

MC714

Sistemas Distribuídos

2º semestre, 2013

Consistência e replicação

Consistência e replicação

- Dados são replicados para aprimorar confiabilidade ou melhorar desempenho.
- Problema: manter réplicas consistentes.
 - Cópia é atualizada → assegurar que as outras também sejam
- Ex. confiabilidade: sistema de arquivos replicados
 - Um cai, outro continua atendendo solicitações
 - Melhor proteção contra dados corrompidos
 - 3 cópias: toda operação de leitura e escrita feita em cada cópia; confia na maioria.

Consistência e replicação

- Replicação para desempenho
 - Importante quando SD precisa ser ampliado em (1) quantidade e (2) em área.
 - (1) Número cada vez maior de processos precisa acessar dados de um único servidor → replicar
 - (2) Cópias de dados próximas do usuário diminui tempo de acesso → melhora desempenho → pode aumentar uso de banda.
 - Métrica de eficiência pode ser diferente dependendo do ponto de vista.

Consistência e replicação

- Cópias próximas ao usuário pode melhorar desempenho → resolve problema de escalabilidade.
- Manter cópias atualizadas pode requerer mais largura de banda.
 - Compromisso entre manter cópias atualizadas e uso de banda da rede
- Processo P que acessa uma réplica local N vezes por segundo; réplica atualizada M vezes por segundo.
- Se $N \ll M$ (razão acesso/atualização) for muito baixa, muitas atualizações nunca serão usadas
 - Desperdício de banda

Consistência e replicação

- Intuitivamente: cópias consistentes = cópias sempre iguais
- Leitura em qualquer cópia sempre retornará o mesmo resultado.
- Atualização em uma cópia → atualização em todas as cópias antes de qualquer operação subsequente
 - Atualização em todas as cópias como uma única operação atômica → transação
 - Difícil concluir transação amplamente distribuída de forma rápida.
 - Sincronização global para decisão de sincronizar réplicas pode ser muito cara.

Consistência e replicação

- Dilema:
 - problemas de escalabilidade podem ser amenizados por replicação e cache → melhor desempenho.
 - Manter consistência pode ter desempenho ruim.
- Solução em muitos casos: relaxar restrições de consistência.
 - Evitar sincronização global (instantânea).
 - Preço: cópias que nem sempre são iguais em todos os lugares.
 - Abrandamento da consistência depende de padrões de acesso e atualização e da finalidade dos dados.

Consistência centrada em dados

Consistência centrada em dados

- Consistência no contexto de operações de leitura/escrita em dados compartilhados
 - Memória compartilhada distribuída
 - Banco de dados distribuído
 - Sistema de arquivos distribuído
- Termo genérico: depósito de dados
 - Pode ser distribuído fisicamente por várias máquinas
 - Cada processo tem uma cópia local (ou próxima) do depósito inteiro
 - Escrita propagada para outras cópias.

Consistência centrada em dados

- Não há regra geral para abrandar consistência
 - Depende muito da aplicação
- Pode-se definir três eixos independentes para definir inconsistências (Yu e Vahdat 2004).
 - Desvio numérico (absoluto ou relativo)
 - Desvio em idade
 - Desvio em relação à ordenação de operações de atualização

Consistência centrada em dados

- Não há regra geral para abrandar consistência
 - Depende muito da aplicação
- Pode-se definir três eixos independentes para definir inconsistências (Yu e Vahdat 2004).
 - Desvio numérico (absoluto ou relativo)
 - Desvio em idade
 - Desvio em relação à ordenação de operações de atualização

Unidades de consistência

- Consistency unit – conit
 - Especifica a unidade segundo a qual a consistência deve ser medida (1 registro, 1 formulário, 1 tabela...)
- Fig. 107.
 - $t, i \rightarrow$ operação na réplica i em seu tempo lógico t
 - Réplica **A** recebe **5,B: $x \leftarrow x + 2$**
 - Réplica **A** tem 3 operações de atualização: **8,A; 12,A; 14,A**
 - Desvio de ordenação: 3
 - Relógio vetorial de **A** vai para (15,5)
 - Desvio numérico: (1,5) \rightarrow uma operação de **B** que **A** ainda não viu (**10,B**)
 - Réplica **B** tem duas operações de atualização: **5,B; 10,B**
 - Desvio de ordenação: 2
 - Relógio vetorial: (0,11)
 - Desvio numérico: (3,6)

Unidades de consistência

- Compromisso entre conits de granularidade fina e grossa
- Se conit é um banco de dados completo, atualizações são agregadas para todos os dados
 - Pode levar réplicas a estado inconsistente mais cedo.
- Conits pequenas podem retardar propagação de atualizações
 - Mas pode aumentar custo de gerência
- Implementação necessita:
 - Protocolos de consistência
 - Especificar requisitos de consistência para aplicações

Unidades de consistência

- São necessárias interfaces de programação simples para especificar consistência
- Ex.: biblioteca onde conit é declarada com atualização de dados.

```
AffectsConit(ConitQ, 1, 1);
```

```
Inclua mensagem m na fila Q;
```

```
-----
```

```
DependsOnConit(ConitQ, 4, 0, 60); //desvNum, desvOrd,  
                                     //desvTempo
```

```
Leia mensagem m da fila Q;
```

