Introdução

- Conjunto de operações sobre objetos a serem executadas como uma unidade indivisível pelos servidores que estão executando esses objetos.

  Conjunto de operações realizadas sobre um conjunto de objetos executadas como uma unidade indivisível mesmo que em vários servidores.

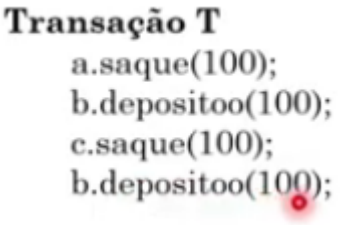
  Ou seja, todas as operações ou nenhuma deve ser concluída (**Atomicidade**).
- Sincronização simples (sem transações)
  - A diretiva synchronized do java como forma de sincronização

  Permite que determinado método só possa ser executado por uma thread por vez.
  - Operações atômicas

  Ou todas são executadas ou nenhuma.
  - Os métodos Java wait e notify a fim de garantir sincronização entre threads.

  Garante que uma thread aguarde a execução de outra e depois entre em execução..

Transações



- Propriedades ACID
  - Atomicidade

  Ou todas operações são executadas ou nenhuma.
  - Consistência

  Uma transação leva o sistema de um estado consistente para outro.
  - Isolamento

  Uma transação não deve sofrer interferência de outras.
  - Durabilidade

  Os dados gerados por uma transação devem ser guardados em meios persistentes.
- Transações em série resolveriam o problema!

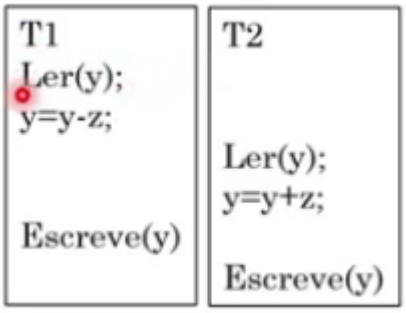
  Não podem haver condições de corridas em algumas operações para garantir as propriedades ACID.

  Contudo a execução em série (não concomitante) de TODAS as operações torna o sistema com mau desempenho, assim apenas as que levam a

inconsistências não podem ser executadas simultaneamente.

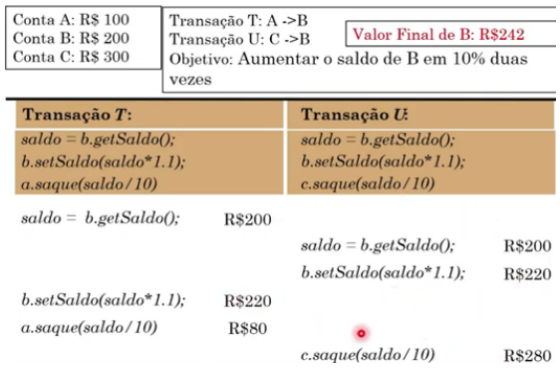
CONTROLE DE CONCORRÊNCIA

- Atualização Perdida
  - Duas transações acessam o mesmo dado de forma intercalada, gerando assim um valor incorreto.
  - Exemplo:



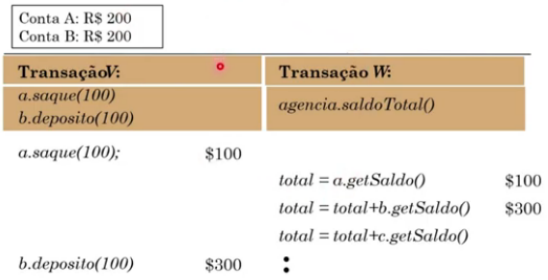
T1 lê o valor atualizado por T2 e não por T1, assim houve uma atualização perdida.

Problema da Atualização Perdida



Recuperações inconsistentes

Transação acessa os dados antes de outro concluir sua atualização.



W deveria ter sido executada apenas depois de V.

