Universidade Estadual de Mato Grosso dos Sul
Curso de Ciência da Computação
Disciplina de Algoritmos Paralelos e Distribuídos

Aula 8 – Consistência e Replicação

Por que usar Replicação?

- Confiabilidade
  - Garante comportamento correto, apesar de alguns tipos de erro.
  - Exemplo: Dados corrompidos.
  - O serviço continua enquanto pelo menos um servidor estiver funcionando.

- Desempenho
  - É importante quando um sistema distribuído precisa ser ampliado em quantidade e área geográfica.
  - No caso de ampliação em quantidade:
    - Tem-se diversos servidores WEB com o mesmo DNS. E os servidores são selecionados alternadamente.
  - No caso de ampliação geográfica:
    - As requisições são repassadas para o servidor mais próximo geograficamente.

Replicação

- Réplicas permitem que alguns servidores continuem a funcionar mesmo em ocasiões de falhas.
- É possível proteger contra corrupção de dados.
- Réplicas permitem que dados fiquem mais perto de onde serão acessados → diminuição da latência.

Replicação e Escalabilidade

- Replicação e cache são usados para melhorar o desempenho nos problemas de escalabilidade.
- A ideia é colocar cópias de dados próximas aos processos que as estão usando, reduzindo o tempo de acesso.

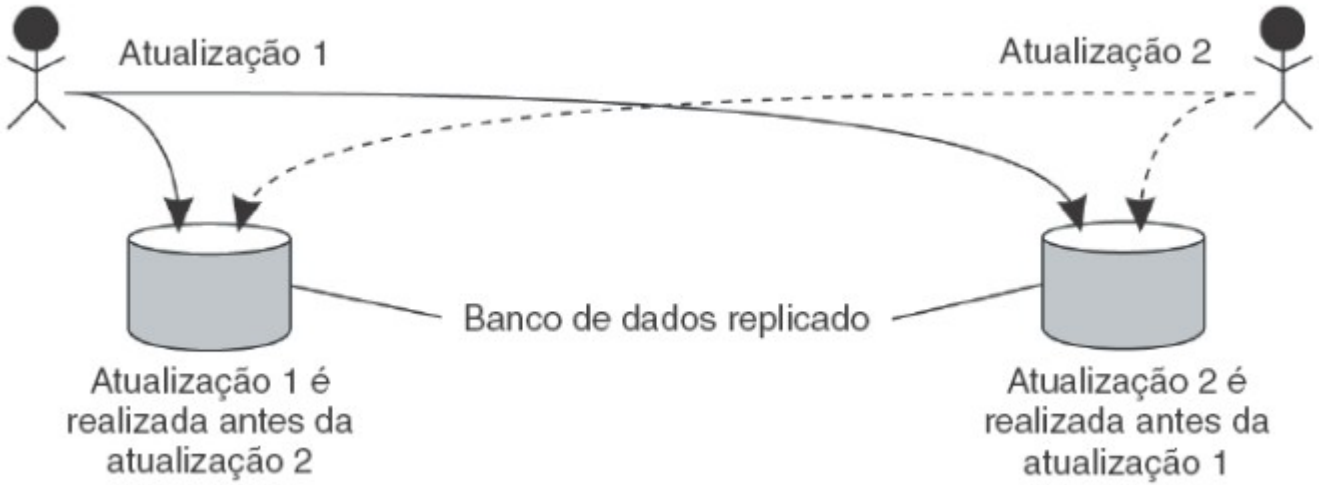
- Como agir em caso de consistência entre os dados e as diversas réplicas?

O Problema da Consistência

- No caso da utilização de cache em um servidor WEB:
  - Os navegadores costumam armazenar no local uma cópia de uma página WEB que já foi buscada anteriormente.
  - Se um usuário requisitar aquela página mais uma vez, o navegador automaticamente retorna a cópia local.
  - O tempo de acesso é excelente, porém:
    - E se o usuário quiser ter a versão mais recente da página?
    - E no caso de um banco de dados com informações bancárias?

O Problema da Consistência

- Atualização de banco de dados replicado que o deixa em estado inconsistente.



O Problema da Consistência

- A maior dificuldade está em como sincronizar todas as réplicas.
  - Todas as réplicas precisam chegar a um acordo sobre quando uma atualização deve ser realizada.
  - Geralmente, requer uma atualização global.
  - A única solução real é relaxar as restrições de consistência → evitar que atualizações sejam executadas como operações atômicas.

Modelos de Consistência

- O que é consistência?

- Quais são os diferentes modos de implementar modelos de consistência?

