**Aplicações Móveis**

Exercícios de Revisão

**Coordenação de Engenharia Informática**

Departamento de Engenharias e Tecnologias

Instituto Superior Politécnico de Tecnologias e Ciências

Nome: Marcelo Rocha - 20210032

# **Replicação**

1. **Em um sistema replicado, o que é consistência?**

**R:** Em um sistema replicado, consistência refere-se ao grau de uniformidade no comportamento e nos dados disponibilizados aos usuários, independentemente da replicação.

Isso significa que, idealmente, um sistema consistente apresenta os mesmos resultados que um sistema não replicado, como se houvesse apenas uma única cópia dos dados (propriedade chamada de one-copy serializability). Assim, qualquer leitura dos dados deve refletir as atualizações mais recentes realizadas no sistema, garantindo que todas as réplicas estejam sincronizadas e atualizadas de forma coordenada.

A consistência determina o quanto os usuários e as aplicações precisam se preocupar com os detalhes do processo de replicação e as políticas de sincronização.

1. **Em relação a consistência, qual é a noção de: fraca/melhor esforço / eventual / causal / limitada / VFC/sessão?**

**R:**

**Consistência Forte:** é um modelo de consistência em sistemas distribuídos onde todas as réplicas de um conjunto de dados são garantidamente consistentes em todos os momentos. Isso significa que qualquer operação de leitura sempre retorna os dados mais recentes, refletindo todas as atualizações feitas, independentemente de qual réplica é acessada (comportamento similar a um sistema não replicado).

**Consistência Fraca**: é um modelo de consistência para sistemas distribuídos em que o sistema não garante que todas as réplicas de dados estejam sincronizadas a qualquer momento. Em vez disso, ele fornece uma abordagem mais flexível, onde as garantias de consistência podem ser relaxadas para priorizar disponibilidade e desempenho em detrimento da coerência imediata dos dados.

**Consistência Melhor Esforço :** Nesse modelo, o sistema faz o "melhor esforço" para entregar as atualizações a todas as réplicas, mas não garante consistência real. Diferentes réplicas podem ter dados diferentes, mesmo que nenhuma atualização recente tenha sido realizada. O sistema tenta propagar as atualizações para todas as réplicas, mas não garante que todas as réplicas eventualmente cheguem ao mesmo estado. Não há compromissos formais sobre a ordem ou completude das atualizações, e leituras podem retornar resultados inconsistentes.

**Consistência Eventual:** Garante que todas as réplicas irão eventualmente convergir para um estado consistente, desde que as atualizações cessem. Se nenhuma nova atualização for realizada, todas as réplicas convergirão para um estado consistente. Permite divergências temporárias durante a propagação das atualizações.

**Consistência Causal:** Garante que operações relacionadas causalmente sejam vistas na ordem correta em todas as réplicas. Por exemplo, se U2 depende de U1, U1 deve ser visível antes de U2 em qualquer réplica.

**Consistência de Sessão:** Garante que um único usuário ou aplicação tenha uma visão consistente das operações de leitura e escrita realizadas durante uma sessão. Garante consistência para operações realizadas por um único usuário ou aplicação durante uma sessão.

**Consistência Limitada (Bounded Consistency):** Define limites específicos para a inconsistência, como tempo, erro numérico ou ordem , ou seja , estabelece limites para o desvio permitido entre réplicas em termos de tempo, valores, ou ordem. Por exemplo, garantir que dados lidos não tenham mais de 1 minuto de atraso.

**Consistência VFC (Virtual Full Consistency ou Híbrida):** Apresenta uma escolha entre consistência forte e fraca, dependendo das necessidades da aplicação permitindo que aplicações escolham o nível apropriado para cada operação.

1. **Conforme introduzido por Bayou, o que é uma sessão? Uma sessão corresponde a uma transacção atómica?**

**R:** Uma sessão é uma abstração que representa a sequência de operações de leitura e escrita realizadas durante a execução de uma aplicação. A ideia é que, mesmo que a aplicação acesse servidores diferentes que possam estar em estados inconsistentes, ela veja uma visão consistente do banco de dados com base nas suas próprias ações.

No entanto, uma sessão não corresponde a uma transação atômica. Diferentemente de uma transação atômica, que garante atomicidade (ou seja, ou todas as operações de leitura e escrita são bem-sucedidas ou nenhuma delas é), as sessões não garantem atomicidade e serialização (a ordem exata das operações). A principal intenção das sessões é fornecer uma visão consistente para o aplicativo, semelhante à de um servidor centralizado, mas sem as garantias rigorosas de uma transação atômica, como no caso de sistemas com replicação fraca.

