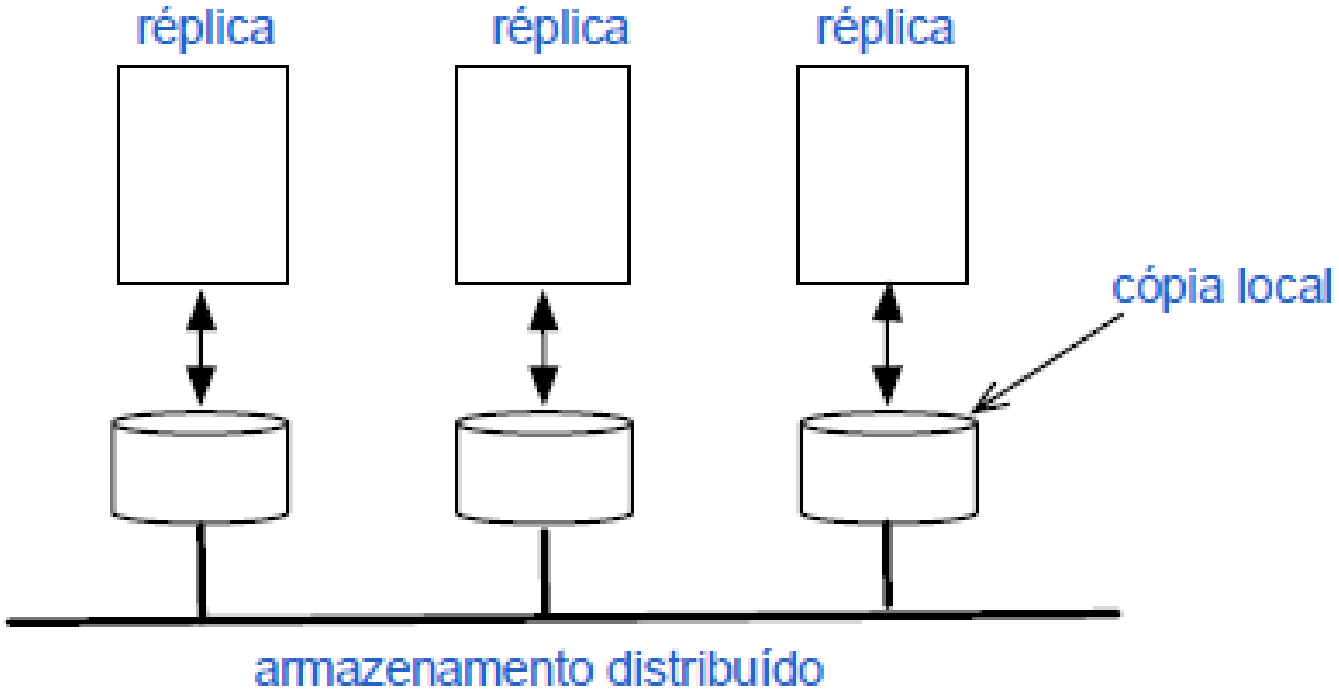
Conceito

- Replicação de dados consiste em manter cópias de dados em várias máquinas (**réplicas***), conectadas através de uma rede.

(*)Também chamadas de seguidoras, escravas, secundárias ou *hot standbys)*

Conceito

- Cada nó que armazena uma cópia é chamado de réplica.
- As atualizações precisam ser refletidas em todas as réplicas.



Motivação

- **Menor latência**: manter os dados geograficamente perto dos usuários.
- **Maior disponibilidade**: Permitir que o sistema continue funcionando mesmo que uma ou várias máquinas falhem.
- **Escalabilidade**: permite que as operações de leitura sejam feitas nas réplicas.

Motivação

- Apesar de ser conceitualmente simples, replicação de dados é uma tarefa desafiadora.
- Nós indisponíveis
- Interrupções de rede

Principais abordagens

- A mais comum é a replicação baseada no líder
  - *single leader* (também conhecida como réplica ativo/passivo ou réplica mestre/escravo)

Replicação baseada no líder

- Uma réplica é designada como líder.
- Clientes enviam todas as operações de escrita para um único nó (líder) que envia essas alterações para as réplicas (*followers*)
- Cada réplica recebe o log e atualiza sua cópia local dos dados na mesma ordem em que ocorreram no líder.