Modelos de Consistência Centrados em Dados

- Em operações de leitura e escrita em dados compartilhados, tem-se a disponibilidade por meio de:
  - Memória compartilhada (distribuída);
  - Banco de dados distribuído;
  - Sistema de arquivos;

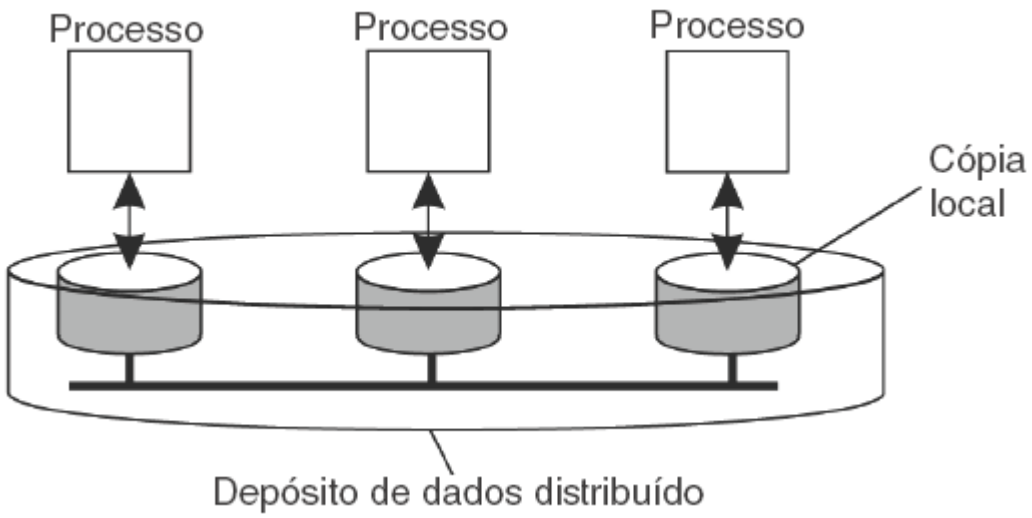
Depósito de Dados

- Um depósito de dados pode ser distribuído fisicamente por várias máquinas.
- Cada processo que pode acessar os dados do depósito tem uma cópia local (ou próxima) do depósito.
- Tem-se dois tipos de operações:
  - Escrita → altera os dados;
  - Leitura → não altera os dados;

- Operações de escrita são propagadas para outras cópias.

Depósito de Dados

- Organização geral de um depósito de dados lógico, fisicamente distribuído e replicado por vários processos.



Modelos de Consistência

- É definido como sendo o contrato entre processos e o depósito de dados.
  - Se os processos concordarem em obedecer certas regras, o depósito promete funcionar de maneira correta.
  - Exemplo: Um processo que executa um operação de leitura sobre um item de dados espera que a operação retorne um valor que mostre os resultados da última operação de escrita executada sobre aqueles dados.
- Na ausência de um relógio global, é difícil definir com precisão qual operação de escrita é a última.
  - Modelos de consistência restringem efetivamente os valores que uma operação de leitura sobre um item de dados pode retornar.
- Para obter soluções eficientes para o problema de consistência, deve-se 'relaxar' o conceito de consistência → o que se pode tolerar de inconsistência depende de cada aplicação.

Consistência Contínua

- Três caminhos distintos definem inconsistência:
  - Diferenças nos valores numéricos entre réplicas.
  - Diferença nas idades entre réplicas
  - Diferença em relação à ordenação de operações de atualização.

Desvios Numéricos

- Dados possuem semântica numérica
  - Exemplo: Registros que contém preços do mercado de ações.
  - Exemplo: Número de atualizações que foram aplicadas a uma determinada réplica, mas que ainda não foram vistas pelas outras.

Desvios de Idade

- Estão relacionados com a última vez em que uma réplica foi atualizada.
- Algumas aplicações podem tolerar que uma réplica forneça dados antigos, contanto que não sejam tão antigos.
  - Exemplo: Previsões de tempo, em geral, permanecem razoavelmente exatas durante algum tempo → servidor principal pode receber atualizações e tempos oportunos.

Desvios em Relação à Ordenação de Operações

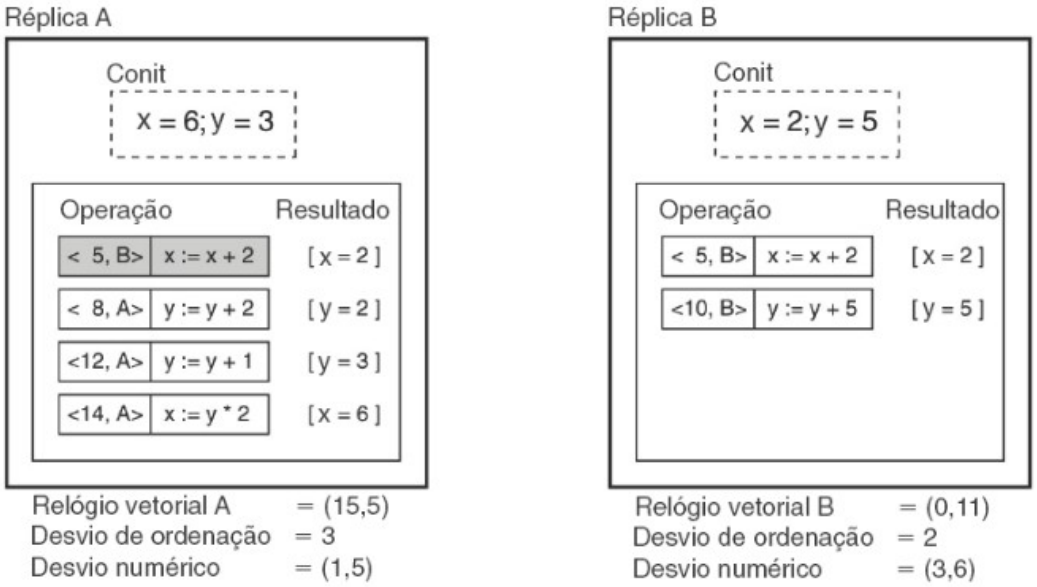
- Em algumas aplicações é permitido que a ordenação das atualizações seja diferente nas várias réplicas, dentro de um limite.
- Atualizações são aplicadas provisoriamente a uma cópia local, à espera de um acordo global de todas as réplicas.
- Em alguns casos, algumas atualizações podem precisar voltar atrás e serem aplicadas em uma ordem diferente antes de se tornarem permanentes.