Portanto, as sessões buscam mitigar o problema de inconsistência fraca em sistemas distribuídos, oferecendo uma forma de consistência mais flexível, sem os custos e restrições de transações atômicas.

1. **Considere a garantia da sessão Escritas Monotónicas / ou qualquer outra. Em que instantes (operação de escrita ou leitura) o estado da sessão é: i) actualizado e ii) verificado. Justifique sua resposta.**

**R:** Quando uma operação de escrita é realizada, o estado da sessão é atualizado, ou seja, o valor da variável ou local de memória compartilhada é alterado para refletir a nova escrita , garantindo que o estado seja modificado com a nova informação.

O estado da sessão é verificado durante a operação de leitura. Durante uma operação de leitura, o sistema verifica se o valor lido está em conformidade com a ordem das escritas realizadas, garantindo que a leitura retorne sempre o valor mais recente ou um valor que siga a sequência das escritas anteriores (no contexto de *Escritas Monotônicas*).

1. **Leve em consideração a consistência de melhor esforço. Indique quatro situações em que, mesmo com entrega confiável, as réplicas não convergirão.**

**R:**

* As atualizações forem realizadas de forma diferente em diferentes réplicas (ou seja, a aplicação de uma atualização não é determinística);
* As atualizações forem aplicadas em ordens diferentes em diferentes réplicas e não forem comutáveis;
* As réplicas tiverem diferentes políticas de resolução de conflitos;
* Metadados, como tombstones de exclusão, forem descartados muito cedo;
* As réplicas perderem ou corromperem dados, como quando uma réplica é restaurada de um backup antigo;
* O sistema estiver mal configurado, como quando a topologia de sincronização não forma um grafo bem conectado.

1. **Considere as várias garantias de sessão possíveis em um sistema distribuído com uma base de dados replicada em vários servidores (por exemplo, Bayou). Tal sistema é utilizado em um cenário de uma empresa que comercializa diversos modelos de pneus e cujos vendedores, cada um utilizando um dispositivo móvel, podem se conectar a qualquer servidor a qualquer momento:**

* 1. **Existe uma tabela da base de dados onde cada registo está relacionado a um modelo de pneu e suas características;**
  2. **Cada registo corresponde a um modelo específico de pneu;**
  3. **Cada campo de cada registo na tabela corresponde a uma característica do produto (por exemplo, preço, unidades existentes, etc.);**
  4. **Cada um dos vendedores vende apenas um modelo de pneu distinto dos demais;**
  5. **A base de dados é modificada pelos vários vendedores, de forma a refletir o número de unidades já vendidas, ou seja, cada vendedor diminui/aumenta o número de unidades existentes tendo em conta as unidades vendidas/compradas no modelo de pneu pelo qual é responsável.**

**Qual garantia de sessão deve ser aplicada para garantir que cada vendedor saiba sempre o valor exacto das unidades existentes do modelo que vende?**

**Indique a garantia menos exigente e justifique a sua resposta.**

**R:**

A garantia de sessão mais adequada para garantir que cada vendedor saiba sempre o valor exato das unidades existentes do modelo de pneu que vende é a garantia de leitura própria (*read-your-writes*). A garantia de leitura própria assegura que, após um vendedor realizar uma operação de escrita, qualquer leitura subsequente feita por ele refletirá a sua própria escrita. Isso significa que o vendedor terá uma visão consistente das alterações que ele mesmo realizou no modelo de pneu que vende.

Cada vendedor precisa apenas garantir a consistência das operações sobre o modelo de pneu que ele gerencia, sem necessidade de coordenar suas leituras/escritas com os de outros vendedores.

A garantia de leitura própria é menos exigente em comparação com outras garantias, como escritas monotônicas ou consistência causal, porque ela não exige que as operações de outros vendedores (em outros modelos de pneus) sejam refletidas imediatamente para o vendedor atual. Ela se limita a assegurar a consistência das operações realizadas por um único vendedor no modelo de pneu específico que ele gerencia.

Essa abordagem reduz a complexidade e a sobrecarga de comunicação entre os servidores replicados, já que não há necessidade de sincronizar dados entre os vendedores em tempo real, respeitando as características do cenário.

1. **Considere a seguinte frase e diga se concorda (ou não) e porquê: “Os sistemas eventualmente consistentes não oferecem nenhuma garantia sobre a actualização dos dados retornados por uma operação de leitura”. Dê um exemplo que ilustre sua resposta.**

**R:** Concordo com a frase: “Os sistemas eventualmente consistentes não oferecem nenhuma garantia sobre a atualização dos dados retornados por uma operação de leitura”, pois a consistência eventual garante apenas que, em algum momento no futuro, todas as réplicas irão convergir para um estado consistente, desde que nenhuma nova atualização seja feita. No entanto, não há garantia de que uma leitura realizada em determinado momento retorne os dados mais recentes ou consistentes.

