ACH 2147 — DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS

CONSISTÊNCIA E REPLICAÇÃO

Daniel Cordeiro

24 de junho de 2019

Escola de Artes, Ciências e Humanidades | EACH | USP

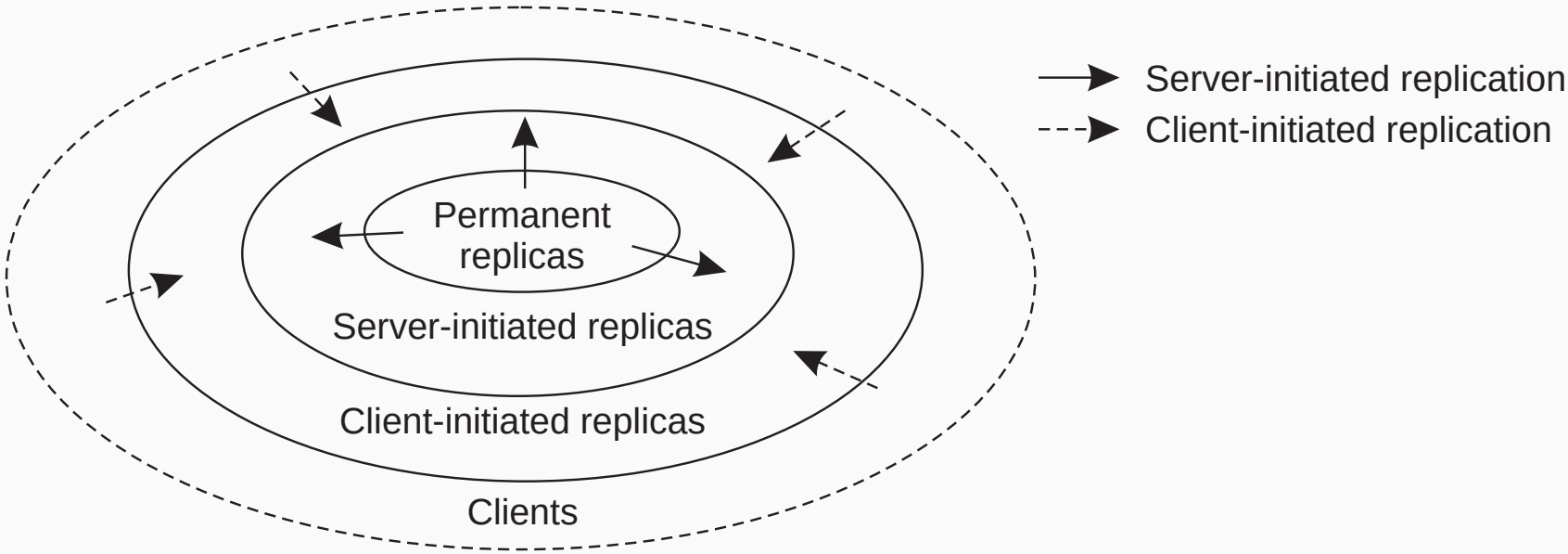
REPLICAÇÃO DE CONTEÚDO

Distingue diferentes processos

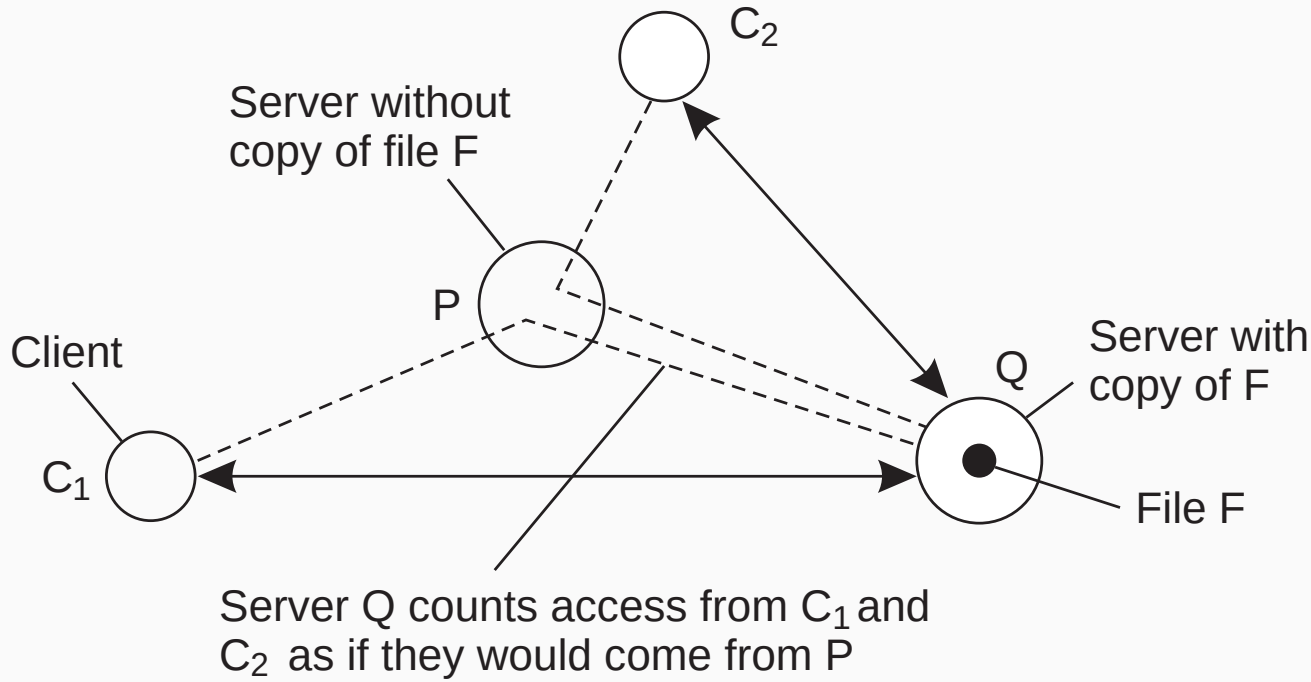
Um processo é capaz de hospedar uma réplica de um objeto ou dado:

- **réplicas permanentes:** processo/máquina sempre tem uma réplica
- **réplica iniciada pelo servidor:** processos que podem hospedar uma réplica dinamicamente, sob demanda de um outro servidor ou *data store*
- **réplica iniciada pelo cliente:** processos que podem hospedar uma réplica dinamicamente, sob demanda de um cliente (cache do cliente)

REPLICAÇÃO DE CONTEÚDO



RÉPLICAS INICIADAS PELO SERVIDOR



- mantenha o número de acessos aos arquivos, agregando-os pelo servidor mais próximo aos clientes que requisitarem o arquivo
- número de acessos cai abaixo de um threshold $D \Rightarrow$ descartar arquivo
- número de acessos acima de um threshold $R \Rightarrow$ replicar arquivo
- número de acessos entre D e $R \Rightarrow$ migrar arquivo

Modelo
Considere apenas uma combinação cliente–servidor:

- propaga apenas a **notificação/invalidação** de uma atualização (normalmente usada por caches)
- transfere dados de uma cópia para outra (bancos de dados distribuídos): replicação passiva
- propaga **operações** de atualização para outras cópias: replicação ativa

Nota
Nenhuma abordagem é melhor que outra, seu uso depende da largura de banda disponível e a razão leituras/escritas nas réplicas

pushing *iniciada pelo servidor*; uma atualização é propagada mesmo que o alvo não tenha pedido por ela

pulling *iniciada pelo cliente*; uma atualização solicitada pelo cliente

Observação
