

Computação Distribuída

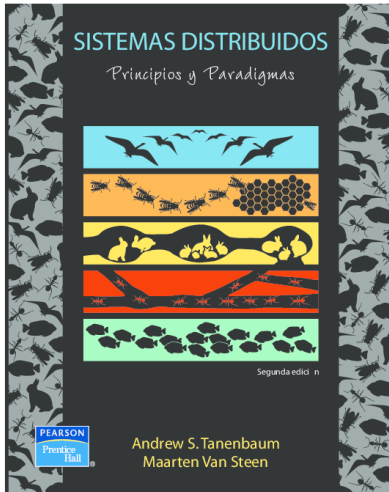
Replicação e Consistência

CHAPTER 7

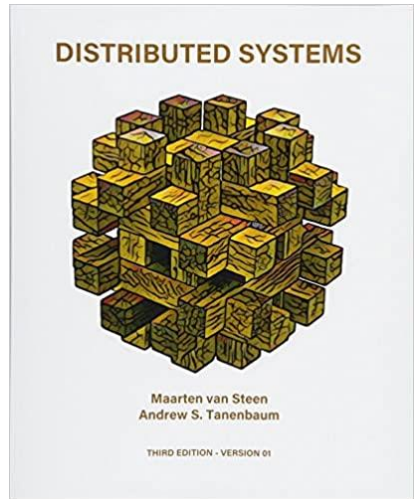
CONSISTENCY AND REPLICATION

Vladimir Rocha (Vladi)

CMCC - Universidade Federal do ABC



2007



2017

Disclaimer

- Estes slides foram baseados nos do professor **Emilio Franceschini** para o curso de Sistemas Distribuídos na UFABC.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- Estes slides foram adaptados daqueles originalmente preparados (e gentilmente cedidos) pelo professor **Daniel Cordeiro, da EACH-USP** que por sua vez foram baseados naqueles disponibilizados online pelos autores do livro "Distributed Systems", 3ª Edição em: <https://www.distributed-systems.net>.

Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
- Protocolos de consistência

Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
- Protocolos de consistência

Introdução

Razões para replicar

Aumentar a disponibilidade do sistema (caso alguma réplica morra ou haja alguma corrupção nos dados)

Aumentar o desempenho e a escalabilidade de tamanho e geográfica do sistema

Pergunta a resolver

Se a replicação ajuda tanto, qual é o problema?

Consistência !

Introdução

Problema principal

Para manter a consistência entre as réplicas, geralmente precisamos garantir que todas as operações **conflitantes** sejam realizadas na mesma ordem em todas as réplicas

Operações conflitantes

Terminologia da área de controle de transações:

read-write conflito onde uma operação de leitura e uma de escrita ocorrem de forma concorrente

write-write conflito com duas operações concorrentes de escrita

Problema

Garantir a ordem global de operações conflitantes pode ser muito custoso, diminuindo a escalabilidade.

Solução: diminuir os requisitos de consistência e, com sorte, conseguir evitar sincronizações globais custosas.

Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
- Protocolos de consistência

Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
 - Modelo do sistema
 - Linearizable [Herlihy 1991]
 - Sequential [Lamport 1979]
 - Causal [Hutto 1990]
 - Read your Writes [Terry 1994]
 - Eventual [Vogels 2009]

Cap8

Linearizable

Paxos/Raft

Sequential

Causal

RyW

Eventual

Dynamo

Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
 - Modelo do sistema
 - Sequencial
 - Causal
 - RyW
 - Eventual

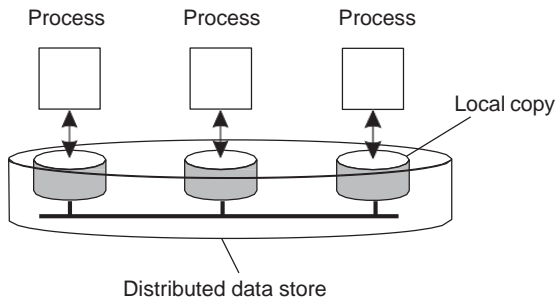
Consistência nos dados

Modelo do sistema

Um *data store* é uma coleção de dispositivos de armazenamento. Processos que enviam operações de leitura e escrita.