Imagine um sistema de contatos replicado em vários dispositivos (smartphone, tablet e laptop), onde um usuário atualiza o número de telefone de um contato no smartphone.Logo depois, ele realiza uma consulta ao mesmo contato no laptop, mas o número antigo ainda é exibido porque a atualização não foi propagada para o laptop. Eventualmente, após a sincronização entre as réplicas, o número atualizado será exibido em todos os dispositivos.

Esse exemplo demonstra que, mesmo que uma atualização válida tenha sido feita, o sistema não garante que uma leitura imediata em outro dispositivo reflita essa atualização.

1. **Considere a seguinte frase e diga se concorda: a garantia de sessão “Read your Writes” garante que, em cada cópia da base de dados, as escritas feitas durante a sessão sejam ordenadas após quaisquer escritas cujos efeitos foram vistos por leituras anteriores na sessão. Justifique sua resposta.**

**R:** Concordo com a frase, pois a garantia de sessão "Read Your Writes" (RYW) assegura que as leituras subsequentes dentro da mesma sessão reflitam todas as escritas feitas anteriormente na mesma sessão. Isso implica que as escritas feitas durante a sessão estão logicamente ordenadas após quaisquer escritas cujos efeitos foram vistos por leituras anteriores na sessão.

A garantia RYW estabelece que, se um usuário realizar uma escrita em uma sessão e, em seguida, realizar uma leitura dentro da mesma sessão, a leitura sempre refletirá a escrita anterior. Isso evita situações onde o usuário escreve algo no banco de dados, realiza uma leitura e não vê o valor atualizado.

Durante a sessão, se uma leitura observar os efeitos de escritas realizadas fora da sessão (por exemplo, por outros usuários), todas as escritas subsequentes feitas na mesma sessão devem ser ordenadas após essas escritas externas. Isso garante consistência lógica dentro da sessão.

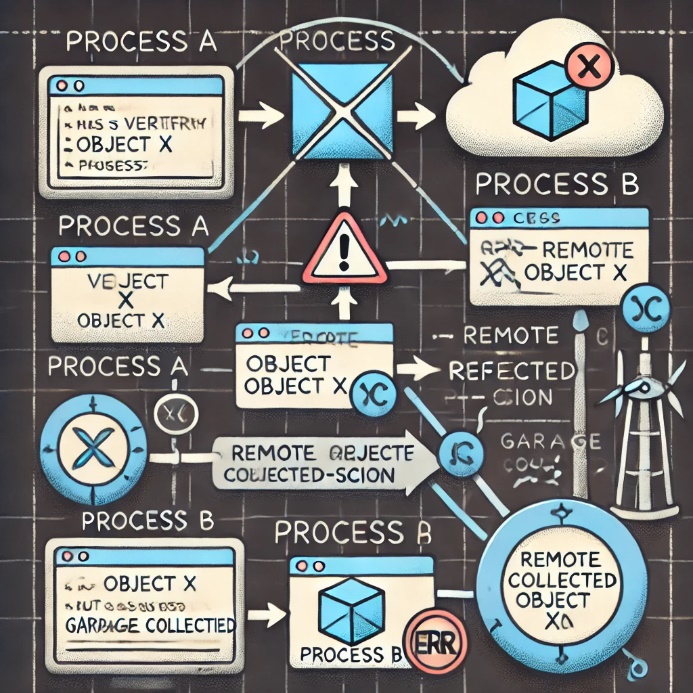
Não apenas assegura que as leituras refletem as escritas feitas na mesma sessão, mas também que a ordem lógica entre escritas e leituras é mantida, o que é essencial para evitar inconsistências percebidas pelo usuário.

1. **Considere a seguinte regra: “antes de enviar uma mensagem contendo uma réplica de um objecto X de um processo P, X deve ser verificado quanto a referências e os correspondentes GC-scions criados em P.”. Como é chamada esta regra? O que pode acontecer se esta regra não for assegurada pelo sistema? (desenhe uma figura descrevendo o que acontece se a regra não for garantida.)**

**R:** Essa regra está relacionada ao "garbage collection" distribuído e é comumente chamada de **Clean Before Send Replica**.

Quando um objeto X é enviado de um processo P para outro processo, todas as referências desse objeto precisam ser verificadas. Para cada referência existente, devem ser criados GC-scions no processo P, que funcionam como "marcadores" para manter o controle de referências externas ao objeto X. Se essa regra não for assegurada, podem ocorrer referências pendentes ou referências "órfãs" que levam a falhas graves no gerenciamento de memória liberando recursos associados ao objeto 𝑋 causando inconsistências ou erros de acesso a memória liberada.