CONTROLE DE CONCORRÊNCIA

- Equivalência serial
  Entre o conjunto de operações executando de maneira concorrente, o **resultado final deve ser o mesmo de quando as operações são executadas serialmente.**
  - Interposição serialmente equivalente
  - Evita recuperações inconsistentes
- Operações conflitantes
  Para se chegar a equivalência serial deve-se detectar as operações que podem levar a inconsistências.
  - Operações que seus efeitos combinados dependem da ordem em que são executadas.

Operações de diferentes transações		Conflito	Motivo
leitura	leitura	Não	Porque o efeito de duas operações de leitura não depende da ordem em que elas são executadas
leitura	escrita	Sim	Porque o efeito de uma operação de leitura e de uma operação de escrita depende da ordem de sua execução
escrita	escrita	Sim	Porque o efeito de duas operações de escrita depende da ordem de sua execução

A partir do momento que se tem escrita, já deve-se ter cuidado com a ordem das execuções das operações.

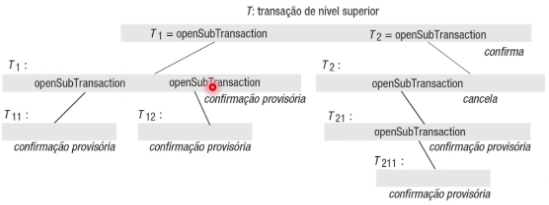
Equivalência serial de T e U

Transação T:		Transação U:	
saldo = b.getSaldo()		saldo = b.getSaldo()	
b.setSaldo(saldo*1.1)		b.setSaldo(saldo*1.1)	
a.saque(saldo/10)		c.saque(saldo/10)	
saldo = b.getSaldo()	$200		
b.setSaldo(saldo*1.1)	$220		
		saldo = b.getSaldo()	$220
		b.setSaldo(saldo*1.1)	$242
a.saque(saldo/10)	$80		
		c.saque(saldo/10)	$278

Apenas depois das escritas podem ser feitas as leituras.

TRANSAÇÕES ANINHADAS

- Transações dentro de transações;
  Uma transação é fragmentada em sub transações.
- Transação de nível superior é a transação mais externa em um conjunto de transações.
- As outras são sub transações.
  A transação de nível superior tem suas transações subordinadas. Criando uma hierarquia entre sub transações. Assim a transação T depende do resultado de todas ou da maioria das sub transações.



TRANSAÇÕES ANINHADAS

- Sub Transações de mesmo nível podem ser executadas concorrentemente, mas os acessos a objetos compartilhados deve ser feito em série.
- Quando uma sub transação é cancelada, a transação ascendente pode escolher uma subtratação alternativa.
  Se uma subtransação for cancelada não obrigatoriamente leva ao cancelamento da transação ascendente. Pode-se apenas criar outra subtransação.
  - Exemplo: Envio de email para uma lista de usuários
- Transação plana
  Cenário mais estático, ou tudo é executado ou tudo é cancelado.
  - Todo o seu trabalho é feito no mesmo nível.

- **Não é possível efetivar ou cancelar partes dela.**

TRANSAÇÕES ANINHADAS-VANTAGENS

- Subtransações de um mesmo nível podem ser executadas concorrentemente.
  Diferente das transações planas.
- Subtransações podem ser confirmadas ou canceladas independentemente.
  Também diferente das transações planas.

Ou seja, há uma concorrência maior e consequentemente um desempenho maior.

REGRAS DAS TRANSAÇÕES ANINHADAS

- Uma transação só é confirmada se suas descendentes tiverem sido concluídas.
  O commit só acontece quando as transações descendentes forem concluídas **(realizadas ou canceladas).**
- Quando uma subtransação é concluída, ela é confirmada provisoriamente ou cancelada.
  Provisoriamente: quando terminou seu trabalho mas ainda precisa de confirmação das outras transações. Ou quando foi cancelada.
- Quando uma transação é cancelada, todas as suas subtransações são canceladas.
  Se a transação ascendente for cancelada as transações descendentes também são canceladas.
- Quando uma subtransação é cancelada, a transação ascendente pode decidir se vai ser cancelada ou não.
  A transação de nível superior decide se a subtransação cancelada vai interferir numa transação ascendente.
- Se uma transação é confirmada, todas as suas subtransações confirmadas provisoriamente podem ser confirmadas, desde que nenhuma ascendente sua tenha sido cancelada.