Podemos trocar dinamicamente entre os métodos *pulling* e *pushing* com o uso de **leases**: um contrato no qual o servidor promete enviar (*push*) atualizações para o cliente até que o *lease* expire.

Problema
Fazer com que a data de expiração do *lease* dependa do comportamento do sistema (*leases* adaptativos):

Problema
Fazer com que a data de expiração do *lease* dependa do comportamento do sistema (*leases* adaptativos):

- **leases com idade**: um objeto que não for modificado nos últimos tempos não será modificado em um futuro próximo, então conceda um *lease* que dure bastante

Problema
Fazer com que a data de expiração do *lease* dependa do comportamento do sistema (*leases* adaptativos):

- **lease baseado na frequência de renovação**: quanto maior a frequência com que o cliente requisitar o objeto, maior a data de expiração para aquele cliente (para aquele objeto)

Problema
Fazer com que a data de expiração do *lease* dependa do comportamento do sistema (*leases* adaptativos):

- **lease baseado no estado**: quando mais sobrecarregado o servidor estiver, menor a data da expiração se torna

Problema
Fazer com que a data de expiração do *lease* dependa do comportamento do sistema (*leases* adaptativos):

- **leases com idade**: um objeto que não for modificado nos últimos tempos não será modificado em um futuro próximo, então conceda um *lease* que dure bastante
- **lease baseado na frequência de renovação**: quanto maior a frequência com que o cliente requisitar o objeto, maior a data de expiração para aquele cliente (para aquele objeto)
- **lease baseado no estado**: quando mais sobrecarregado o servidor estiver, menor a data da expiração se torna

Por que fazer tudo isso? Para tentar reduzir ao máximo o estado do servidor, mas ainda assim prover consistência forte.