Replicação baseada no líder

- As operações de leitura podem ser feitas na máquina líder ou nas réplicas.
- Escrita é somente na máquina líder. As seguidoras são somente-leitura do ponto de vista do cliente.

Replicação baseada no líder

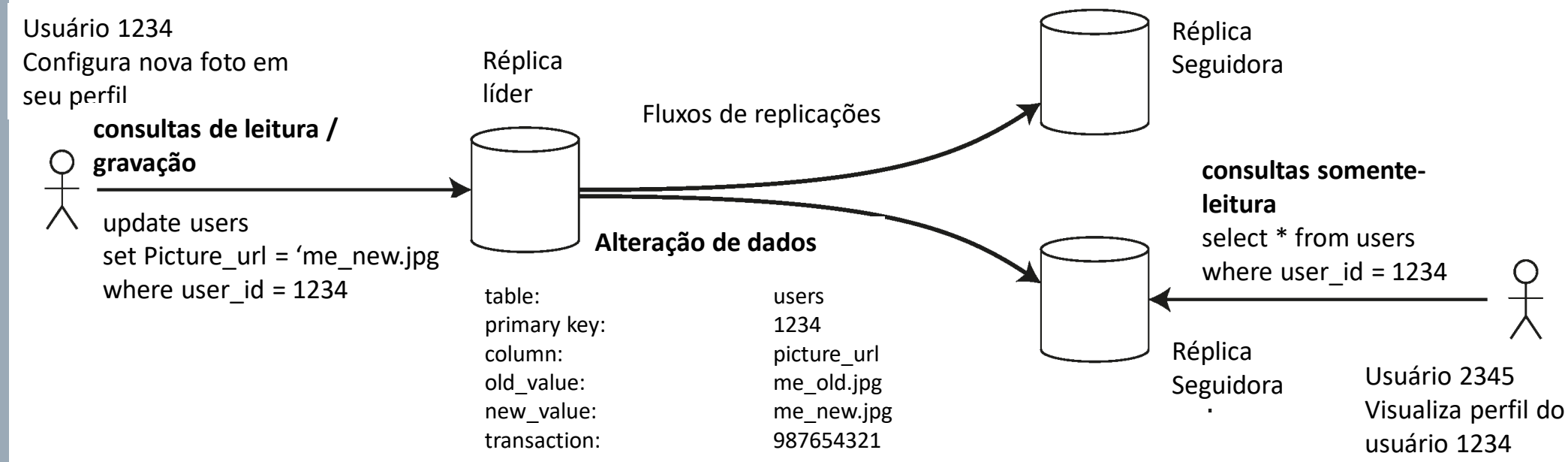


Figura 2 - Replicação centrada no lider (mestre-escravo ou ativo-passivo)

Replicação baseada no líder

- Esse modelo é nativo em muitos BD relacionais: PostgreSQL (desde a versão 9.0), MySQL, Oracle Data Guard, SQL Server's AlwaysOn Availability Groups.
- Em alguns não relacionais: MongoDB, RethinkDB e Espresso.

Replicação Síncrona *versus* Assíncrona

- Um ponto importante a ser considerado é o momento em que a replicação ocorre: de forma síncrona ou assíncrona.
- Em BD relacionais normalmente é configurável
- Em outros sistemas é *hardcoded*

Replicação Síncrona *versus* Assíncrona

- Um ponto importante a ser considerado é o momento em que a replicação ocorre: de forma síncrona ou assíncrona.
- Em BD relacionais normalmente é configurável
- Em outros sistemas é *hardcoded*