- T - transferência de 100 reais de A para B.
  - T - B.deposito(100)

    - T"- A.saque(100)
- T' e T" são subtransações de T.
- Se T" é confirmada provisoriamente e T' é cancelada, T deve ser cancelada e portanto T" também será cancelada.

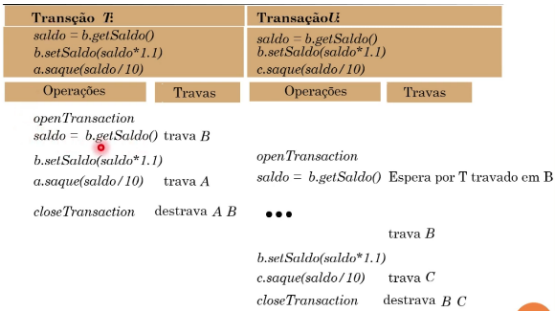
Travas

- Permite o controle do acesso serial a dados compartilhados por transações.
  Impede operações delicadas concorrentes. Garante acesso exclusivo a recursos compartilhados, semelhante aos semáforos.

- Compatibilidade de travas
  Quando uma trava é travada pode-se bloquear um conjunto de operações.

Para um objeto		*Trava solicitada*	
		leitura	*escrita*
Trava já alocada	*nenhum*	OK	OK
	leitura	OK	espera
	escrita	espera	espera

Transações com Travas Exclusivas



Travas

A possibilidade de impasses

Transação *T*		Transação *U*	
Operações	Travas	Operações	Travas
a.deposit(100);	trava de escrita em *A*		
		b.deposit(200)	trava de escrita em B
b.withdraw(100)			
...	espera pela trava de *U* em *B*	*a.withdraw(200);*	espera pela trava de T em A
...		...	
...		...	

Quando uma transação espera infinitamente uma pela outra.

Controle de Concorrência Otimista

Uma das mais utilizadas hoje em dia. Visto que o padrão é que não ocorram condições de corridas, o controle de concorrência otimista trata como uma exceção e não uma regra.

- Inconvenientes das travas:
    - A sobrecarga das travas mesmo considerando que só serão necessárias no pior caso.

  - Uso de travas pode causar impasse.
  - Travas devem ser mantidas até o final da transação, o que pode reduzir significativamente a concorrência.
    Manter uma trava até o fim de uma transação reduz a concorrência, atrapalhando no desempenho.

- Otimista: **A probabilidade de transações de dois clientes acessarem o mesmo objeto é baixa.**

Controle de Concorrência Otimista

- Transações prosseguem como se não houvesse possibilidade de conflito.
- Ao tentar encerrar a transação, se surgir conflito, a mesma é encerrada e precisa ser reiniciada.
  É recomeçado o processo da transação.

  A ideia é deixar tudo ser executado, e, apenas em caso de conflito, é encerrada a transação e o processo é reiniciado.

- Fases de uma transação
  A transação passa por 3 etapas:
  - **Trabalho**

  Momento de execução das operações sobre (uma cópia) dos objetos.
  - **Validação**

  Verifica-se se existem conflitos (condições de corridas).
  - **Atualização**

  No caso de não haverem condições de corridas os dados são atualizados no meio persistente.

Ordenação por carimbo de tempo

- Cada transação recebe um timestamp.
  A hora de início da transação é atribuída a ela. Existe um mesmo timestamp para a transação, para as operações da transação e para todos os dados alterados por essa transação.
  - Cada operação recebe o mesmo timestamp.

- Cada dado possui o timestamp da transação que

realizou a última operação sobre ele.
- As requisições das transações são ordenadas pelos seus timestamps.

Ordenação por carimbo de tempo

Regras:

- Requisição de escrita: é válida se o objeto foi lido ou escrito pela última vez por transações de timestamps anteriores.
  Só é possível escrever se o objeto tiver sido lido ou escrito com timestamp anterior a ele.
- Requisição de leitura: é válida se o objeto foi escrito pela última vez por transações de timestamps anteriores.
  Só é possível escrever se o objeto tiver sido escrito com timestamp anterior a ele.