Interação entre um *data store* distribuído e os processos, no qual o *data store* define o resultado de operações concorrentes de leitura e escrita.

Como criamos a consistência do resultado?



Consistência nos dados

Notação

Eixo horizontal é o tempo que avança de esquerda à direita.
 P_i indica o processo i que está executando uma operação.
A operação de escrita no objeto x é $W(x)$ e de leitura é $R(x)$.

P1:	$R(x)a$
P2:	$W(x)a$

Veja $W(x)a$ como um *update* no saldo da conta x com o valor a .

Veja $R(x)a$ como um *select* do saldo da conta x e devolve o valor a

Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
 - Modelo do sistema
 - Sequencial
 - Causal
 - Eventual

Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
 - Modelo do sistema
 - **Linearizable**
 - Sequencial
 - Causal
 - RyW
 - Eventual

Consistência nos dados: linearizable

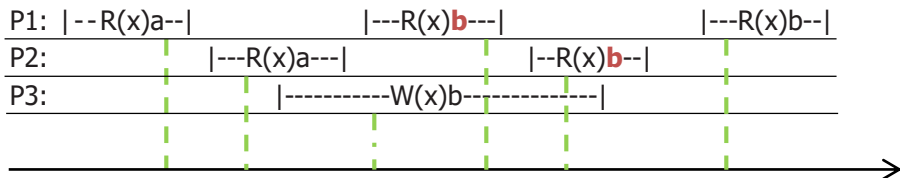
Definição

Informalmente: não é possível dizer que o dado está replicado.

A operação deve ser realizada “instantaneamente” em um determinado ponto do tempo (propriedade *real-time*)

- Cada operação deve ser vista com um início e fim |-----operação-----|

Note que o primeiro R(x)b do P1 influencia no R(x)b do P2, mesmo que o W(x)b do P3 pareça estar sendo executado.



Agenda

Consistência e Replicação



- Introdução
- Consistência centrada nos dados
 - Modelo do sistema
 - Contínua
 - Sequencial
 - Causal
 - RyW
 - Eventual

Consistência nos dados: sequencial

Intuição

Faz sentido obter NIL (depois de um Write) na imagem abaixo?
Note que não há início-fim nas operações como no linearizable.
Há um tempo de propagação das operações.

P1:	W(x)a	
P2:	R(x)NIL	R(x)a

P1:	 W(x)a
P2:	R(x)NIL  R(x)a

Consistência nos dados: sequencial

Detalhes

Criado por Lamport em 1979

Fácil de entender, difícil de implementar



FoundationDB

ACID + NoSQL

Desenvolvido pela Apple (2013 -)

Consistência nos dados: sequencial

Definição

O resultado de qualquer execução é o mesmo, como se as operações de todos os processos fossem executadas na mesma ordem sequencial e as operações de cada processo aparecem nessa sequência na ordem especificada pelo seu programa.

Em outras palavras:

Qualquer intercalação de operações (read e write) é aceitável desde que *todos os processos vejam a mesma intercalação e na ordem especificada pelo seu programa*

Consistência nos dados: sequencial

Em outras palavras:

Qualquer intercalação de operações (read e write) é aceitável desde que *todos os processos vejam a mesma intercalação e na ordem especificada pelo seu programa*

P1:	W(x)a		
P2:	W(x)b		
P3:		R(x)b	R(x)a
P4:		R(x)b	R(x)a

(a)

(a) Apresenta *data store* com consistência sequencial?