Replicação Síncrona *versus* Assíncrona

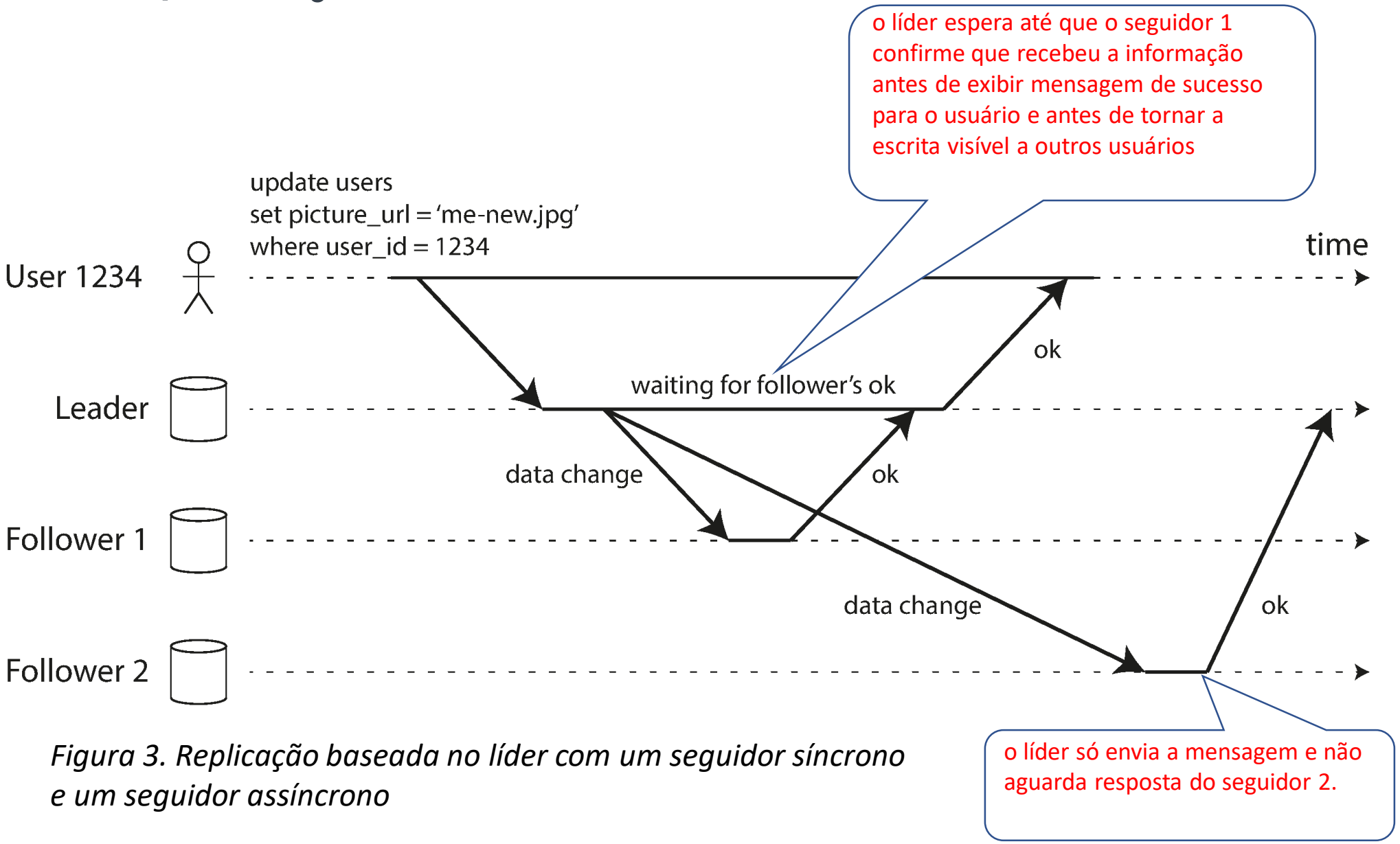


Figura 3. Replicação baseada no líder com um seguidor síncrono e um seguidor assíncrono

Replicação Síncrona *versus* Assíncrona

- A vantagem da replicação síncrona é a garantia de que o seguidor terá uma cópia atualizada e consistente com a do líder.
- Se o líder falhar, existe uma cópia no seguidor.
- Uma desvantagem é se o seguidor falha e não responde rapidamente, a escrita não pode ser processada.
- O líder bloqueia todos as operações de escrita e espera até que a réplica esteja disponível novamente.