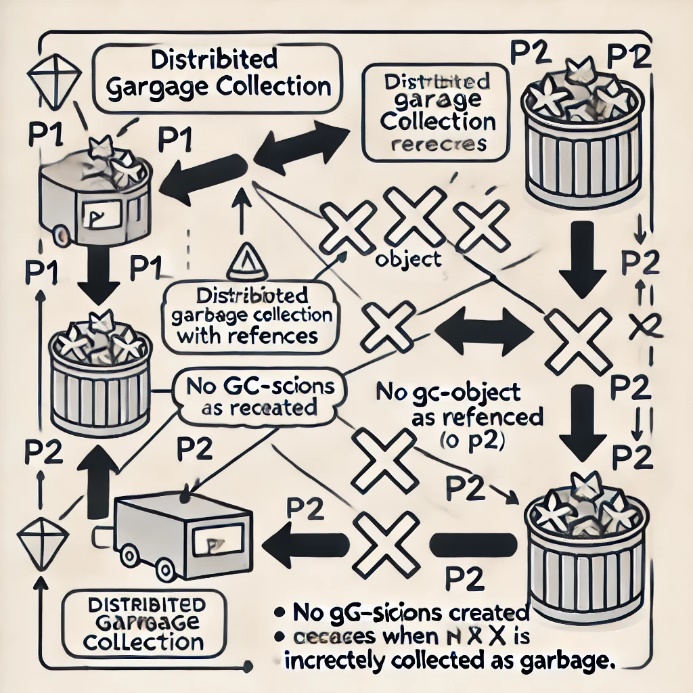
Sem GC-scions adequados, o objeto X pode permanecer na memória do processo P, mesmo que não seja mais utilizado, levando ao acúmulo de objetos não coletados e consumo excessivo de recursos.



1. **Considere a seguinte regra: “antes de entregar uma mensagem contendo uma réplica de um objecto X em um processo P, X deve ser verificado quando as referências e os GCstubs correspondentes criados em P.”. Como é chamada essa regra? O que pode acontecer se esta regra não for assegurada pelo sistema? (desenhe uma figura descrevendo o que acontece se a regra não for garantida.)**

**R:** A regra mencionada é chamada de "**Clean Before Deliver Replica**". Ela estabelece que, antes de entregar uma mensagem contendo uma réplica de um objeto X em um processo P, o objeto deve ser verificado quanto às referências de saída e os correspondentes GC-stubs (referências de coleta de lixo para objetos externos) devem ser criados. Se essa regra não for assegurada pelo sistema, podem ocorrer os seguintes problemas:

* **Referências Incorretas:** Sem a verificação adequada das referências de saída, o objeto pode ser entregue com referências erradas ou não tratadas, o que pode levar a inconsistências nos dados ou falhas de comunicação entre os processos envolvidos.
* **Coleta de Lixo Ineficiente:** Se os GC-stubs não forem criados corretamente, a coleta de lixo pode ser mal feita. Referências inter-processos podem não ser detectadas, resultando em objetos que não são coletados adequadamente, o que pode levar a vazamentos de memória.
* **Desempenho Reduzido:** Falhas na criação de GC-stubs podem causar sobrecarga no gerenciamento de memória, já que o sistema pode não conseguir identificar corretamente quando um objeto está inacessível, afetando a eficiência da coleta de lixo.
* **Erros de Consistência:** A entrega de réplicas com referências não verificadas pode resultar em inconsistências entre os processos, já que um processo pode estar operando com dados desatualizados ou incorretos, comprometendo a integridade do sistema.



1. **Considere um utilizador a editar um ficheiro replicado em seu laptop. Quando um ficheiro é guardado, é armazenado em um servidor; então, o sistema é responsável por propagar a nova versão do ficheiro para outros servidores. O sistema deve garantir que, e o utilizador guardar a versão N do ficheiro e depois salvar a versão N+1, a versão N+1 substituirá a versão N em todos os servidores. Portanto, a seguinte situação deve ser evitada: a versão N é guardada em um servidor e versão N+1 é guardada em um servidor diferente, e ambas versões são propagadas de forma que a versão N seja aplicada após N+1. Qual é a garantia de sessão mais adequada?**

**R:** A garantia de sessão mais adequada para essa situação é a Monotonic Writes (MW). Ela assegura que as escritas (Writes) dentro de uma sessão sejam propagadas de forma ordenada, ou seja, se uma versão N do arquivo foi salva e, posteriormente, uma versão N+1 for salva, a versão N+1 substituirá corretamente a versão N em todos os servidores, evitando que a versão N seja propagada após a versão N+1.