Lembre que com a sequencial é possível mover o W(x)a

Consistência nos dados: sequencial

Em outras palavras:

Qualquer intercalação de operações (read e write) é aceitável desde que *todos os processos vejam a mesma intercalação e na ordem especificada pelo seu programa*

P1:	W(x)a		
P2:	W(x)b		
P3:		R(x)b	R(x)a
P4:		R(x)b	R(x)a

(a)

P1:	W(x)a		
P2:	W(x)b		
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

(b)

(a) Apresenta *data store* com consistência sequencial?

(b) Apresenta consistência sequencial?

Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
 - Modelo do sistema
 - Contínua
 - Sequencial
 - Causal
 - RyW
 - Eventual

Consistência nos dados: causal

Detalhes

Criado por Hutto e Ahamad em 1990

Enfraquecimento da sequencial (usando o conceito de causalidade)



Document NoSQL (2009 -)



Graph NoSQL (2007 -)



Consistência nos dados: causal

Definição

Operações de write que potencialmente têm uma *relação de causalidade* devem ser vistas por todos os processos na mesma ordem.

Consistência nos dados: causal

Definição

Operações de write que potencialmente têm uma *relação de causalidade* devem ser vistas por todos os processos na mesma ordem.

Writes concorrentes podem ser vistos em uma ordem diferente por processos diferentes (desde que não tenham uma relação de causalidade).



Sequencial não tem isso !

Contrastando com a sequencial:

Qualquer intercalação de operações (read e write) é aceitável desde que *todos os processos vejam a mesma intercalação e na ordem especificada pelo seu programa*

Consistência nos dados: causal

Exemplo

Operações de write que potencialmente têm uma *relação de causalidade* devem ser vistas por todos os processos na mesma ordem.

P1:	W(x)a		
P2:	R(x)a	W(x)b	
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

(a)

(a) apresenta consistência causal?

Consistência nos dados: causal

Exemplo

Operações de write que potencialmente têm uma *relação de causalidade* devem ser vistas por todos os processos na mesma ordem.

P1:	W(x)a		
P2:	R(x)a	W(x)b	
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

(a)

P1:	W(x)a		
P2:		W(x)b	
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

(b)

(b) apresenta consistência causal? apresenta consistência sequencial?

Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
 - Modelo do sistema
 - Contínua
 - Sequencial
 - Causal
 - Read your Writes
 - Eventual

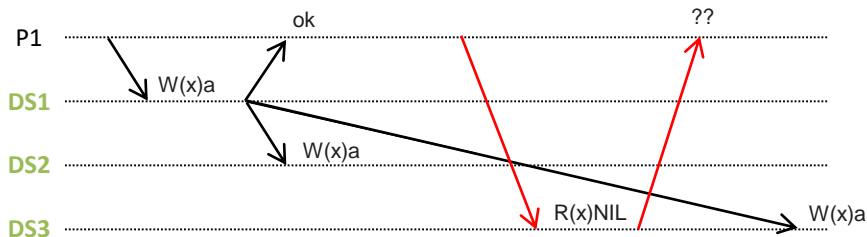
Centrada na sessão



Consistência na sessão: Read your Write

Definição

Uma operação de write realizada por um processo em um dado x sempre será vista por qualquer operação de read posterior realizada pelo mesmo processo.



Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
 - Modelo do sistema
 - Contínua
 - Sequencial
 - Causal
 - RyW
 - Eventual

Consistência nos dados: Eventual consistency

Detalhes

- Dependendo da aplicação é aceitável que as **atualizações não sejam propagadas imediatamente**
 - Normalmente os clientes acessam a mesma réplica, então não há problemas de inconsistência (na visão do cliente)
- Permite uma implementação de **modelo de consistência com menos restrições e portanto mais eficiente**
- Exemplos:
 - Facebook
 - DNS
 - Páginas Web em Geral



