**Protocolos de Consistência**

Um protocolo de consistência descreve uma implementação de um modelo de consistência específico.

**Entendendo as necessidades de Consistências**

**Replicação de dados**

Para que replicar dados? Há duas razões primordiais.

- Melhorar confiabilidade de um SD;

- Melhorar desempenho.

Contudo, a replicação de dados introduz um problema de consistência, pois sempre que uma réplica é atualizada, ela se torna diferente das outras.

É necessário propagar atualizações de tal modo que inconsistências temporárias não sejam notadas, com a finalidade de manter réplicas consistentes.

O contra: fazer isso degrada o desempenho, principalmente em sistemas distribuídos de grande porte.

A saída: relaxar um pouco a consistência.

**Consistência Contínua**

Dentro do nosso contexto, existem diferentes modelos de consistência.

Para que haja consistência contínua, a meta é estabelecer limites para o desvio numérico entre réplicas, para o desvio entre idades e para desvios entre as ordenações de operações.

**Desvio numérico**

Refere-se ao valor da diferença entre réplicas que pode ser tolerado.

É um desvio que depende muito de aplicação, mas pode ser usado na replicação de valores de ações.

**Desvio de idade**

O desvio de idade se refere ao tempo durante o qual uma réplica ainda é considerada consistente.

Costuma ser usado para caches web.

**Desvio de ordenação**

Refere-se ao número máximo de escritas provisórias que podem ficar pendentes em qualquer servidor sem terem sido sincronizadas com outros servidores de réplicas.

**Protocolos baseados em primários**

Todas as operações de atualização são repassadas para uma cópia primária que, na sequência, garante que a atualização seja adequadamente ordenada e repassada.

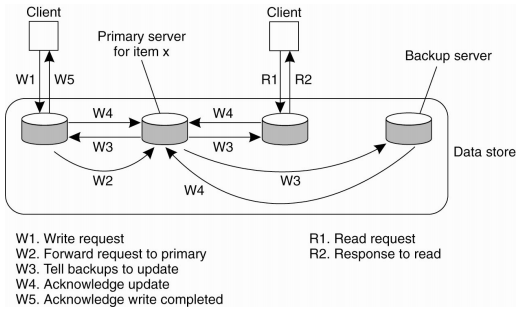
**Protocolos de escrita remota**

É o protocolo mais simples baseado em primário e que suporta replicação.

Nesse caso, as operações de escrita precisam ser enviadas para um único servidor fixo.

Operações de leitura podem ser executadas localmente.

São esquemas também conhecidos como **Protocolos de primário e backup**.



Um processo que quer realizar uma operação de escrita, em um item de dados *x*, envia essa operação para o servidor de primários para *x.* O servidor primário executa a atualização em sua cópia local de *x* e, na sequência, envia a atualização para os servidores de backup. Cada servidor de backup também efetua a atualização e envia um reconhecimento de volta ao servidor primário. Quando todos os seervidores de backup tiverem atualizado sua cópia local, o servidor primário envia um reconhecimento de volta ao processo inicial.

**Problema desse esquema:** pode levar muito tempo antes que o processo que iniciou a atualização tenha permissão para continuar.

**Solução para esse esquema:** Usar uma abordagem não bloqueadora. Tão logo o servidor primário tenha atualizado sua cópia local de *x,* ele retorna um reconhecimento. Depois disso, diz ao servidor primário de backup que também efetue a atualização.

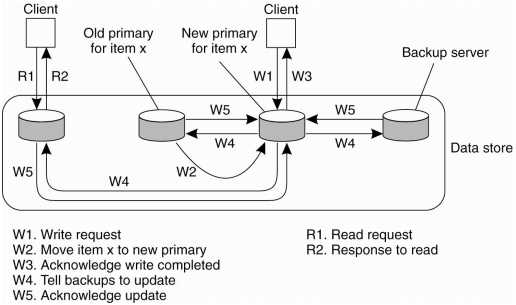
**Principal problema:** O principal problema de protocolos de primário-backup não bloqueadores tem a ver com tolerância a falha, pois o processo cliente não sabe que a operação de atualização é apoiada por vários outros servidores.

**A vantagem:** As operações de escrita podem ser consideravelmente aceleradas.

**Protocolos de escrita local**

É uma variante dos protocolos de primário e backup onde a cópia primária migra entre processos que desejam realizar uma operação de escrita.