Ordenação por carimbo de tempo

- Exemplo de leitura:
  Uma transação T solicita read(T,x)
    - Se $TS(T) < TS_{WR}(x)$
      - T é abortada!
- Exemplo de escrita:
  Uma transação T solicita write(T,x)
    - Se $TS(T) < TS_{RD}(x)$
      - T é abortada!
    - Se $TS(T) > TS_{RD}(x)$
      - A escrita é processada.

APLICAÇÕES ATUAIS

- Utilizam controle de concorrência otimista
- Exemplos
  - Dropbox: Granularidade a nível de arquivo.
    No dropbox caso duas escritas sejam realizadas "ao mesmo tempo" apenas a primeira é levada em consideração. Contudo mesmo a rejeitada fica registrada como uma nova cópia em conflito.

    Além disso, há um controle de versão para caso de recuperação.
  - Google Apps: Granularidade mais fina. A nível de célula (planilha) e de caracteres (editor).

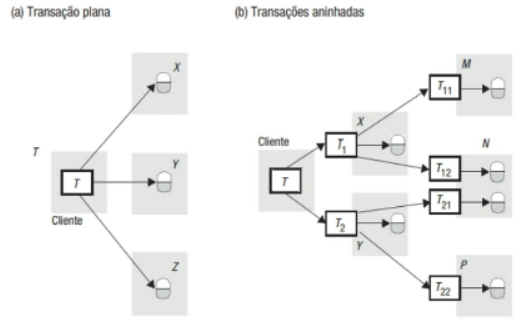
O primeiro usuário que acessa a célula tem domínio dela enquanto estiver escrevendo.

- Wikipedia: A primeira escrita em uma página é a que prevalece. O usuário é informado de conflitos e deve resolver.

A primeira escrita também é a que vale e em caso de conflito o usuário deve corrigir.

TRANSAÇÕES DISTRIBUÍDAS

Transações planas ou aninhadas que acessam objetos gerenciados por vários servidores.



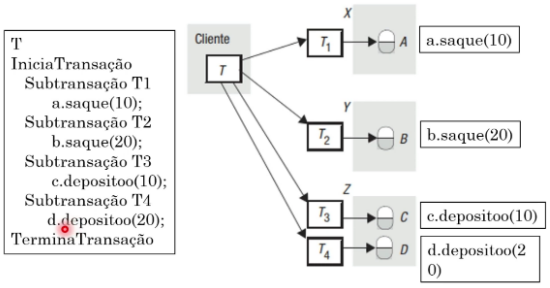
planas: acesso sequência

aninhadas: divisão de uma transação em sub transações.

TIPOS DE TRANSAÇÕES

- Transação plana:
  - **Acessa os objetos dos servidores em sequência.**
  - **Quando utiliza travas, uma transação só pode estar esperando um objeto por vez.**
- Transação aninhada:
  - **Uma transação pode abrir subtransações.**
  - **Subtransações de mesmo nível podem ser executadas em paralelo.**

TRANSAÇÃO ANINHADA



Coordenador de uma transação

- Cada transação distribuída possui um servidor que coordena todos os participantes.
- O coordenador atribui um ID para a transação e envia para o cliente que a iniciou. O ID deve ser exclusivo.
- O coordenador é responsável por confirmar ou cancelar a transação.
- Cada participante rastreia seus objetos envolvidos na transação.

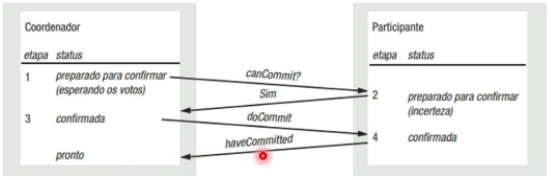
  Cada participante busca os respectivos objetos

necessários para execução de uma subtransação.

- Durante a transação, o coordenador mantém a referência de todos os participantes
- Cada participante possui uma referência do coordenador.
  Os participantes podem enviar mensagens para o coordenador e vice-versa.