amazon
DynamoDB



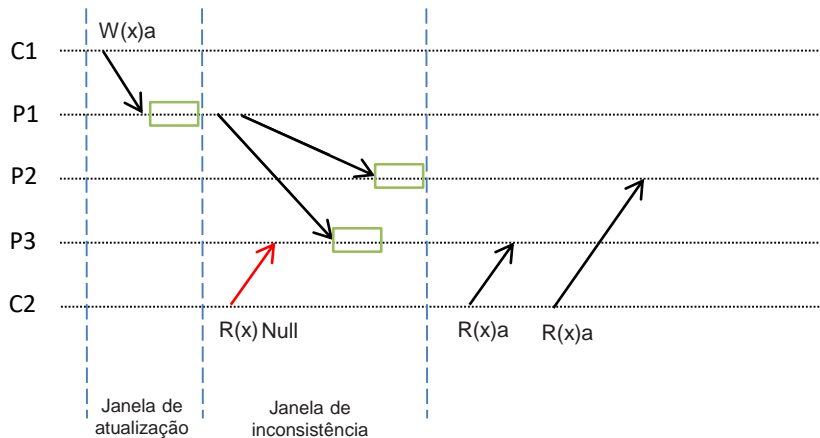
Consistência nos dados: Eventual consistency

Detalhes

- Relativamente fácil de implementar
- Em ausência de conflitos write-write, todas as réplicas convergirão a cópias idênticas (*eventualmente*)
- Assume-se que somente uma pequena quantidade de processos realiza operações de write.
- Quando acontece um conflito write-write, geralmente declara-se vencedor a última escrita, i.e., **last-write-wins**, sobrescrevendo as anteriores

Consistência nos dados: Eventual consistency

Exemplo: Cliente C2 obtém um valor antigo



Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
- Protocolos de consistência

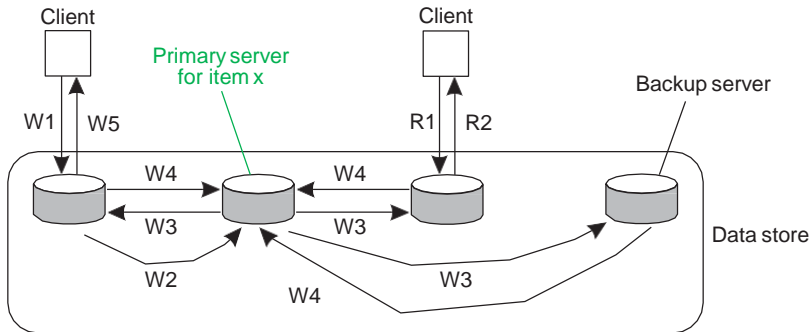
Protocolos de consistência

Descrevem a **implementação** de um modelo de consistência específico.

1. Protocolos **primary-based** (centrada nos dados: sequencial)
 - Remote Write Backup
 - Local Write
 - Chain Replication
2. Protocolos de replicação de escrita (centrada nos dados: sequencial)
 - Quorum (**leader-less**)

1. Protocolos primary-based

Remote-write backup [Budhijara 1993]



- 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

Note que existe um primary

1. Protocolos primary-based

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.

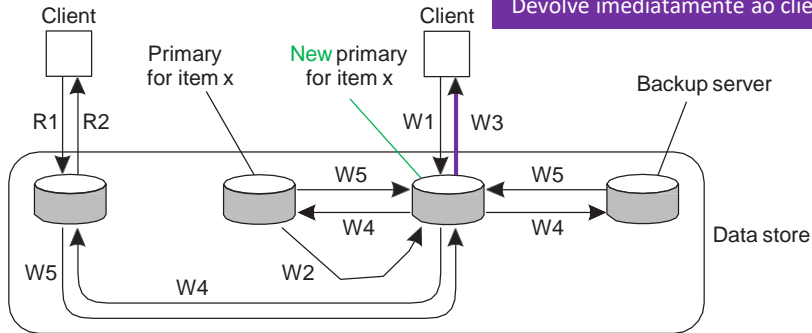
Problema?