Como anteriormente, sempre que um processo quer atualizar o item de dados *x*, ele localiza a cópia primária de *x* e, na sequência, move essa cópia para sua própria localização.



**A vantagem:** Múltiplas operações sucessivas de escrita podem ser executadas no local enquanto processos leitores ainda podem acessar sua cópia local.

**O porém:** Só é possível conseguir tal melhoria se for seguido um protocolo não bloqueador pelo qual as atualizações são propagadas para as réplicas após o servidor primário ter concluído as atualizações realizadas localmente.

**Possível aplicação:** Computadores móveis que são capazes de operar em modo desconectado. Enquanto conectado, o computador móvel é o servidor primário para cada item de dados que ele espera atualizar. Enquanto desconectado, todas as operações de atualização são executadas localmente. Ao conectar-se novamente, as atualizações são propagadas do primário para os backups.

**Protocolos de escrita replicada**

Uma atualização é repassada a diversas réplicas ao mesmo tempo. Nesse caso, a ordenação correta das operações costuma ficar mais difícil.

**Replicação ativa**

Nesse caso, cada réplica tem um processo associado que realiza as operações de atualização.

Ao contrário de outros protocolos, as atualizações são propagadas por meio da operação de escrita que causa a atualização.

**Traduzindo:** A operação é enviada para cada réplica.

**O problema:** As operações precisam ser executadas na mesma ordem em todos os lugares.

**A solução:** Nesse caso é necessário um mecanismo de multicast totalmente ordenado, baseado em relógios lógicos de Lamport.

**Inviabilidade:** A implementação do multicast é muito complexa em grandes SD’s.

**A altenativa:**  É possível conseguir ordenação total com o uso de um coordenador central, também denominado ***sequenciado***.

**A abordagem:**  Primeiramente, repassar cada operação ao sequenciador, que lhe designa um número de sequência exclusivo, e, logo depois enviar a operação para todas as réplicas.

**Protocolos baseados em quórum**

Usar **votação** é uma abordagem diferente para suportar escritas replicadas.

A ideia básica é exigir que clientes requisitem e adquiram a permissão de vários servidores antes de ler ou escrever um item de dados replicados.

**Explicando:**

Para poder atualizar um arquivo, é ncessário que a metade dos servidores +1 (maioria simples) concordem em fazer a atualização. Se concordarem, a atualização ocorre e o número da versão do arquivo é alterada para que seja possível identificar a nova versão.

Para ler um arquivo replicado, um cliente deve contatar no mínimo metade dos servidores +1 e solicitar que eles enviem os números das versões associadas com os arquivos.

**Por que dá certo?**

Se houver cinco servidores e um cliente determinar que 3 deles tem a versão 8, é impossível que outros dois tenham a versão 9. Afinal, qualquer atualização bem-sucedida da versão 8 para a versão 9 requer conseguir que três servidores concordem com isso, e não apenas dois.

**Protocolos coerência em cache**

Caches são um caso especial de replicação, no sentido de que, em geral, são controladas por clientes, em vez de servidores.

Há dois contextos para aplicação dos protocolos de coerência em cache:

1. Em sistemas Multiprocessadores de memória compartilhada;
2. Em Sistemas Distribuídos baseados em middleware.

Aboraderemos o segundo caso onde as soluções para caches baseadas em SW são mais interessantes.

Nesse caso, há dois critérios de estratégia de classificação:

1. Estratégia de detecção de coerência;
2. Estratégia de de imposição de coerência.

**Estratégia de detecção de coerência**

O primeiro caso ocorre quando as inconssistências são realmete detectadas.

Aplica-se a soluções estáticas, onde um compilador realiza a análise necessária anterior à execução e determina quais dados podem realmente levar a inconsistências porque podem ser colocados em cache. Assim, o compilador simplesmente insere instruções que evitam inconsistências.

Já nos SD’s , normalmente, são aplicadas soluções dinâmicas. Nesse caso, as inconsistências são detectadas em tempo de execução.

Por exemplo, é feita uma verificação no servidor para ver se os dados em cache foram modificados desde que entraram na cache.

**Estratégia de imposição de coerência**

Essa estratégia determina como as caches são mantidas consistentes com as cópias armazenadas em servidores.

A solução mais simples é não permitir que dados compartilhados não sejam colocados em cache.

Ao invés disso, dados compartilhados são colocados somente nos servidores, que mantêm consistência usando um dos protocolos baseados em primário ou de replicação de escrita.