Consistência Contínua

- Especifica a unidade segundo a qual a consistência deve ser medida:
  - Registro que representa uma única ação.
  - Boletim individual de previsão do tempo.

Consistência Contínua

- Exemplo de monitoração de desvios de consistência.
  - Réplica A recebeu a operação 5, B: x ← x+2 e a tornou permanente. A tem três operações de atualização provisórias: 8,A; 12,A; 14,A.
  - Desvio de ordenação = 3; vetor lógico = (15,5);
  - Desvio numérico = 1 (# atualizações não vistas); y = 5 para a atualização feita em B no valor de y e não vista por A;



Consistência Contínua

- Dois pontos de destaque:
  1. Granularidade:
     1. Grossa: Conit com grande quantidade de dados → réplicas entram em estado inconsistente mais rapidamente.
     2. Fina: Grande número de conits a serem gerenciados.

  2. Como colocar em prática o uso de conits
     1. Protocolos para impor inconsistência
     2. Especificação por parte dos programadores → bibliotecas.

Consistência Sequencial e Causal

- Modelos de consistência sequencial e causal ampliam os modelos de consistência contínua.
  - No entanto, quando for preciso efetivar atualizações provisórias em réplicas, estas terão que chegar a um acordo sobre uma ordenação global dessas atualizações.
  - Concordar com uma ordenação consistente das atualizações.
  - Assim, os modelos de consistência sequencial e causal visam chegar a tais ordenações consistentes.

Consistência Sequencial e Causal

- As operações de um processo são representadas ao longo de um eixo de tempo.
- O eixo de tempo é sempre representado na horizontal – o tempo avança da esquerda para a direita.
- $W_i(x)a$ → Escrita pelo processo P_i para o item de dados x com o valor a.
- $R_i(x)a$ → Leitura pelo processo P_i do item de dados x retornando o valor b.

Consistência Sequencial e Causal

- Exemplo: (1) P_1 executa uma escrita para o item x, modificando o seu valor para a. Esta operação é feita localmente e depois propagada para os outros processos. (2)Adiante P_2 lê o valor *NILL* e, pouco tempo depois, lê a. Existe um retardo para propagar a atualização de x para p.
- Exemplo2: Comportamento de dois processos que operam sobre o mesmo item de dados. O eixo horizontal representa o tempo.

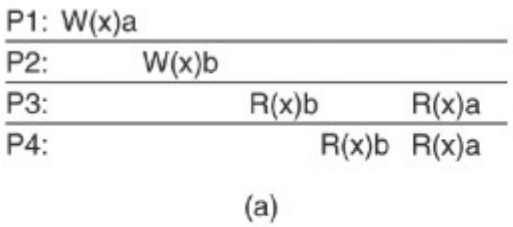
Consistência Sequencial

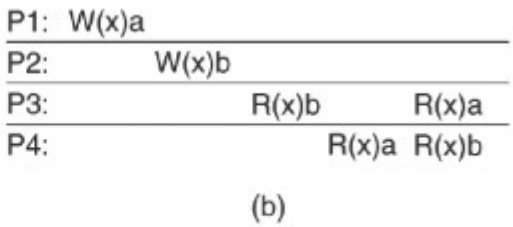
- Modelo de consistência centrado em dados definido por Lamport (1979) no contexto de memória compartilhada para sistemas multiprocessdores.

- Quando dois processos executam concorrentemente em máquinas diferentes, qualquer intercalação válida de operações de leitura e escrita é um comportamento aceitável, mas todos os processos vêem a mesma intercalação de operações.

Consistência Sequencial

- Tempo não é considerado:
  - Não há nenhuma referência à operação de escrita ‘mais recente’ sobre um item de dados.
  - Exemplo:
    - P_1 executa W(x)a para x. Mais tarde (tempo absoluto), o processo P_2 também executa uma operação de escrita, ajustando o valor de x para b. Os processos P_3 e P_4 primeiro lêem o valor b e, mais tarde, o valor a. A operação de escrita do processo P_2 parece ter ocorrido antes da de P_1.