Replicação Síncrona *versus* Assíncrona

- Normalmente uma das réplicas é síncrona, as demais são assíncronas.
- Se a síncrona falha, uma das outras se torna síncrona.
- Assim se garante cópia atualizada dos dados em pelo menos dois nós: no líder e na réplica síncrona.

Replicação Síncrona *versus* Assíncrona

- Mas é comum a replicação ser configurada como completamente assíncrona quando se tem muitas réplicas ou as mesmas estão geograficamente distribuídas.
- Neste caso se o líder falha, todas as operações de escrita não enviadas às réplicas são perdidas.
- A vantagem é que o líder continua processando operações de escrita, mesmo que todas as réplicas falhem.

Configurar novas réplicas

- Fazer um *snapshot* da base de dados do líder
- Copiar os dados para a nova réplica
- Nova réplica conecta-se ao líder e requisita todas as mudanças ocorridas desde que foi feito o *snapshot.*

Quando um seguidor falha

- Todas as réplicas possuem em seu disco local um log de alterações.
- Quando volta a operar, consulta qual foi a última transação ocorrida antes da falha.
- Conecta-se ao líder e requisita as últimas mudanças desde a falha.

Falha no líder (*failover)*

- Quando o líder falha, uma das réplicas é promovida a líder.
- Clientes e seguidores precisam ser reconfigurados para reconhecer o novo líder.
- Esse processo pode ser manual ou automático