Isso garante que, independentemente de para qual servidor as versões são enviadas, a ordem das versões será mantida, e não ocorrerá a situação em que a versão N é aplicada depois de N+1, o que poderia causar inconsistências nos dados replicados. Mas não garante que outros utilizadores, fora da sessão, tenham imediatamente acesso à versão mais recente, a menos que o sistema implemente mecanismos adicionais, como consistência causal ou forte.

1. **Considere a garantia de sessão Read Your Writes. Essa garantia de sessão afecta os utilizadores fora da sessão em consideração?**

**R:** A garantia de sessão Read Your Writes (RYW) não afeta os utilizadores fora da sessão em consideração. Ela garante que as leituras dentro de uma sessão reflitam as escritas feitas anteriormente dentro da mesma sessão, ou seja, qualquer atualização realizada em um servidor será imediatamente visível para leituras subsequentes dentro da mesma sessão. No entanto, não há garantia de que outras sessões ou usuários fora da sessão vejam a mesma versão ou as atualizações de imediato.

Isso significa que, enquanto um usuário dentro de uma sessão pode ver as mudanças que fez (por exemplo, após mudar sua senha ou deletar um e-mail), outros usuários, ou mesmo o mesmo usuário em outra sessão, podem não ver essas atualizações até que a propagação da escrita ocorra entre os servidores ou até que o sistema implemente mecanismos adicionais de consistência.

1. **Considere o seguinte cenário. O calendário de compromisso de um utilizador é armazenado online em uma base de dados replicada, onde pode ser actualizada tanto pelo utilizador quanto pelos agendadores automáticos de reuniões. O programa de calendário do utilizador actualiza periodicamente sua apresentação (tela) lendo todos os compromissos de calendário de hoje da base de dados. Se acessar servidores com cópias inconsistentes da base de dados, as reuniões recentemente adicionadas (ou excluídas) podem parecer ir e vir, a menos que uma garantia de sessão impeça que isso aconteça. Qual é a garantia de sessão mais fraca que deve ser usada?**

**R:** A garantia de sessão mais fraca que deve ser usada neste cenário é a Monotonic Reads (MR). Essa garantia garante que as leituras subsequentes dentro de uma sessão serão feitas em cópias da base de dados que contêm todas as atualizações vistas nas leituras anteriores.

No caso do calendário de compromissos, a Monotonic Reads garante que, ao ler os compromissos de hoje, a aplicação sempre verá uma versão de dados que é tão atual ou mais atual que a última leitura. Isso evita a situação onde uma reunião recém-adicionada ou excluída poderia desaparecer ou reaparecer erroneamente ao acessar servidores com cópias inconsistentes da base de dados.

1. **Considere as garantias de quatro secções apresentadas por Bayou. Explique cada uma e dê um exemplo.**

**R:**

* **Read Your Writes (RYW):** Esta garantia assegura que, dentro de uma sessão, qualquer leitura realizada após uma escrita refletirá a alteração feita pela escrita anterior. Ou seja, as leituras sempre visualizarão a versão mais recente dos dados que foram alterados dentro da mesma sessão.
* **Exemplo:** Imagine que um usuário altere sua senha em um sistema de login. Após a alteração, ele tenta fazer login novamente imediatamente. Com a garantia RYW, ele poderá acessar a versão mais recente da senha e realizar o login com sucesso, sem ser informado de que a nova senha não foi propagada ainda.
* **Monotonic Reads (MR):** Esta garantia garante que leituras sucessivas feitas dentro de uma mesma sessão veem um conjunto crescente de dados. Em outras palavras, uma leitura subsequente nunca retornará uma versão mais antiga dos dados que foi lida anteriormente. O sistema só permite que a versão dos dados seja mais atual a cada leitura.
* **Exemplo:** Em um aplicativo de calendário online, se o usuário consulta seus compromissos para o dia e depois de algum tempo faz uma nova leitura, ele não verá compromissos antigos, mas sim qualquer atualização ou novos eventos registrados entre as leituras. Isso impede que o usuário veja eventos antigos enquanto novas alterações já ocorreram no sistema.
* **Writes Follow Reads (WFR):** A garantia de "Writes Follow Reads" assegura que todas as escritas realizadas após uma leitura em uma sessão sejam corretamente ordenadas. Ou seja, as escritas feitas dentro da sessão respeitam a ordem de dependência das leituras anteriores e se propagam em todas as cópias do banco de dados de maneira consistente.
* **Exemplo:** Imagine um banco de dados bibliográfico onde um usuário lê uma entrada e a corrige. Se ele corrigir a entrada e fizer uma nova atualização (escrita), a alteração será propagada para todos os servidores, mantendo a ordem de como as leituras e escritas ocorreram, garantindo que todos vejam a versão mais recente da entrada de forma consistente.
* **Monotonic Writes (MW):** Esta garantia assegura que as escritas dentro de uma sessão ocorram em uma sequência ordenada. Ou seja, uma escrita anterior sempre será propagada antes de uma escrita subsequente no banco de dados, garantindo que a ordem das escritas seja preservada, mesmo que as cópias do banco de dados sejam atualizadas de forma assíncrona.
* **Exemplo:** Se um programador faz duas atualizações em um sistema de código-fonte distribuído, uma para a biblioteca e outra para o aplicativo que utiliza essa biblioteca, a garantia MW garante que a atualização do aplicativo só será propagada depois que a atualização da biblioteca for devidamente propagada para todos os servidores. Isso evita problemas como dependências quebradas entre as versões da biblioteca e do aplicativo.