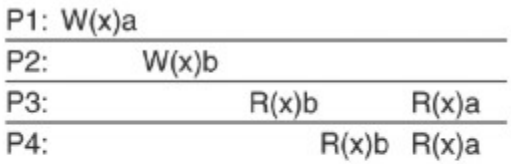




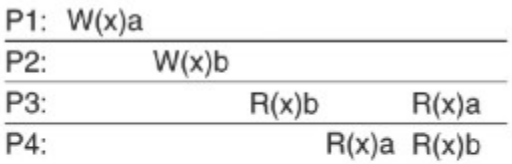
(a) Depósito de dados seqüencialmente consistente. (b) Depósito de dados que não é seqüencialmente consistente.

Consistência Sequencial

- Neste exemplo, a consistência sequencial é violada porque nem todos os processos vêem a mesma intercalação de operações de escrita. Para o processo P_3 parece que o item de dados foi primeiro alterado para b, e mais tarde para a. Por outro lado, P_4 concluirá que o valor final é b.



(a)



(b)

(a) Depósito de dados seqüencialmente consistente. (b) Depósito de dados que não é seqüencialmente consistente.

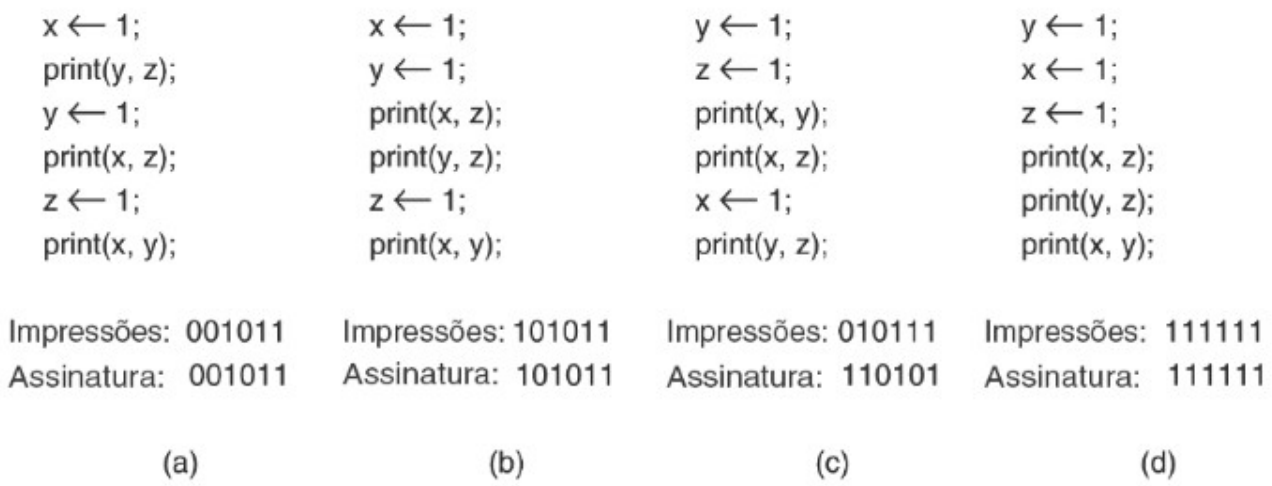
Consistência Sequencial

- Tem-se três processos que estejam executando concorrentemente P_1, P_2, P_3. Os itens de dados são x,y,z. Cada variável é inicializada com zero (0). Uma atribuição corresponde a uma escrita, enquanto um *print* corresponde a uma operação de leitura de dois argumentos.
- Várias sequências são possíveis, mas somente um subconjunto é válido (90 sequências).
- Exemplo de 3 processos que executam concorrentemente.

Processo P1	Processo P2	Processo P3
x ← 1;	y ← 1;	z ← 1;
print(y, z);	print(x, z);	print(x, y);

Consistência Sequencial

- Neste exemplo, em (a) P_1, P_2 e P_3 são executados em ordem. Já (b), (c) e (d) demonstram intercalações diferentes, mas igualmente válidas.
- O contrato entre processos e o depósito de dados é que os processos devem aceitar TODOS os resultados válidos como respostas adequadas e devem trabalhar corretamente se qualquer um deles ocorrer. Um programa que funciona somente com um subconjunto de resultados está incorreto.
- Exemplo: Quatro sequências de execução válidas para os processos. O eixo vertical é o tempo.



Consistência Causal

- O modelo de Consistência Causal faz uma distinção entre eventos que são potencialmente relacionados por causalidade e os que não são.
- Se o evento b é causado ou influenciado por um evento anterior a, a causalidade requer que todos vejam primeiro a e, depois b.
- Suponha que um processo P_1 escreva um item de dados **x**. Então P_2 lê **x** e escreve **y**. A leitura de **x** e a escrita de **y** são potencialmente relacionadas por causalidade, porque o cálculo de **y** pode ter dependido do valor de **x** lido por P_2, isto é, o valor escrito por P_1.

Consistência Causal

- Se dois processos executam uma escrita espontânea e simultânea de dois itens de dados diferentes, estes não estão relacionados por causalidade.
- Operações que não estão relacionadas por causalidade são denominadas concorrentes.
- Escritas que são potencialmente relacionadas por causalidade devem ser vistas por todos os processos na mesma ordem. Escritas concorrentes podem ser vistas em ordens diferentes em máquinas diferentes.