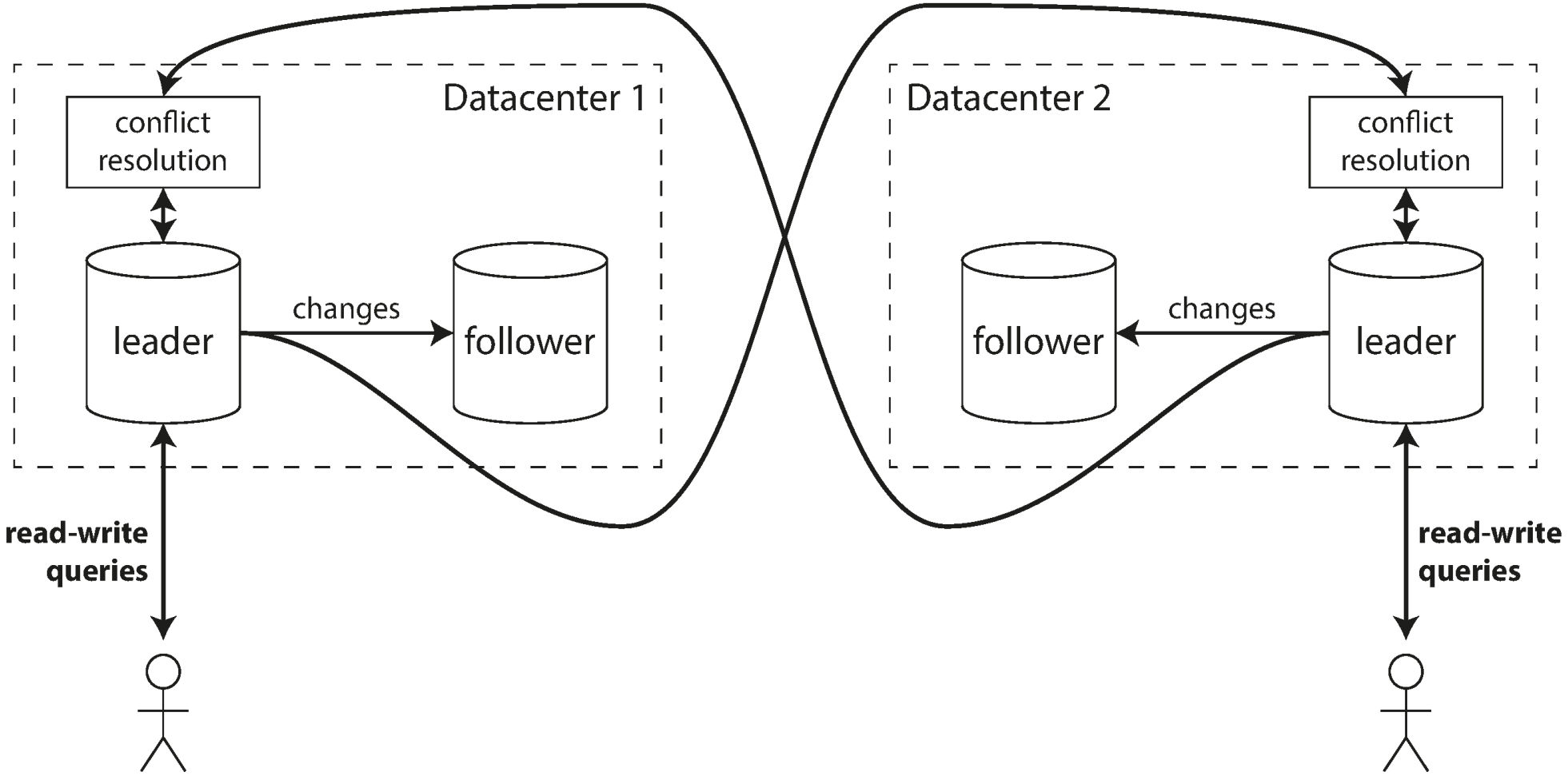
Failover automático

- Reconhecer a falha no líder:
- Clientes e seguidores precisam ser reconfigurados para reconhecer o novo líder.
- Esse processo pode ser manual ou automático

Replicação com vários líderes (*multi-leader)*

- Os clientes enviam as requisições de escrita a vários nós líderes
- Qualquer um deles pode aceitar
- Os líderes enviam alterações para os demais líderes e para os nós seguidores.

Replicação com vários líderes (*multi-leader)*



Exemplo de replicação multi-leader entre múltiplos datacenters.