Clientes só tem permissão de colocar em cache seus próprios dados privados.

**Quando dados podem ser armazenados em cache**

Para esse caso, há duas abordagens:

1. Permitir que um servidor envie uma invalidação a todas as caches sempre que um item de dados for modificado;
2. Simplesmente propagar a atualização.

**Considerando que um processo modifique dados em cache**

Quando são usadas caches somente de leitura, operações de atualização só podem ser realizadas por servidores que, na sequência, sigam algum protocolo de distribuição para garantir que as atualizações sejam propagadas para as caches.

Em muitos casos é seguida uma abordagem de recuperação de atualizações. Nesse caso, um cliente detecta que sua cache tem itens de dados antigos e requisita uma atualização ao servidor.

**Alternativa**

Permitir que clientes modifiquem diretamente os dados em cache e enviem a atualização aos servidores.

Essa abordagem é seguida em em **caches de escrita direta**, que costumam ser usadas em sistemas de arquivos distribuídos.

**Vantagem**

Caches de escrita direta oferecem melhor desempenho em comparação com outros esquemas, pois todas as operações podem ser executadas no local.

**Implementação de consistência centrada no cliente**

Não consideram o fato de que os dados podem ser compartilhados por diversos usuários, mas se concentram na consistência que deve ser oferecida a um cliente individual.

Essencialmente, modelos de consistência centrados no cliente garantem que, sempre que um cliente se conectar com uma nova réplica, essa réplica seja atualizada com os dados que tinham sido manipulados por aquele cliente antes e que possivelmente residam em outros sites de réplicas.

Implementar consistência centrada no cliente é algo relativamente direto se forem ignoradas questões de desempenho.

**Implementação ingênua**

A cada operação de escrita *W* é designado um identificador globalmente exclusivo. Esse identificador é designado pelo servidor ao qual a escrita foi apresentada.

Esse servidor é conhecido como a origem de *W.*

São monitorados dois conjuntos de escritas para cada cliente.

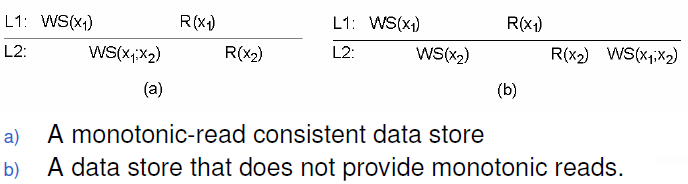
O conjunto de leitura para um cliente consiste nas escritas relevantes para as operações de leituras executadas por esse cliente. Da mesma maneira, o conjunto de escrita consiste nas escritas realizadas pelos clientes.

São quatro os modelos:

**Leituras Monotônicas** – garante que após ler um objeto de um dado processo nunca lerá uma versão anterior do mesmo;

Definição:

Se um processo ler o valor de um item de dados X, qualquer operação de leitura sucessiva de X executada por esse mesmo processo sempre retornará o mesmo valor ou um valor mais recente. Isto é, se um processo viu um valor de X no tempo t, ele nunca verá uma versão mais velha de X em um tempo posterior.

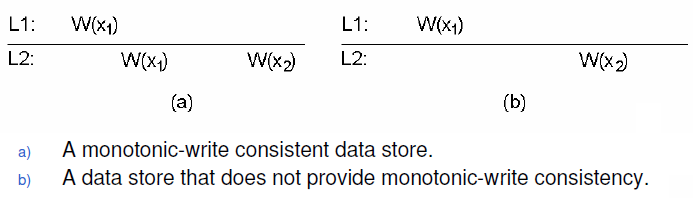


Exemplo: Banco de dados distribuído de email

**Escritas Monotônicas** – garante que toda escrita sobre um objeto será completada em todas as cópias antes que o mesmo processo faça outra operação de escrita sobre ele;

Definição:

Uma operação de escrita executada por um processo em um item de dados X é "concluída" antes de qualquer operação de escrita sucessiva em X pelo mesmo processo.