Protocolos de consistência

PROTOCOLOS DE CONSISTÊNCIA

Descrevem a implementação de um modelo de consistência específico.

- consistência contínua
- protocolos *primary-based*
- protocolos de replicação de escrita

Operação

- cada servidor S_i tem um log, denotado por $log(S_i)$
- considere o item x e seja $weight(W)$ o mudança numérica em seu valor depois de uma operação de escrita W. Assuma que

$$\forall W : weight(W) > 0$$

- W é inicialmente enviado para uma das N réplicas, denotadas por $origin(W)$. $TW[i, j]$ são as *escritas executadas pelo servidor* S_i *que originaram de* S_j:

$$TW[i, j] = \sum \{weight(W) | origin(W) = S_j \ \& \ W \in log(S_i)\}$$

Nota
Valores atuais $v(t)$ de x:

$$v(t) = v_{init} + \sum_{k=1}^{N} TW[k, k]$$

valor v_i de x na réplica i:

$$v_i = v_{init} + \sum_{k=1}^{N} TW[i, k]$$

CONSISTÊNCIA CONTÍNUA: ERROS NUMÉRICOS

Problema
Precisamos garantir que $v(t) - v_i < \delta_i$ para todo servidor S_i

Problema
Precisamos garantir que $v(t) - v_i < \delta_i$ para todo servidor S_i

Abordagem
Faça cada servidor S_k manter uma **visão** $TW_k[i, j]$ do que ele acredita ser o valor de $TW[i, j]$. Essa informação pode ser enviada por **gossip** quando uma atualização for propagada.

Problema
Precisamos garantir que $v(t) - v_i < \delta_i$ para todo servidor S_i

Abordagem
Faça cada servidor S_k manter uma **visão** $TW_k[i,j]$ do que ele acredita ser o valor de $TW[i,j]$. Essa informação pode ser enviada por **gossip** quando uma atualização for propagada.

Nota:

$$0 \leq TW_k[i,j] \leq TW[i,j] \leq TW[j,j]$$

Solução

S_k envia as operações de seu log para S_i *quando perceber que* $TW_k[i,k]$ *está ficando muito longe de* $TW[k,k]$; em particular quando

$$TW[k,k] - TW_k[i,k] > \delta_i/(N-1)$$

Solução

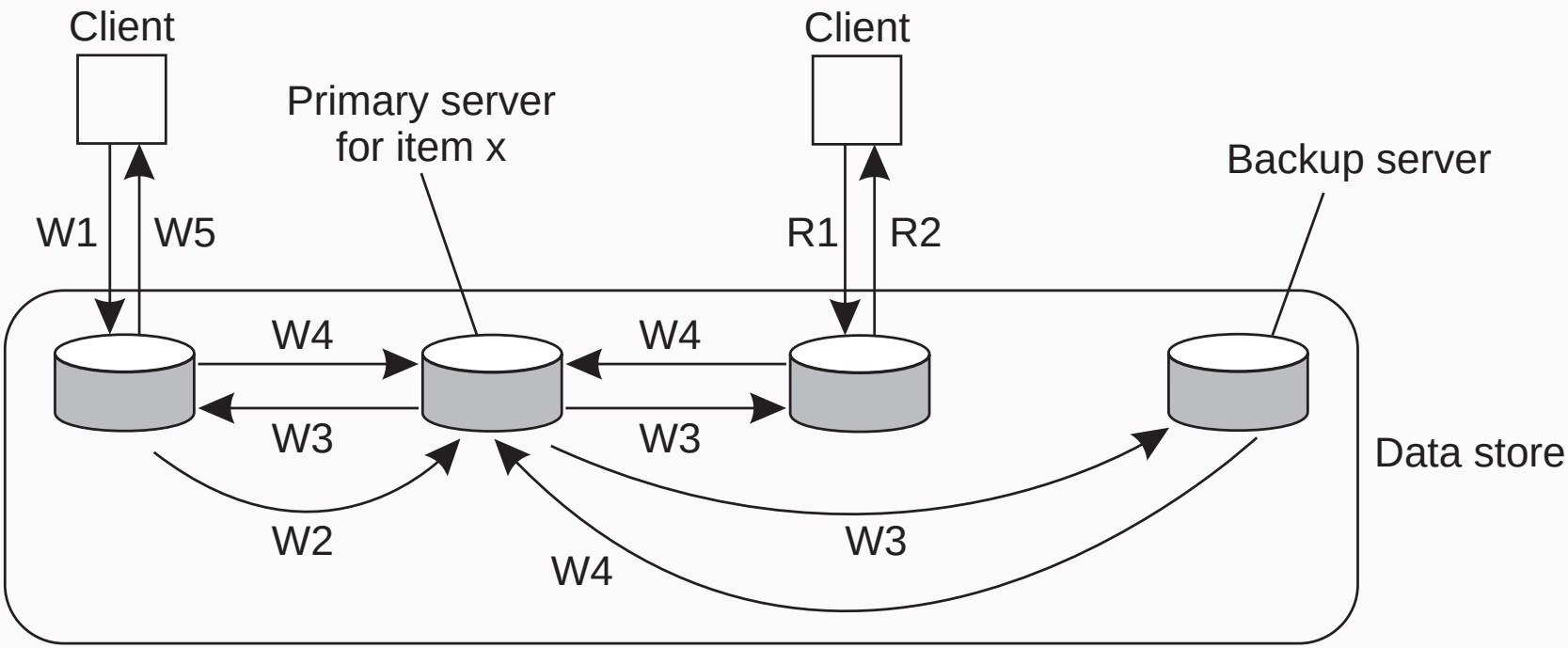
S_k envia as operações de seu log para S_i *quando perceber que* $TW_k[i,k]$ *está ficando muito longe de* $TW[k,k]$; em particular quando