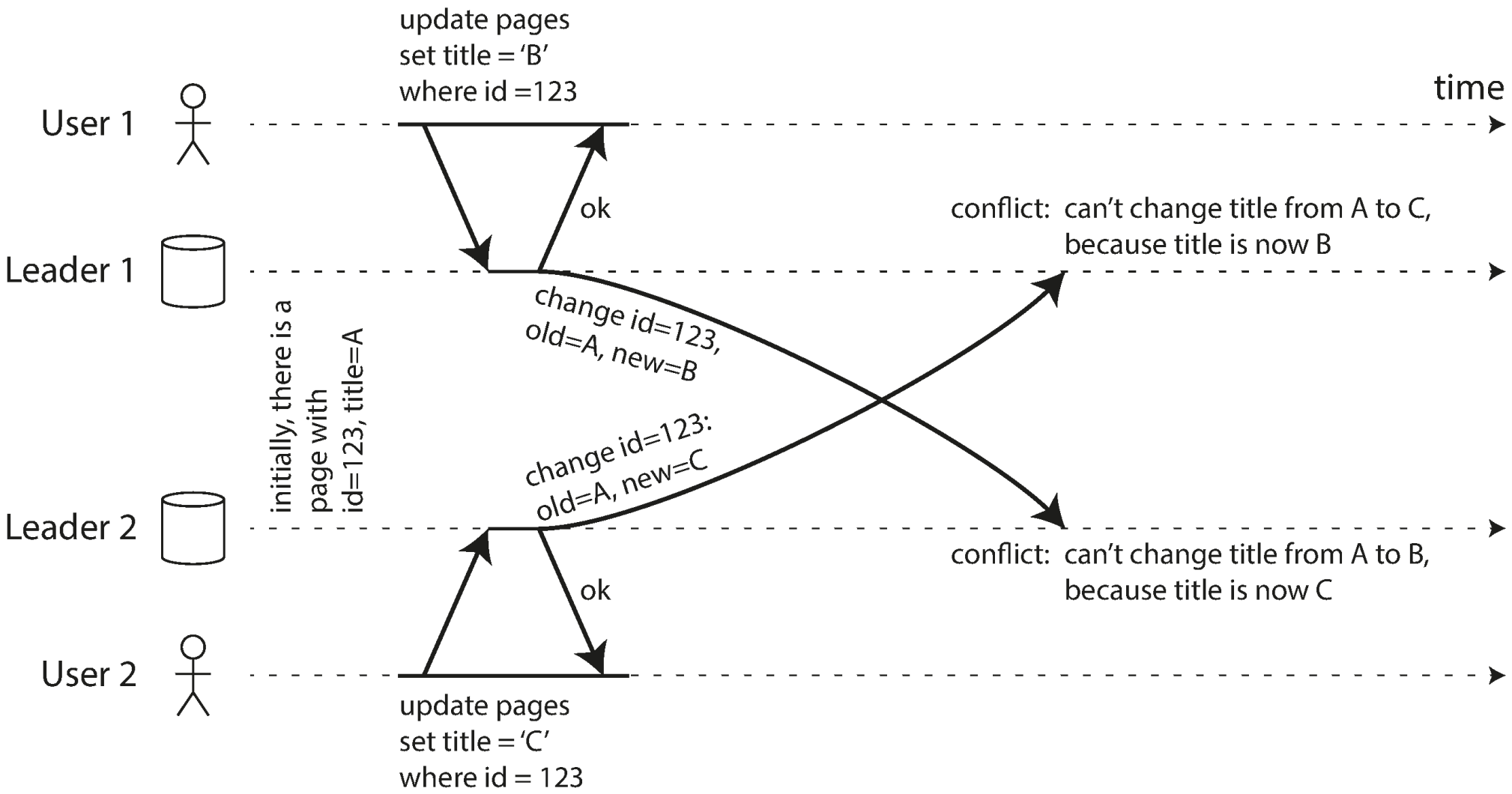
Replicação *com vários líderes - uso*

- Operações entre vários *datacenter*
- Clientes com operação *offline* (ex. agenda)
- Edição colaborativa (ex. google docs, etherpad)

Replicação *com vários líderes - desvantagens*

- O mesmo dado pode ser modificado concorrentemente por dois *datacenter* diferentes
- Necessidade de mecanismos para resolução de conflitos

Resolução de conflitos



Exemplo de conflito: dois líderes atualizam concorrentemente o mesmo registro

Resolução de conflitos

- Evitar o conflito: garantir que as requisições de um determinado usuário sejam enviadas a um mesmo datacenter/líder.
- Cada escrita recebe um ID único (ex.: timestamp, hash, chave-valor, etc.) e cada réplica também recebe um ID único. Escritas vindas da réplica com maior ID têm preferência

Resolução de conflitos

- Ordenar de alguma maneira e então concatenar. No exemplo da figura ficaria B/C.
- Gravar o conflito em alguma estrutura de dados e perguntar ao usuário qual a ordem correta

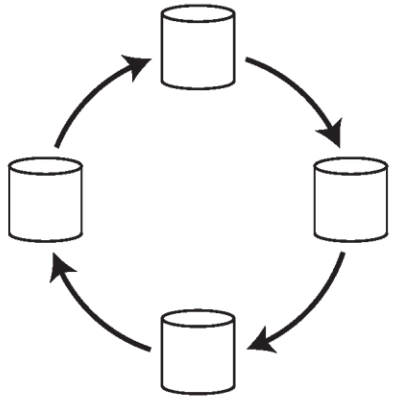
Resolução de conflitos

- Ordenar de alguma maneira e então concatenar. No exemplo da figura ficaria B/C.
- Gravar o conflito em alguma estrutura de dados e perguntar ao usuário qual a ordem correta