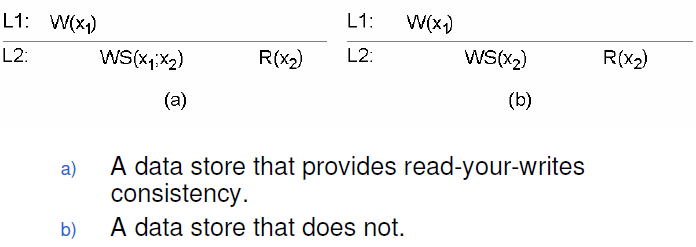


Concluir uma operação de escrita significa que a cópia na qual uma operação sucessiva é executa, reflete o efeito de uma operação de escrita anterior executada pelo mesmo processo, independente de onde essa ocorreu.

Exemplo: Biblioteca de software.

**Leia-suas-escritas** – garante que a leitura de um objeto escrito pelo mesmo processo terá sempre seu valor atualizado;

Definição:

O efeito de uma operação de escrita por um processo no item de dados X sempre será visto por uma operação de leitura sucessiva em X pelo mesmo processo. 

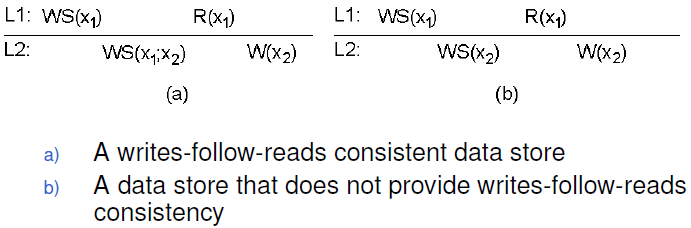
Isto é, uma operação de escrita é sempre concluída antes de uma operação de leitura sucessiva pelo mesmo processo, independente de onde essa ocorrer.

Exemplo: Senhas (podem ocorrer problemas), páginas Web (podem ser visualizadas páginas antigas, armazenadas em cache).

**Escritas-seguem-leituras** – garante que a operação de escrita de um objeto recém-lido ocorrerá sobre o mesmo valor ou ocorrerá sua atualização;

Definição:

Garante-se que uma operação de escrita por um processo em um item de dados X em seguida a uma operação de leitura anterior em X pelo mesmo processo ocorre sobre o mesmo valor, ou sobre o valor mais recente de X que foi lido.



Isto é, qualquer operação de escrita sucessiva executada por um processo em um item de dados X será realizada sobre uma cópia de X atualizada com o valor lido mais recentemente por esse processo.

Exemplo: Grupo de discussão sobre um artigo.

**Como melhorar a eficiência**

É possível observar que o conjunto de leitura e o conjunto de escrita associados com cada cliente podem se tornar muito grandes. Para manter a facilidade de gestão desses conjuntos as operações de leitura e escrita dos clientes são agrupadas por sessão. Uma sessão é associada a uma aplicação e ela é aberta quando a aplicação começa e fechada quando a aplicação sai.

**Problema principal:** Sempre que um servidor recebe uma requisição de leitura ou escrita de um cliente, o servidor também recebem um conjunto de identificadores para que ele veja se todas as operações de escrita relevantes para a requisição foram executadas por esse servidor

**Como funciona:** Sempre que um servidor aceita uma nova operação de escrita, ele designa a essa operação um identificador globalmente exclusivo junto com uma marca de tempo.

Uma operação de escrita posterior é apresentada àquele servidor e recebe uma marca de tempo de valor mais alto, e cada servidor também tem uma marca de tempo vetorial.

Sempre que um cliente envia uma requisição para executar uma operação de leitura ou de escrita em um servidor especifico, esse servidor retorna sua marca de tempo corrente com os resultados solicitados, em seguida conjuntos de leituras e de escritas são representados por marcas de tempo vetorial

Pode se dizer que as marcas de tempo de uma sessão sempre representa as últimas operações de escrita que foram vistas pelas aplicações em execução como parte dessa sessão. A compactação é obtida representando todas as operações de escrita observadas que se originaram de um mesmo servidor por uma única marca de tempo

**Exemplo:** Suponha que um cliente, como parte da sessão A, se registre em um servidor qualquer X, com essa finalidade ele passa sua marca de tempo vetorial para o servidor X, se o servidor ainda não visualizou todas as escritas originadas de outro servidor Y, dependendo da consistência requerida, agora o servidor X tem de buscar as escritas do servidor Y e responder a solicitação do cliente. Quando essa operação é realizada o servidor X retorna sua marca de tempo corrente. Nessa hora a marca de tempo vetorial é atualizada.

Dessa forma, conclui-se que marcas de tempo vetorial podem proporcionar um modelo elegante e compacto de representar históricos em um sistema distribuído.