Protocolo de confirmação de duas fases

- Fase 1: Coordenador pergunta a todos os participantes se estão preparados para confirmar.
- Fase 2: Coordenador diz para os participantes confirmarem ou cancelar a transação.
  Depende das respostas da fase 1.



É enviada uma mensagem **"canCommit?"**, e os participantes respondem se sim ou não.

Baseado nas respostas o coordenador resolve continuar ou cancelar.

Em caso de continuar é feito um **"doCommit"** e os participantes preparados vão realizar suas ações e depois enviar um **"haveCommited"** para confirmar que concluíram.

Em caso de falhas o coordenador manda um **"doAbort"** para os participantes e cancela a transação e suas operações.

Protocolo de confirmação de duas fases

Fase 1 (fase de votação):

1. O coordenador envia uma requisição *canCommit?* para cada um dos participantes da transação.
2. Quando um participante recebe uma requisição *canCommit?*, ele responde com seu voto (*Sim* ou *Não*) para o coordenador. Antes de votar em *Sim*, ele se prepara para confirmar, salvando objetos no armazenamento permanente. Se o voto for *Não*, o participante cancelará imediatamente.

Fase 2 (conclusão de acordo com o resultado do voto):

3. O coordenador reúne os votos (incluindo o seu próprio).
   (a) Se não houver falhas, e todos os votos forem *Sim*, o coordenador decide confirmar a transação e envia uma requisição *doCommit* para cada um dos participantes.
   (b) Caso contrário, o coordenador decide cancelar a transação e envia requisições *doAbort* para todos os participantes que votaram *Sim*.
4. Os participantes que votaram *Sim* estão esperando por uma requisição *doCommit* ou *doAbort* do coordenador. Quando um participante recebe uma dessas mensagens, ele age correspondentemente e, em caso de confirmação, faz uma chamada de *haveCommitted* para o coordenador.

Protocolo de confirmação de duas fases para transações aninhadas

- Transação de nível superior
  Transação que originou todas as subtransações.
- Sub Transações – começam depois de sua ascendente e terminam antes dela.
- Uma sub transação, ao terminar, pode ser confirmada provisoriamente ou cancelada.

confirmada provisoriamente: conseguiu realizar todas suas operações mas ainda não guardou de fato os dados em meio persistente.

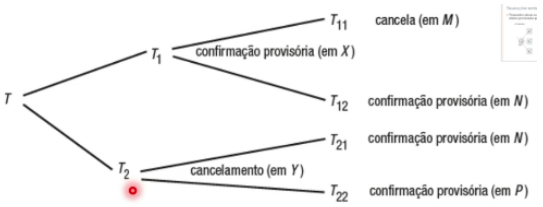
- Servidores das sub transações confirmadas provisoriamente expressam intenção de confirmar

  É enviada uma mensagem para o coordenador para confirmação das subtransações.
- Operações no coordenador de transações aninhadas
  - openSubTransaction(trans)

    Abre uma nova subtransação para "trans" e retorna o ID dessa nova subtransação.
  - getStatus(trans)

    Solicita ao coordenador o estado de uma determinada transação.

Protocolo de confirmação de duas fases para transações aninhadas



Coordenador da transação	Transações descendentes	Participante	Lista de confirmações provisórias	Lista de cancelamentos
T	T_1, T_2	sim	T_1, T_{12}	T_{11}, T_2
T_1	T_{11}, T_{12}	sim	T_1, T_{12}	T_{11}
T_2	T_{21}, T_{22}	não (cancelada)		T_2
T_{11}		não (cancelada)		T_{11}
T_{12}, T_{21}		T_{12} mas não T_{21}*	T_{21}, T_{12}	
T_{22}		não (ascendente cancelada)	T_{22}	

* T_{21}: ascendente foi cancelada

Cada transação possui um coordenador, se duas transações estão em um mesmo servidor então elas possuem o mesmo coordenador.

ESTRATÉGIAS

Na primeira estratégia é obedecida a hierarquia da estrutura de transações aninhadas, na segunda o coordenador pergunta diretamente para cada coordenador das transações na lista de confirmadas provisoriamente..