Consistência Causal

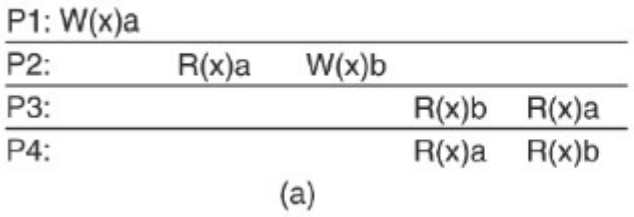
- Exemplo: A sequência é permitida quando o depósito é consistente por causalidade, mas não quando o depósito é sequencialmente consistente.

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

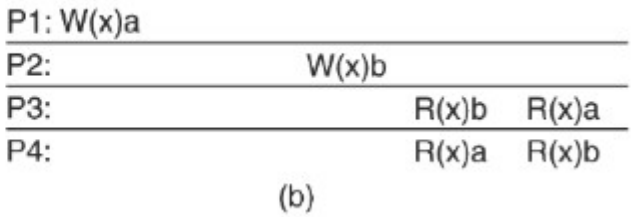
- Neste exemplo, temos uma sequência de eventos permitidos quando o depósito é consistente por causalidade, mas proibida quando o depósito é sequencialmente consistente. As escritas W2(x)b e W1(x)c são concorrentes, não sendo exigido que todos os processos as vejam na mesma ordem.

Consistência Causal

- Exemplo: (a) violação de um depósito consistente por causalidade; (b) sequência correta de eventos em um depósito consistente por causalidade.



(a)

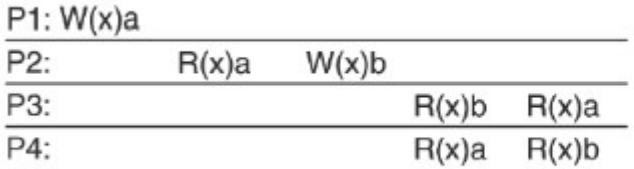


(b)

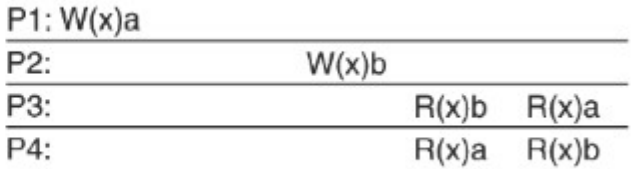
- Em (a) tem-se $W_2(x)b$ potencialmente dependente de $W_1(x)a$, porque **b** pode ser resultado de um cálculo que envolva o valor lido por $R_2(x)a$. As duas escritas são relacionadas por causalidade, portanto temos uma violação na ordenação das operações.

Consistência Causal

- Exemplo: (a) violação de um depósito consistente por causalidade; (b) sequência correta de eventos em um depósito consistente por causalidade.



(a)



(b)

- Em (b), como a leitura $R_2(x)a$ foi removida, $W_2(x)b$ e $W_1(x)a$ agora são escritas concorrentes. Um depósito consistente por causalidade não requer que escritas concorrentes sejam ordenadas globalmente.

Consistência Causal

- Requer monitorar quais processos se tornam quais escritas.
- É preciso construir e manter um gráfico de dependência que mostre qual operação é dependente de quais operações.
- Usar marcas de tempo vetoriais para marcar as dependências.

Gerenciamento de Réplicas

- O problema no Gerenciamento de Réplicas é decidir onde, quando e por quem as réplicas devem ser posicionadas.

- Identificar os mecanismos para manter as réplicas consistentes.

Gerenciamento de Réplicas
Problema de Posicionamento

- Pode ser subdividido em dois subproblemas:
  - Posicionar servidores de réplicas;
  - Posicionar conteúdo.

- Posicionar servidores de réplicas: refere-se a achar as melhores localizações para colocar um servidor que pode hospedar um depósito de dados (ou parte dele).
- Posicionamento de conteúdo: refere-se a achar os melhores servidores para colocar conteúdo.
- Antes de decidir o posicionamento de conteúdo, é preciso que os servidores de réplicas já tenham sido posicionados.

Posicionamento do Servidor de Réplicas

- Há vários modos de calcular o melhor posicionamento de servidores de réplicas.
- Problema de otimização, no qual as melhores **K** de **N** posições precisam ser selecionadas → Heurística.
- Uma muito utilizada é a distância entre clientes e localizações.
  - Pode ser medida em termos de latência ou largura de banda.
  - A posição é definida baseada na métrica que a distância média entre o servidor e os seus clientes é mínima.