1. **Para garantir que as garantias do Bayou sejam cumpridas, o que os servidores podem fazer?**

**R:** Para garantir que as garantias do Bayou sejam cumpridas, os servidores podem restringir as operações de leitura e escrita a um subconjunto de servidores que estão suficientemente atualizados. Isso significa que, em vez de permitir que qualquer servidor seja usado para uma operação de leitura ou escrita, os servidores devem ser selecionados com base em quão atualizados estão em relação aos dados.

O objetivo dessa restrição é garantir que as garantias, como "Read Your Writes", "Monotonic Reads", "Writes Follow Reads" e "Monotonic Writes", sejam atendidas em cada operação dentro de uma sessão. Isso ocorre porque, em um sistema de dados replicados fracamente consistentes, as operações de escrita são propagadas de maneira preguiçosa (lazy propagation) entre os servidores, e nem todos os servidores estarão necessariamente atualizados ao mesmo tempo.

Ao restringir o conjunto de servidores disponíveis para uma operação, o sistema pode garantir que as operações de leitura e escrita sigam as propriedades de consistência desejadas. No entanto, essa abordagem pode ter um impacto negativo na disponibilidade do sistema, pois a restrição de servidores pode limitar a capacidade de atender a solicitações de outros clientes ou sessões que não estão buscando essas garantias.

Portanto, os servidores precisam equilibrar a consistência com a disponibilidade ao oferecer essas garantias, e as garantias podem ser solicitadas individualmente em uma base por sessão, sem afetar outros aplicativos ou sessões que não exigem essas garantias.

1. **No sistema Bayou concorda com o seguinte? “o termo base de dados não significa nenhum modelo ou organização de dados em particular, nem as técnicas são específicas para qualquer modelo de dados”. Justifique sua resposta.**

**R:** Sim, no sistema Bayou, o termo "base de dados" não significa um modelo ou organização de dados em particular, e as técnicas apresentadas não são específicas para qualquer modelo de dados. Um banco de dados é simplesmente um conjunto de itens de dados, onde um item de dados pode ser qualquer coisa, desde um arquivo convencional até uma tupla em um banco de dados relacional. Além disso, o sistema Bayou não impõe uma estrutura rígida sobre como os dados são organizados, permitindo flexibilidade para diversos modelos de dados. As garantias e técnicas discutidas no contexto de Bayou podem ser aplicadas a diferentes modelos e organizações de dados, sem depender de um formato específico.

1. **Bayou assume que o sistema replicado subjacente fornece consistência eventual e, portanto, inclui mecanismos para garantir duas propriedades. Quais são?**

**R:** O sistema Bayou assume que o sistema replicado subjacente fornece consistência eventual e, portanto, inclui mecanismos para garantir duas propriedades principais:

* **Propagação total:** Garante que cada escrita (Write) será eventualmente recebida por cada servidor.
* **Ordenação consistente:** Garante que todas as escritas não-comutativas sejam aplicadas na mesma ordem em todos os servidores que receberam essas escritas.

Essas propriedades são fundamentais para assegurar que, em sistemas de consistência eventual, os servidores convergem para cópias de banco de dados idênticas na ausência de atualizações.

1. **As implementações das garantias de sessão requerem apenas uma pequena cooperação dos servidores que processam operações de leitura e escrita. Especificamente, quais informações um servidor deve estar disposto a fornecer?**

**R:** As implementações das garantias de sessão requerem que o servidor esteja disposto a fornecer as seguintes informações:

* O identificador único (WID) atribuído a uma nova escrita (Write).
* O conjunto de WIDs para as escritas que são relevantes para uma leitura (Read) específica.
* O conjunto de WIDs para todas as escritas no banco de dados do servidor.