- Protocolo hierárquico de confirmação de duas fases

  Cada ascendente pergunta aos seus descendentes "canCommit".
  - **Cada participante faz um canCommit para cada uma de suas subtransações diretas. Cada participante**

reúne as respostas de seus descendentes, antes de responder ao seu ascendente.

- Protocolo plano de confirmação de duas fases
  O coordenador da transação de nível superior manda "canCommit" para todos os participantes na lista de confirmação provisória.
    - **Coordenador da transação de nível superior envia mensagem canCommit para todos os coordenadores de todas as subtransações na lista de confirmações provisórias.**

    - Cada participante, ao receber o canCommit, verifica se possui alguma transação provisoriamente confirmada que sejam descendentes de trans
        - Se não tem ancestral cancelada, prepare-se para confirmar e envia Sim para o coordenador.
        - Caso contrário, deve ter havido falha e envia Não para o coordenador.

CONTROLE DE CONCORRÊNCIA EM TRANSAÇÕES DISTRIBUÍDAS

- Travamento
  A ideia é bloquear determinado objeto e só desbloqueá-lo quando concluir sua tarefa.
    - **As travas são mantidas localmente por cada servidor de maneira independente.**

  Um servidor não sabe das travas de outro.
    - Pode levar a impasses distribuídos

  Impasses mais complexos, referências a objetos ou travas em máquinas diferentes.

- Carimbo de tempo
    - **Cada transação recebe um carimbo de tempo único globalmente.**

  O carimbo de tempo é composto de duas partes: o instante de tempo local

associado ao ID do servidor onde o objeto está.

A ordem é baseada no carimbo de tempo e o ID do servidor associado a ele é usado em critério de desempate, o menor ID terá carimbo de tempo menor.

- **O coordenador, em cada servidor que possui objetos que executam operações da transação, recebe o carimbo de tempo da transação**.
- **Os carimbos de tempos dos coordenadores devem ser aproximadamente sincronizados.**

Para garantir a sincronia é usado NTP por exemplo. As regras de ordenação ainda são usadas.

CONTROLE DE CONCORRÊNCIA EM TRANSAÇÕES DISTRIBUÍDAS

Otimista

- Cada transação é validada antes de ser confirmada.

  O padrão é esperar sempre o melhor, que tudo vai dar certo e não haverá impasses. Segue as etapas de **trabalho**, **validação** e **atualização**.

- Uma transação distribuída é validada por um conjunto de servidores independentes.

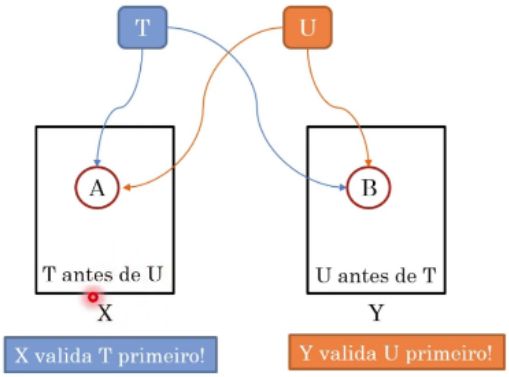
  Cada servidor valida uma transação realizada em um objeto localizado dentro dele de maneira independente, isso pode levar a **impasses**.

- A possibilidade de impasse no processo de validação.
  - Ex: Transações T e U, que acessam os objetos A e B nos servidores X e Y.

- As validações são executadas em paralelo pelos servidores evitando impasse.
  - Risco: diferentes servidores serializarem o mesmo conjunto de transações em ordens diferentes. Ex: T antes de U em X e U antes de T em Y
  - Solução: validação global após a validação local em cada servidor

---

Exemplo



A ordem de acesso aos objetos podem ser diferentes e caso os servidores comecem a validar ao mesmo tempo, pode ocorrer de um esperar infinitamente o outro terminar sua validação.