Posicionamento do Servidor de Réplicas

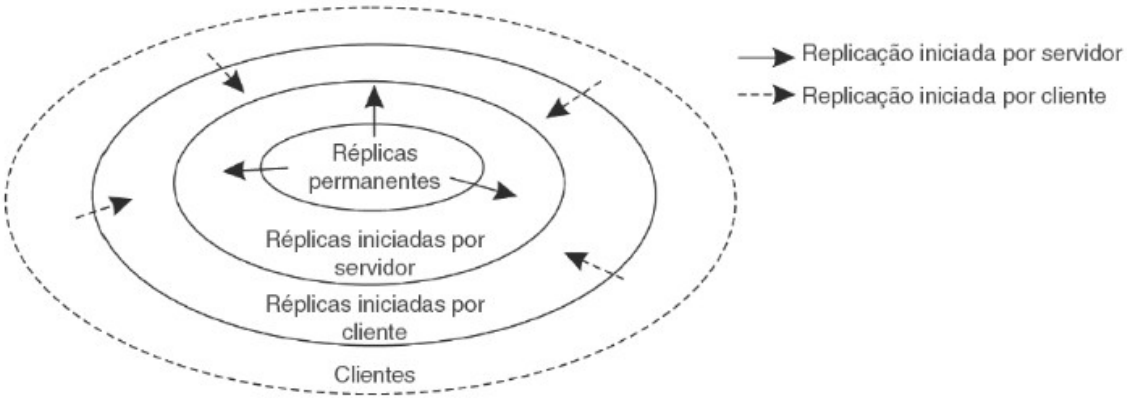
- Ignorar a posição dos clientes e apenas considerar que a topologia da Internet é formada pelos sistemas autônomos (AS).
  - Rede no qual os nós executam o mesmo protocolo de roteamento, que é gerenciada por uma única organização.
  - Consideram o maior AS e colocam um servidor no roteador com maior número de enlaces.
  - Os ASs são escolhidos em ordem de tamanho.

Posicionamento do Servidor de Réplicas

- Desenvolver um método no qual pode-se identificar rapidamente uma região para o posicionamento de réplicas.
  - Uma região é identificada como um conjunto de nós que acessam o mesmo conteúdo, com uma latência baixa entre os nós.
  - Primeiramente, seleciona-se as regiões com o maior número de nós, e permite que um dos nós aja como servidor de réplicas.

Replicação e Posicionamento de Conteúdo

- Há três tipos diferentes de réplicas organizadas logicamente:
  - Réplicas permanentes.
  - Réplicas iniciadas por servidor.
  - Réplicas iniciadas por cliente.
- Organização lógica de diferentes tipos de cópias de um depósito de dados em três anéis concêntricos.



Réplicas Permanentes

- Conjunto inicial de réplicas que constituem um depósito de dados distribuídos.
- O número de réplicas permanentes é pequeno.
- Exemplo: Site da web.
  - Arquivos são replicados para um número limitado de servidores que estão em uma única localização.
  - Espelhamento: o site da web é copiado para um número limitado de servidores, sites espelhados, que estão geograficamente espalhados pela internet.

Réplicas Iniciadas por Servidor

- Cópias de um depósito de dados que existem para aprimorar desempenho e que são criadas por iniciativa do (proprietário do) depósito de dados.

- Exemplo:

  - Considere um servidor em Dourados. Pode acontecer uma rajada de requisições que vêm de uma localização inesperada, longe do servidor.

  - Neste caso, talvez valha a pena instalar uma quantidade de réplicas TEMPORÁRIAS nas regiões onde estão vindo as requisições.

Réplicas Iniciadas por Clientes

- São mais conhecidas como *caches* (de cliente).
- Apresenta recurso de armazenamento local, usado por um cliente para armazenar temporariamente uma cópia de dados que ele acabou de requisitar.
- O gerenciamento cabe inteiramente ao cliente: depósito de dados de onde os dados foram trazidos nada tem a ver com a manutenção da consistência dos dados em *cache*.
- *Cache* são usadas para melhorar o tempo de acesso aos dados.
- Quando um cliente quer acessar alguns dados, se conecta com a cópia do depósito de dados mais próxima.
- Quando a maioria das operações envolve somente ler
- dados, o desempenho pode ser melhorado, dado que o

Distribuição de Conteúdo

- O gerenciamento de réplicas também trata da propagação de conteúdo (atualizado) para os servidores de réplicas relevantes.

Estado *versus* Operações

- Pode-se propagar:
  - Uma notificação de atualização;
  - Dados de uma cópia para outra;
  - Operação de atualização para outras cópias.

Notificação de Atualização

- Realizada por protocolos de invalidação
- Cópias são informadas de uma atualização que ocorreu e que os dados que elas contêm não são mais válidos
- Dados não são propagados
- Toda vez que é requisitada uma operação em uma cópia invalidada, em geral esta cópia precisa ser atualizada anteriormente, segundo o modelo de consistência.

Notificação de Atualização

- Principal vantagem dos protocolos de invalidação é que estes utilizam pouca largura de banda de rede
- Única informação: especificação de quais dados não são mais válidos
- Estes protocolos funcionam melhor quando há mais operações de atualização em comparação com operações de escrita
  - A razão entre leitura/escrita é relativamente pequena.

Notificação de Atualização

- Exemplo:
  - Considere um banco de dados onde a ocorrência de atualizações é mais frequente que as operações de leitura.
  - Pode-se ter a situação em que duas atualizações ocorrem uma após a outra sem que nenhuma operação de leitura seja realizada entre elas.
  - A propagação da primeira atualização é inútil, porque terá que ser sobrescrita pela segunda atualização.
  - Neste caso, o mais eficiente é enviar uma notificação.