Essas informações permitem que o gerenciador de sessão controle e valide as garantias de sessão, como "Read Your Writes" e "Monotonic Reads".

1. **Para cada sessão, o gestor de sessão mantém dois conjuntos de WIDs; O que são esses conjuntos?**

**R:** Os dois conjuntos de WIDs mantidos pelo gestor de sessão são:

* **read-set:** Este conjunto contém os WIDs das escritas (Writes) que são relevantes para as leituras (Reads) realizadas na sessão.
* **write-set:** Este conjunto contém os WIDs das escritas realizadas dentro da sessão.

1. **Em relação ao suporte para “Read your Writes”, existem duas etapas básicas. Quais são? Onde essas verificações podem ser feitas?**

**R:** As duas etapas básicas para fornecer o suporte para a garantia “Read Your Writes” são:

* **Adicionar o WID ao write-set**: Sempre que uma escrita (Write) é aceita por um servidor, o WID atribuído a essa escrita é adicionado ao write-set da sessão.
* **Verificar se o write-set é um subconjunto do DB(S,t):** Antes de cada leitura (Read) ao servidor S no tempo t, o gestor de sessão deve garantir que o write-set é um subconjunto de DB(S,t) (o banco de dados do servidor no tempo t). Essa verificação pode ser feita de duas maneiras:
* **No servidor:** O write-set pode ser passado para o servidor, que realizará a verificação.
* **No cliente:** O cliente pode recuperar a lista de WIDs do servidor e realizar a verificação.

1. **Em relação ao suporte para “Leituras Monotónicas”, existem duas etapas básicas. Quais são?**

**R:** As duas etapas básicas para fornecer suporte à garantia de "Leituras Monotônicas" (Monotonic Reads) são:

* Verificar que o read-set é um subconjunto de DB(S,t) antes de cada operação de leitura no servidor S no tempo t.
* Adicionar os WIDs de cada Write relevante (RelevantWrites(S,t,R)) à sessão após cada leitura R no servidor S.

Essas etapas garantem que as leituras de um cliente sempre retornarão dados consistentes de acordo com as leituras anteriores feitas pela mesma sessão.

1. **Em relação ao suporte para “Escritas Segue Leituras”, existem duas etapas básicas. Quais são?**

**R:** As duas etapas básicas para fornecer suporte à garantia de "Escritas Seguem Leituras" (Writes Follow Reads) são:

* Adicionar os WIDs relevantes ao read-set: Cada leitura R no servidor S no tempo t resulta na adição dos WIDs de todas as escritas relevantes (RelevantWrites(S,t,R)) ao read-set da sessão.
* Verificar que o read-set é um subconjunto de DB(S,t) antes de cada operação de escrita no servidor S no tempo t.

Essas etapas garantem que, após uma leitura, uma escrita subsequente só será aceita em um servidor se as leituras anteriores feitas pela sessão forem refletidas no estado do servidor.

1. **Em relação ao suporte para “Escritas Monotónicas”, existem duas etapas básicas. Quais são?**

**R:** As duas etapas básicas para fornecer suporte à garantia de "Escritas Monotônicas" (Monotonic Writes) são:

* **Verificar que a write-set está incluída em DB(S,t):** Antes de aceitar uma escrita no servidor S no tempo t, o banco de dados do servidor (DB(S,t)) deve incluir a write-set da sessão.
* **Adicionar o WID da escrita à write-set:** Sempre que uma escrita é aceita por um servidor, seu WID é adicionado à write-set da sessão.

Essas etapas garantem que as escritas de uma sessão sejam processadas em ordem crescente, respeitando a sequência das operações realizadas.

1. **O que é um vector de versão?**

**R:** Um vector de versão é uma estrutura de dados utilizada em sistemas distribuídos para rastrear o histórico de modificações em uma base de dados replicada, permitindo identificar e resolver conflitos de escrita. Ele é composto por um conjunto de pares <servidor, relógio>, onde:

* **Servidor:** é um identificador único para uma instância de um servidor que mantém uma cópia da base de dados replicada.
* **Relógio:** é um valor de um relógio lógico que cresce de forma monotônica a cada operação de escrita realizada pelo servidor. Esse relógio pode ser, por exemplo, um relógio Lamport, um relógio em tempo real ou simplesmente um contador.

A principal função do vector de versão é garantir que o servidor que mantém uma base de dados replicada saiba quais modificações (escritas) foram aplicadas e quais ainda precisam ser incorporadas. Ele é utilizado para verificar se um servidor tem todas as modificações de outros servidores ou se precisa atualizar sua base de dados.