$$TW[k,k] - TW_k[i,k] > \delta_i/(N-1)$$

Nota

A defasagem (*staleness*) pode ser lidada de forma análoga, mantendo no log o que foi visto por último de S_i.

Remote-write backup



W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

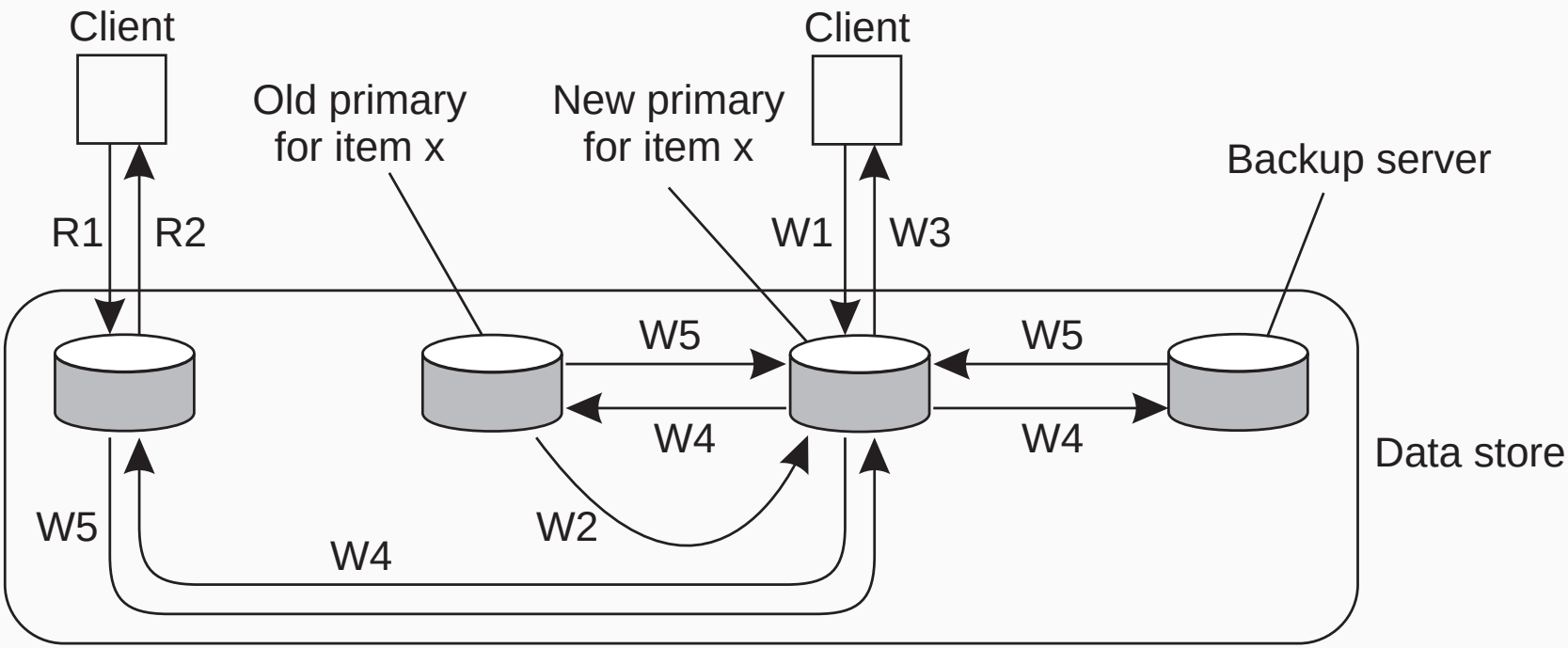
R1. Read request
R2. Response to read

Exemplo de backup com um primary protocol

É tradicionalmente aplicado em bancos de dados distribuídos e sistemas de arquivos que requerem um alto grau de tolerância a falhas. As réplicas são colocadas, em geral, numa mesma LAN.

Garante consistência sequencial

Primary-based protocol com escritas locais



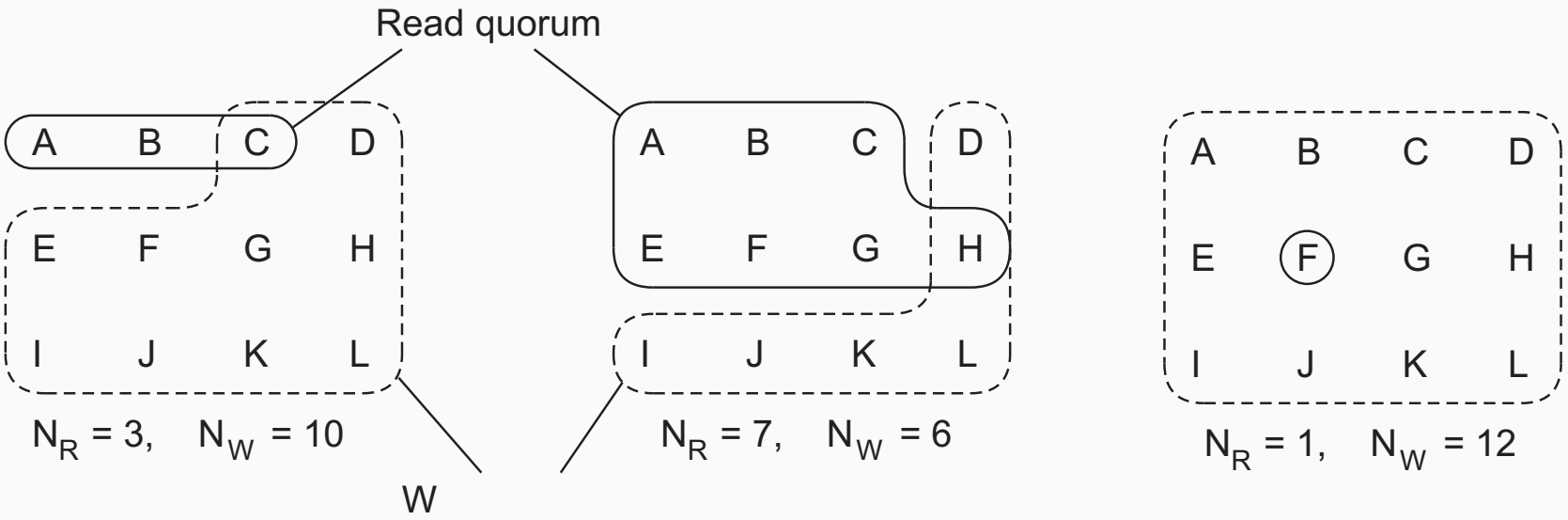
W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

Exemplo de um primary protocol para backup com escritas locais
Computação móvel em modo desconectado (envia todos os arquivos relevantes para o usuário antes do usuário se desconectar e atualiza mais tarde).

Quorum-based protocols

Garante que toda operação é realizada quando existir uma maioria de votos: distingue o **quorum de leitura** do **quorum de escrita**:



necessários: $N_R + N_W > N$ e $N_W > N/2$

THE END?

- Capítulos que não cobrimos:
  - 8. Tolerância à Falhas
  - 9. Segurança
- Algorítimos distribuídos:
  - Paxos (brevemente discutido no Cap. 8)
  - Conjectura de Brewer (“Teorema CAP”) (CAP = *Consistency, Availability, Partitioning tolerance*)
  - BSP/CGM (Bulk Synchronous Parallel / Coarse Grained Multicomputer)
  - Blockchain (e seus algorítimos de consenso)
  - ...
- Exascale Computing