Transferir Dados entre Cópias

- Útil quando a razão leitura/escrita é relativamente alta.
- A probabilidade de uma atualização ser efetivada é grande.
- O fato de haver um grande número de leituras faz com que os dados devam ser modificados antes de ocorrer a atualização seguinte.

Propagação da Operação de Atualização

- Neste último caso, não é transferido nenhuma modificação de dados, mas somente a operação de atualização que a réplica deve realizar e os parâmetros necessários para a atualização.
- Esta abordagem é denominada replicação ativa.
- O principal benefício da replicação ativa é que as atualizações muitas vezes podem ser propagadas com custos mínimos de largura de banda.

Envio *versus* recuperação de atualizações

- Protocolo baseado em envio (servidor):
  - As atualizações são propagadas por outras réplicas sem solicitação.
- Protocolo baseado em recuperação (cliente):
  - Um servidor ou cliente requisita que um outro servidor lhe envie quaisquer atualizações que ele tiver no momento em questão.

Protocolos de Envio de Atualizações

- Costumam ser usados entre réplicas permanentes e réplicas iniciadas por servidor, mas também podem ser usadas para enviar atualizações a caches de clientes.
- São aplicados, de modo geral, quando as réplicas precisam manter um grau de consistência relativamente alto.
- A razão leitura/atualização em cada réplica é relativamente alta.
- Dados consistentes estão disponíveis imediatamente após a solicitação.

Protocolo de Recuperação de Atualizações

- Comumente usado por caches de clientes .
- Cliente sonda o servidor para ver se é necessária uma atualização.
- Eficiente quando a razão leitura/atualização é relativamente baixa.
- A principal desvantagem de uma estratégia baseada em recuperação em comparação com uma abordagem baseada em envio é que o tempo de resposta aumenta no caso de uma ausência de cache.

Envio *versus* Recuperação de Atualizações

- Comparação entre protocolos baseados em recuperação de atualizações e envio de atualizações no caso de sistemas com múltiplos clientes e com um único servidor.

Assunto	Baseadas em envio	Baseadas em recuperação
Estado no servidor	Lista de réplicas e caches de clientes	Nenhum
Mensagens enviadas	Atualizar (e possivelmente buscar atualização mais tarde)	Sondar e atualizar
Tempo de resposta no cliente	Imediato (ou tempo de busca–atualização)	Tempo de busca–atualização

Unicast ou Multicast?

- Protocolos Baseados em Envio
  - Podem utilizar de maneira mais eficiente a propagação de informações usando multicast.
    - Os servidores que receberem as atualizações serão organizados em um único grupo.
- Protocolos Baseados em Recuperação
  - Podem utilizar de maneira mais eficiente a comunicação unicast, já que somente um dos clientes pedirá uma atualização.

Protocolos de Consistência

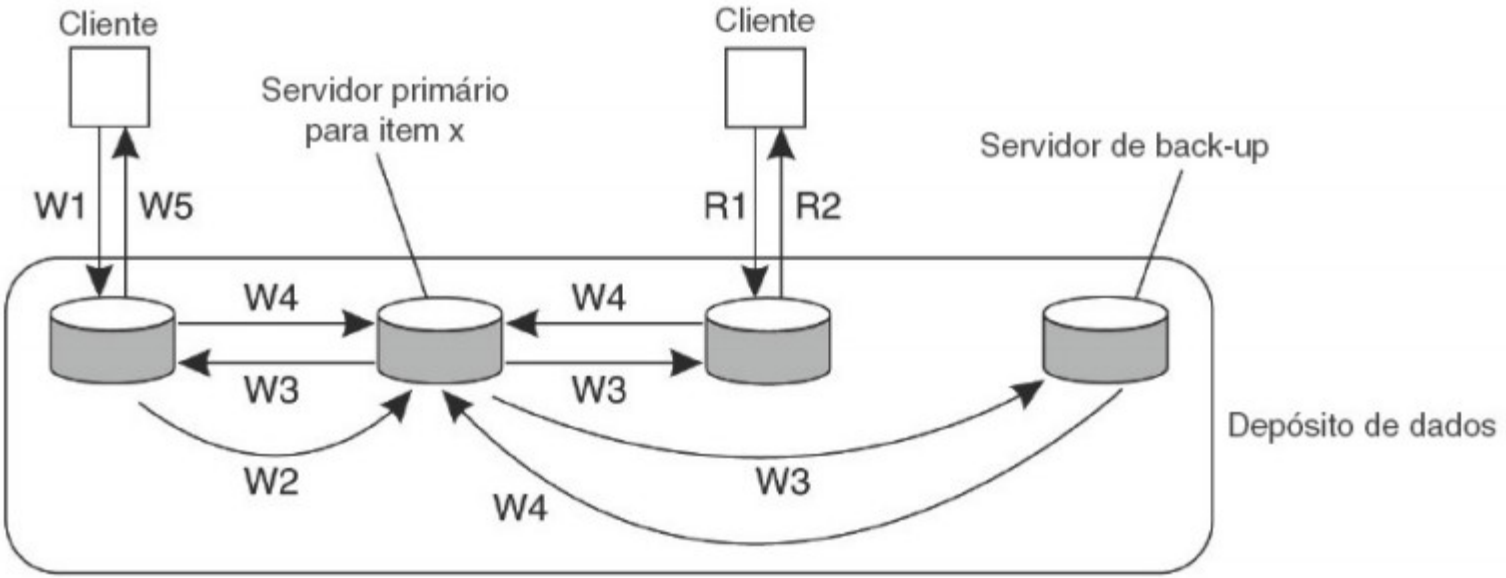
- Um protocolo de consistência descreve uma implementação de um modelo de consistência específico.
- CONSISTÊNCIA CONTÍNUA
  - Limitação de desvio de idade
    - Um servidor k possui um vetor, cujos elementos representam o tempo real em cada servidor i, onde todas as atualizações já foram enviadas a k.
    - Se a diferença entre o tempo local de k e o tempo referente ao servidor i excede um limite, o servidor k recupera as escritas realizadas em i.
    - O servidor de réplicas é responsável por manter sua cópia do dado atualizada em relação às escritas que foram emitidas em outros lugares.