Operações de escrita são lentas, pois todas as réplicas devem concordar com a escrita antes de devolver o resultado ao cliente.

Como resolver isso?

1. Protocolos primary-based

Local Write



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

Note a mudança do primary

1. Protocolos primary-based

Exemplo de um local write

Computação móvel em modo desconectado (envia todos os arquivos relevantes para o usuário antes do usuário se desconectar e atualiza mais tarde).

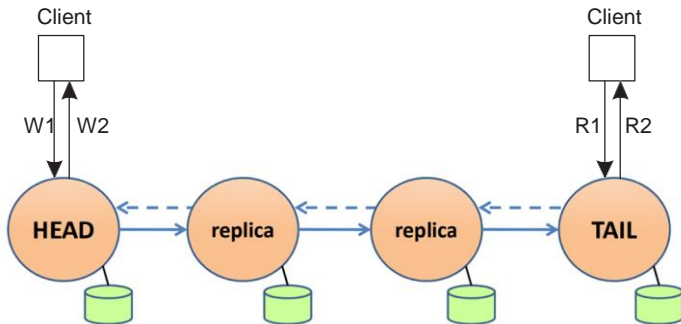
1. Protocolos primary-based

Como as leituras podem ir para qualquer backup server, é necessário considerar os seguintes problemas:

- Read your own write
- Monotonic read

1. Protocolos primary-based

Chain replication [2004] (não está no livro)



W1: write request

Linha azul: propagação do W1

Linha pontilhada: ACK

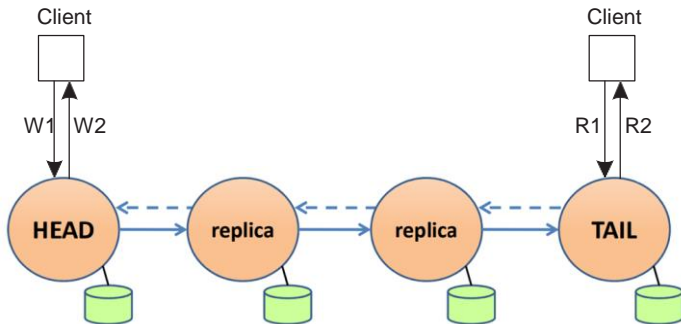
W2: write response (só após o HEAD receber o ACK)

R1: read request

R2: read response (imediatamente pelo TAIL)

1. Protocolos primary-based

Chain replication [2004] (não está no livro)



Caso1: que acontece com as mensagens **W** se morre o **HEAD**.

- O sucessor será o novo **HEAD**, sem perda de escritas (sendo executadas)

Caso2: que acontece com as mensagens **W** se morre o **TAIL**.

- O predecessor será o novo **TAIL**, sem perda de escritas (sendo executadas)

Caso3: que acontece com as mensagens **W** se morre uma réplica **X**.

- O predecessor precisará reenviar os últimos writes (que não foram propagados por **X**)

1. Protocolos primary-based

Remote-Write: **todas** as réplicas devem concordar com a escrita antes de devolver a resposta ao cliente

Local-write: **uma** réplica (a primary) executa a escrita e devolve a resposta ao cliente (depois dissemina a escrita).

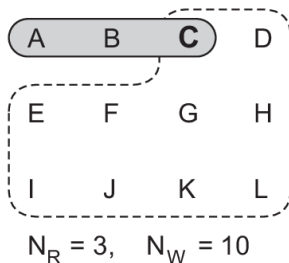
Existe algo intermediário?