Além disso, o vector de versão é utilizado para fornecer garantias de consistência, como as garantias de Monotonic Reads (Leitura Monotônica), Read Your Writes (Ler suas Escritas) e outras, que são necessárias em sistemas distribuídos que mantêm várias cópias de dados.

1. **Qual vector de versão é mantido em cada servidor?**

**R:** Em cada servidor, é mantido um vector de versão que registra o estado das versões das escritas que o servidor recebeu. Este vector de versão é uma sequência de pares <servidor, relógio>, onde:

* **Servidor:** Identificador único para cada servidor que contém uma cópia da base de dados replicada.
* **Relógio:** Um valor associado ao relógio lógico do servidor, que é um contador que aumenta a cada escrita aceita pelo servidor. O relógio pode ser um contador simples, um Lamport clock ou outro tipo de relógio lógico.

O vector de versão de um servidor mantém a seguinte invariante: se o servidor tem o par <S, c> no seu vector de versão, isso significa que ele recebeu todas as escritas que foram atribuídas a um identificador de escrita (WID) pelo servidor S até o tempo lógico c no relógio do servidor S.

Esse vector de versão é atualizado durante o processo de anti-entropy (sincronização entre servidores) para refletir com precisão o conjunto de escritas que o servidor incorporou em sua base de dados.

1. **Como um conjunto de WIDs pode ser substituído por vectores de versão?**

**R:** Um conjunto de WIDs (identificadores de escrita) pode ser substituído por vectores de versão de forma compacta, simplificando o gerenciamento e a verificação de escritas entre servidores. A seguir estão as regras para a substituição de um conjunto de WIDs por vectores de versão:

* **Obtenção de um vector de versão para um conjunto de WIDs:**
* Para representar um conjunto de WIDs, chamado de `Ws`, o servidor cria um vector de versão `V`. Cada entrada no vector de versão corresponde a um servidor distinto. O valor de `V[S]` (onde `S` é o identificador de um servidor) é o tempo lógico (relógio) mais recente de um WID atribuído ao servidor `S` no conjunto `Ws`.
* Caso não haja WIDs originados pelo servidor `S` no conjunto `Ws`, o valor de `V[S]` será definido como 0.
* **União de dois conjuntos de WIDs:**
* Para obter o vector de versão `V` que representa a união de dois conjuntos de WIDs, `Ws1` e `Ws2`, primeiro obtemos os vectores de versão `V1` e `V2` para `Ws1` e `Ws2`, respectivamente.
* Em seguida, para cada servidor `S`, definimos `V[S] = MAX(V1[S], V2[S])`, ou seja, para cada servidor, o valor do relógio em `V` será o maior entre o valor de `V1[S]` e `V2[S]`.
* **Verificação de subset de WIDs:**
* Para verificar se o conjunto de WIDs `Ws1` é um subconjunto do conjunto `Ws2`, primeiro obtemos os vectores de versão `V1` e `V2` para `Ws1` e `Ws2`, respectivamente.
* Em seguida, verificamos se `V2` domina `V1`, ou seja, se para cada servidor `S`, o valor de `V2[S]` é maior ou igual ao valor de `V1[S]`. A relação de domínio é definida como `V2[S] >= V1[S]` para todo servidor `S`.

1. **Descreva como um cliente pode encontrar um servidor adequado?**

**R:** Para encontrar um servidor adequado, o gerente de sessão deve verificar se um ou ambos os vetores de sessão são dominados pelo vetor de versão do servidor. Quais vetores de sessão são verificados dependem da operação que está sendo realizada e das garantias fornecidas durante a sessão.

Os servidores retornam um vetor de versão juntamente com os resultados de leitura para indicar os Writes relevantes. No entanto, os servidores podem ter dificuldades para calcular o conjunto de Writes relevantes, especialmente em consultas complexas, como as SQL. Para esses casos, o servidor pode simplesmente retornar seu vetor de versão atual como uma estimativa grosseira dos Writes relevantes, o que não viola as garantias de Monotonic Reads ou Writes Follow Reads, mas pode causar o gerente de sessão a ser excessivamente conservador ao escolher servidores aceitáveis.

Além disso, para melhorar a performance, as verificações para encontrar um servidor adequado podem ser amortizadas ao longo de várias operações dentro de uma sessão. O servidor contatado anteriormente sempre será uma escolha aceitável para realizar a próxima operação de leitura ou escrita. As verificações podem ser ignoradas enquanto o gerente de sessão "ficar" com o servidor atual, e só será necessária uma comparação do vetor de versão do servidor com os vetores de sessão quando o gerente de sessão mudar para outro servidor. Para facilitar a localização de um servidor suficientemente atualizado, o gerente de sessão pode armazenar em cache os vetores de versão de vários servidores.