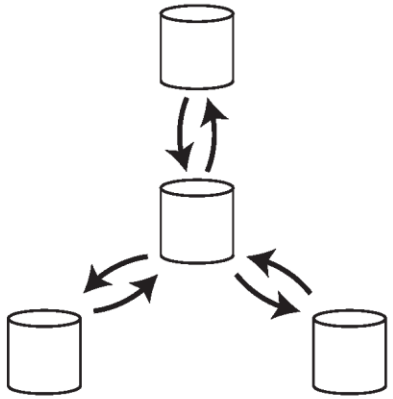
Resolução de conflitos

- Como a resolução de conflitos depende da aplicação, muitas soluções multi-leader permitem resolução de conflitos via código:
  - Na escrita: sem interação com o usuário. Processo roda em background pra resolver os conflitos.
  - Na leitura: os conflitos são identificados e armazenados. Na próxima leitura, a aplicação pode interagir com usuário ou resolver automaticamente o conflito.

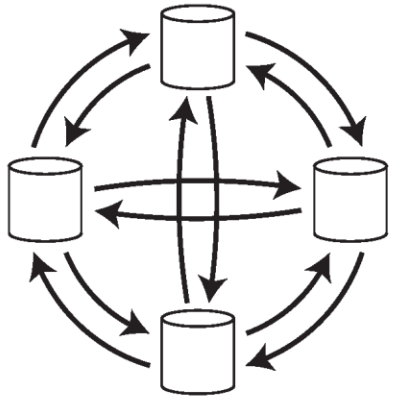
Topologias de replicação multi-leader



(a) Circular topology



(b) Star topology



(c) All-to-all topology

Topologias de replicação multi-leader

- A mais comum é a topologia todos-para-todos.
- MySQL usa topologia circular, onde a operação de escrita vai passando de um nó para outro até atingir todos os nós.
- Na topologia estrela um nó raiz repassa as operações de escrita aos demais nós.