Usando um **quórum**

2. Protocolos de replicação de escrita

Quorum-based protocols [Thomas, Gifford 1979] *leader-less*

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

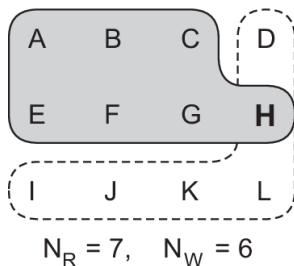


necessários: $N_R + N_W > N$; $N_W > N/2$

2. Protocolos de replicação de escrita

Quorum-based protocols

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

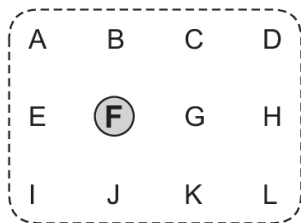


necessários: $N_R + N_W > N$; $N_W > N/2$

2. Protocolos de replicação de escrita

Quorum-based protocols

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



$$N_R = 1, \quad N_W = 12$$

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

2. Protocolos de replicação de escrita

Quorum-based protocols

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

- ROWA: $R=1$, $W=N$
(leitura rápida, escrita lenta)

ROWA não parece com o Primary-based Remote-write backup?
Podemos dizer que ROWA então é consistência sequencial?

2. Protocolos de replicação de escrita

Quorum-based protocols

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

- ROWA: $R=1$, $W=N$
(leitura rápida, escrita lenta)
- RAWO: $R=N$, $W=1$
(leitura lenta, escrita rápida)

2. Protocolos de replicação de escrita



Quorum-based protocols - Cassandra

Desenvolvido pelo Facebook em 2008.

NoSQL KV-store escalável

✓ Write Consistency Levels

Level	Description
ALL	A write must be written to the commit log and memtable on all replica nodes in the cluster for that partition.
QUORUM	A write must be written to the commit log and memtable on a quorum of replica nodes across <i>all</i> datacenters.
ONE	A write must be written to the commit log and memtable of at least one replica node.

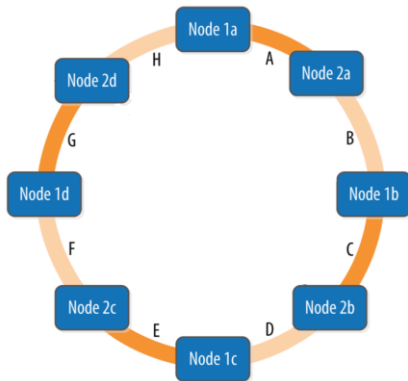
2. Protocolos de replicação de escrita



Quorum-based protocols - Cassandra

Utiliza o conceito de anel Chord para os servidores.

Utiliza o conceito de *consistent hashing* para chaves.



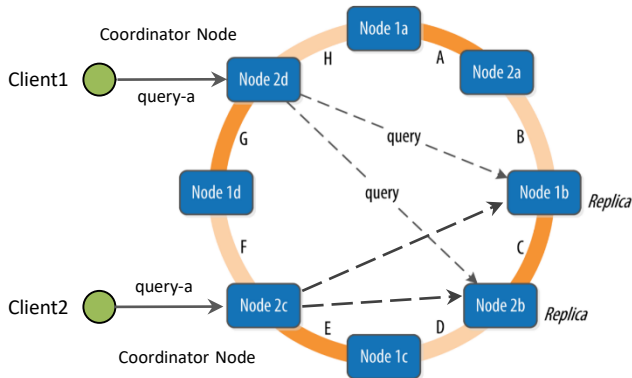
2. Protocolos de replicação de escrita



Quorum-based protocols - Cassandra

Utiliza o conceito de anel Chord para os servidores.

Utiliza o conceito de *consistent hashing* para chaves.



2. Protocolos de replicação de escrita

Como as escritas podem ir para qualquer servidor (já que não há um líder ou primary), é necessário considerar os seguintes problemas:

- Writes concorrentes
- Write com read concorrente

Conceitos adquiridos

- Operações conflitantes.
- Consistência: linearizável, sequencial, causal, read your writes, eventual.
- Protocolos primary-based: remote write, local write, chain.
- Protocolos leader-less: quórum.
- Cassandra.

Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
- Consistência centrada no cliente
- Protocolos de consistência
- Gerenciamento de réplicas
 - replicação de conteúdo e posicionamento
 - distribuição de conteúdo

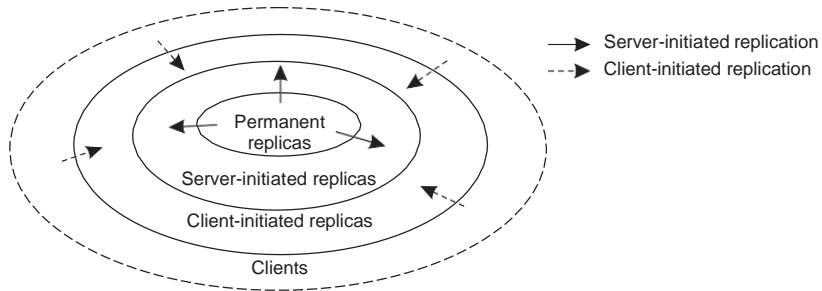
Gerenciamento: Replicação

Distingue diferentes processos

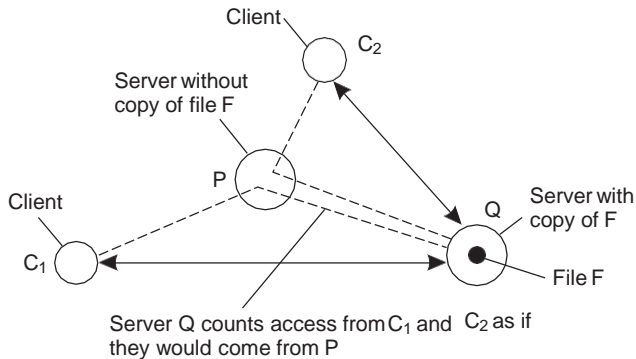
Um processo é capaz de hospedar uma réplica de um objeto ou dado:

- **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**)

Gerenciamento: Replicação



Gerenciamento: Replicação iniciada 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

Agenda

Consistência e Replicação

- Introdução
- Consistência centrada nos dados
- Consistência centrada no cliente
- Protocolos de consistência
- Gerenciamento de réplicas
 - replicação de conteúdo e posicionamento
 - distribuição de conteúdo

Gerenciamento: Distribuição de conteúdo

Modelo

Considere apenas uma combinação cliente-servidor ou servidor-servidor:

- servidor propaga apenas a **notificação/invalidação** de uma atualização para clientes (normalmente usada por caches)
- servidor transfere **dados** para outros servidores “réplica” (bancos de dados distribuídos): **replicação passiva**
- servidor propaga **operações** de atualização para outros servidores “réplica”: **replicação ativa**

Nota

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

Gerenciamento: Distribuição de conteúdo

Nos algoritmos epidêmicos foram usadas as técnicas de pull e push para disseminar informações (neste caso, para transferir dados ou propagar operações):

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.

Gerenciamento: Distribuição de conteúdo

Problema

Fazer com que a data de expiração do *lease* dependa do comportamento do sistema (*leases* adaptativos) e assim reduzir a carga no servidor:

- **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

Gerenciamento: Distribuição de conteúdo

Problema

Fazer com que a data de expiração do *lease* dependa do comportamento do sistema (*leases* adaptativos) e assim reduzir a carga no servidor:

- **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 (o cache dele devolverá o antigo)

Gerenciamento: Distribuição de conteúdo

Problema

Fazer com que a data de expiração do *lease* dependa do comportamento do sistema (*leases* adaptativos) e assim reduzir a carga no servidor:

- **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 (o cache dele devolverá o antigo)
- **lease baseado no estado:** quando mais sobrecarregado o servidor estiver, menor a data da expiração se torna (o servidor controlará menos clientes, pois os *leases* expiram mais rapidamente).

Gerenciamento: Distribuição de conteúdo



Content Delivery Network (CDN)

Provê a infraestrutura para distribuir e replicar conteúdo Web (vídeos, imagens, sites) através da internet