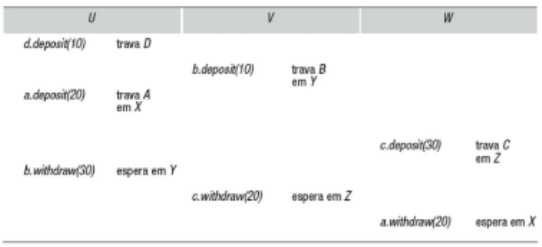
IMPASSES DISTRIBUÍDOS

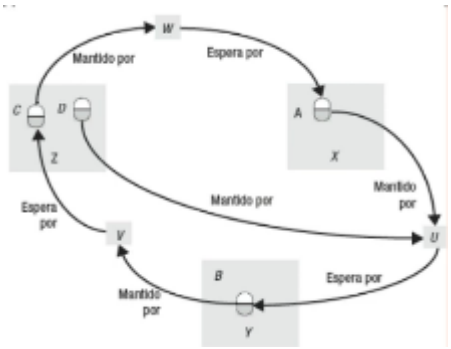
- Detecção de impasses através de ciclos em um grafo de espera.

  Se houver um ciclo então há um impasse.
- Em um SD, cada servidor tem seu grafo local e um grafo global deve ser construído a partir destes.

  Cada servidor possui seu grafo local de espera e um coordenador une todos os grafos em um grafo global para detectar os ciclos.
- Detecção centralizada de impasses distribuídos

  Desvantagens: Não existe tolerância a falhas pois é usado um único coordenador central e não existe escalabilidade também pela centralização pois um único servidor é responsável pela criação do grafo. Por fim, o custo de atualização e envio dos grafos é alto.





RECUPERAÇÃO DE TRANSAÇÕES

- Está relacionada a durabilidade e atomicidade da falha.

  Os dados da transação devem ser persistidos em discos. Em caso de falha é necessário ter todos os dados necessários para se recuperar e voltar ao estado de antes da transação.
- Tarefas do gerenciador de recuperação:

- **Salvar objetos das transações confirmadas em meio permanente.**

Para garantir a durabilidade.

- **Restaura objetos após uma falha**

Para a atomicidade de falhas.

- Reorganizar o arquivo de recuperação visando um melhor desempenho

É necessário avaliar constantemente o arquivo de recuperações para tornar uma tarefa rápida.

- Recuperar espaço de armazenamento.

Quando um dado não é mais necessário o espaço deve ser liberado.

- Lista de intenções guarda todas as operações de uma transação (lista das referências e os valores dos objetos alterados).

Na lista de intenções (persistida no arquivo de recuperação) há todas as informações de uma transação.

RECUPERAÇÃO DE TRANSAÇÕES

- Log
  - Registro contendo o histórico de todas as transações realizadas por um servidor.

O sistema de recuperações salva as operações realizadas em um arquivo de log além de:

- Consiste nos valores dos objetos, status e lista de intenções das transações.

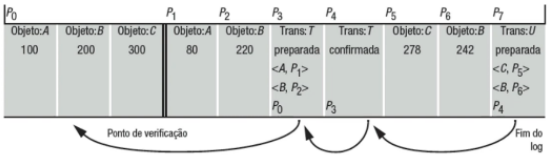
status: confirmada, cancelada ou confirmada provisoriamente.

- A ordem no log reflete a ordem dos acontecimentos.

Porque a medida que as transações ocorrem elas são registradas no log.

- É um instantâneo dos valores presentes no servidor, seguido do histórico das transações posteriores.

Instantâneo: é como uma fotografia dos valores de um objeto em determinado momento.



P_0, p_3 e p_4 são instantâneos?

Equivalência Serial de T e U

Transação *T*:		Transação *U*:	
saldo = b.getSaldo() *b.setSaldo(saldo*1.1)* *a.saque(saldo/10)*		*saldo = b.getSaldo()* *b.setSaldo(saldo*1.1)* *c.saque(saldo/10)*	
saldo = b.getSaldo()	$200		
*b.setSaldo(saldo*1.1)*	$220		
		saldo = b.getSaldo()	$220
		*b.setSaldo(saldo*1.1)*	$242
a.saque(saldo/10)	$80		
		c.saque(saldo/10)	$278