Topologias de replicação multi-leader

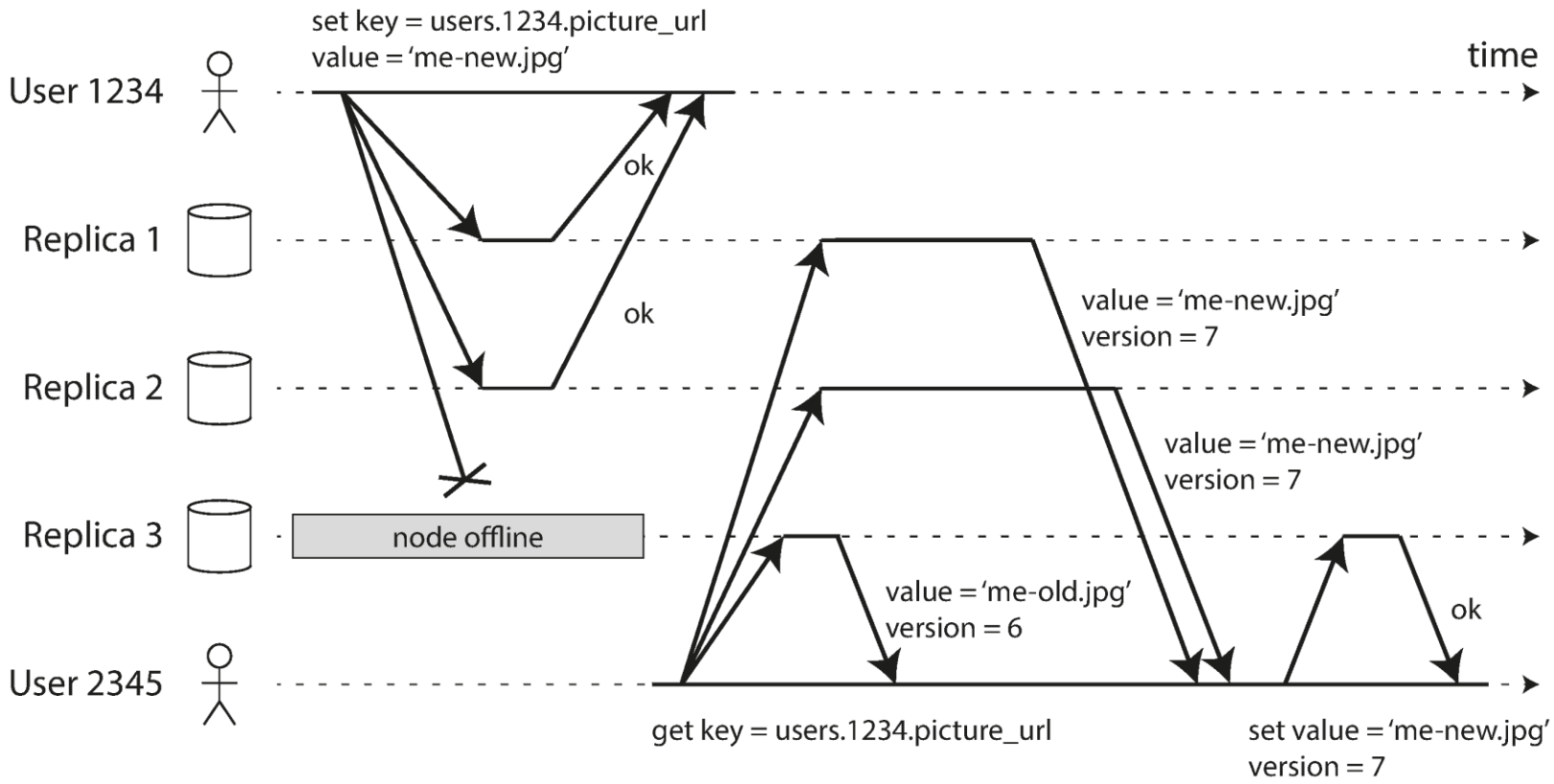
- A mais comum é a topologia todos-para-todos.
- MySQL usa topologia circular, onde a operação de escrita vai passando de um nó para outro até atingir todos os nós.
- Na topologia estrela um nó raiz repassa as operações de escrita aos demais nós.

Replicação *leaderless*

- Os clientes enviam cada requisição de escrita para vários nós e lêem de vários nós em paralelo de forma a detectar e corrigir nós com dados obsoletos.
- Em alguns casos, um nó coordenador envia as requisições de escrita, mas não força uma ordem de escrita.
- Conhecida como Dynamo Style é utilizada no Riak, Cassanda e Voldermort.

Replicação *leaderless*

- Não existe *failover*



Exemplo de quorum write, quorum read e read repair depois de uma interrupção

Replicação *leaderless*

- Não existe *failover*
- A requisição é enviada para todas as 3 réplicas
- Mesmo que uma falhe e não realize a escrita, se duas responderem OK, a que falhou é ignorada.

Read repair e anti-entropia

- Depois de uma falha, quando volta ao aor a réplica fica com dados obsoletos. Para corrigir:
  - Read repair: O cliente percebe que o dado é obsoleto e corrige naquela réplica.
  - Anti-entropia: um processo em *background* procura dados obsoletos e faltantes nas réplicas

Quórum para leitura e escrita

- Como saber se a quantidade de réplicas que respondeu OK é suficiente?
- No exemplo n=3, w=2, r=2. Como w + r > n. w e r são o número mínimo de respostas necessárias.

Quórum para leitura e escrita

- Número tolerado de falhas:
  - Se w < n = tolera-se um nó indisponível
  - Se r < n = tolera-se um nó indisponível
  - Se n=3, w=2, r=2, tolera-se um nó indisponível
  - Se n=5, w=3, r=3, tolera-se dois nós indisponíveis

Exercícios

1) Descreva a diferença entre replicação síncrona e assíncrona, quando utilizar e as vantagens e desvantagens de cada uma.

Exercícios

2) Analise a situação apresentada na figura abaixo e descreva que tipo de abordagem pode ser adotada pra resolver o conflito