Protocolos Baseados em Primários

- Usado no caso de consistência sequencial.
- Ideia Geral:
  - Cada item de dados **x** no depósito de dados tem um primário associado, que é responsável por coordenar operações de escrita em **x**.
- Podem ser de dois tipos:
  - Escrita Remota.
  - Escrita Local.

Protocolos de Escrita Remota

- Princípio de um protocolo primário e backup.

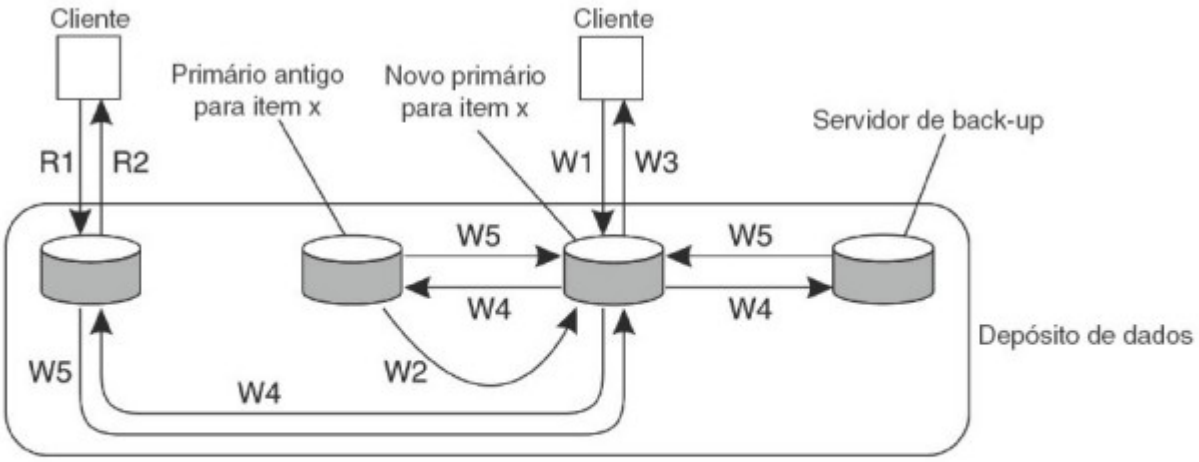


Protocolos de Escrita Remota

- Problema com este tipo de protocolo.
  - O tempo é relativamente longo antes que o processo que iniciou a atualização tenha permissão para continuar.
    - A atualização é implementada como uma operação de bloqueio.
- Estes tipos de protocolos proporcionam uma implementação direta de consistência sequencial porque o servidor primário pode ordenar todas as escritas em uma ordem temporal globalmente exclusiva.

Protocolos de Escrita Local

- Protocolo primário e backup no qual a cópia primária migra para o processo que quer realizar uma atualização.



Protocolos de Escrita Replicada

- As operações de escrita podem ser executadas em várias réplicas em vez de em uma só, como no caso de réplicas baseadas em primários.
- Podem ser de dois tipos:
  - Replicação ativa;
  - Protocolos de consistência baseados em voto majoritário.

Replicação Ativa

- Cada réplica possui um processo associado que realiza as operações de atualização.
- Operações realizadas localmente devem ser propagadas para as outras réplicas.
- Problema: Todas as operações devem ser executadas na mesma ordem em todos os lugares.
  - Multicast totalmente ordenado, usando relógios de lamport.
  - Coordenador central, denominado sequenciador.

Protocolos Baseados em Votação

- Ideia: Clientes devem pedir permissão de vários servidores antes de ler ou escrever um item de dados replicados.
- Exemplo:
  - Considere um sistema de arquivos onde um arquivo é replicado em N servidores.
  - Para atualizar um arquivo, o cliente deve ter permissão da maioria.
  - Para ler um arquivo replicado, o cliente deve ter também a permissão da maioria.

Pontos Importantes

- Duas razões para replicar dados:
  - Melhorar a confiabilidade de um sistema distribuído;
  - Melhorar o desempenho.
- Replicação introduz um problema de consistência
  - Sempre que uma réplica é atualizada, ela se torna diferente das outras.
  - As atualizações devem ser propagadas.
  - Ocorre degradação do desempenho.