Ordenação consistente de operações

- Consistência sequencial e consistência causal
- Ordenar operações consistentemente em dados compartilhados replicados

Ordenação consistente de operações

- $W_i(x)a \rightarrow$ escrita do processo i no item de dados x com o valor a
- $R_i(x)b \rightarrow$ leitura do item x pelo processo i , retornando b
- Tempo: esquerda para direita
- Fig. 108: $W_1(x)a; R_2(x)NIL; R_2(x)a$

Consistência sequencial

- Consistência sequencial – Lamport 1979
- Memória compartilhada para multiprocessadores
- Depósito de dados é sequencialmente consistente quando satisfaz:

O resultado de qualquer execução é o mesmo que seria se as operações (de leitura e escrita) realizadas por todos os processos no depósito de dados fossem executadas na mesma ordem sequencial e as operações de cada processo individual aparecessem nessa sequencia na ordem especificada por seu programa.

Consistência sequencial

- Ou: quando processos executam concorrentemente em máquinas diferentes, qualquer intercalação válida de operações de escrita e leitura é um comportamento aceitável, mas todos os processos vêem a mesma intercalação de operações.
- Nada é dito sobre o tempo
- Processo vê escritas de todos, mas apenas suas leituras.
- Fig. 109.

Consistência causal

- Consistência causal – Hutto e Ahmad 1990
- Enfraquece consistência sequencial
 - Distinção de eventos que são potencialmente relacionados por causalidade.
- Se evento **b** é causado ou influenciado por evento anterior **a**, a causalidade requer que todos vejam primeiro **a** e depois **b**.

Consistência causal

- BD compartilhado
- P1 escreve item de dados x .
- P2 lê de x , escreve no item de dados y .
- Leitura de x e escrita de y potencialmente relacionadas por causalidade: cálculo de y pode ter dependido de x , escrito por P1.
- Escrita “espontânea” e simultânea de dois itens de dados diferentes, não são relacionados por causalidade
 - operações concorrentes (não relacionadas por causalidade)

Consistência causal

- Depósito de dados consistente por causalidade:

Escritas que são potencialmente relacionadas por causalidade devem ser vistas por todos os processos na mesma ordem. Escritas concorrentes podem ser vistas em ordem diferente em máquinas diferentes.

- Fig. 110. Sequencia permitida em depósito consistente por causalidade, mas proibida quando sequencialmente consistente.
- Fig. 111: (a) proibida por causalidade; (b) permitida por causalidade.

Consistência centrada em cliente

Consistência eventual

- Necessidade de garantia de consistência pode variar.
- Tolerância a inconsistência: consistência eventual.
- Na ausência de atualizações, todas as réplicas convergem na direção a cópias idênticas umas às outras.
- Pode haver problemas se usuário se move: pode ler dado diferente daquele que escreveu.
- Consistência centrada no cliente dá *a um único cliente* uma garantia de consistência de acesso a um depósito de dados.

Leituras monotônicas

- Consistência de leitura monotônica:

Se um processo ler o valor de um item de dados x , qualquer operação de leitura sucessiva de x executada por esse processo sempre retornará o mesmo valor ou um valor mais recente.

Escritas monotônicas

- Consistência de escrita monotônica:

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.

Leia-suas-escritas

- Consistência leia-suas-escritas

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.

Escritas-seguem-leituras

- Consistência escritas-seguem-leituras

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.

Gerenciamento de Réplicas

- Posicionamento do servidor de réplicas
- Replicação e posicionamento de conteúdo
 - Réplicas permanentes
 - Réplicas iniciadas por servidor
 - Réplicas iniciadas por cliente

Tolerância a falha

Tolerância a falha

- Sistema tolerante a falha → sistema confiável
- Disponibilidade: capacidade de um sistema estar pronto para ser usado imediatamente
- Confiabilidade: sistema poder funcionar continuamente sem falha
- Segurança: se um sistema deixar de funcionar corretamente durante um certo tempo, nada de catastrófico acontecerá
- Capacidade de manutenção: facilidade com que um sistema que falhou pode ser consertado.

Tolerância a falha

- Sistema que apresenta defeito: sistema que não pode cumprir suas promessas.
- Erro: parte do estado de um sistema que pode levar a uma falha
- Falha: causa de um erro.
- Falha \rightarrow erro \rightarrow defeito

Tolerância a falha

- Sistemas confiáveis dependem de controle de falhas
 - Evitar, remover, prever falhas.
- Tolerância a falhas: sistema pode prover serviço mesmo na presença de falhas.
- Falhas transientes: ocorrem uma vez e desaparecem.
- Falha intermitente: ocorre e desaparece, reaparece.
- Falha permanente: continua existindo até que o componente faltoso seja substituído.

Modelos de falha

- Em SDs há dependências entre, componentes, servidores e serviços.
 - Falha pode estar localizada em diversos lugares.
- Classificação de falhas.
- Falha por queda: servidor pára prematuramente.
- Falha por omissão: servidor deixa de responder a uma requisição.
- Falha de temporização: resposta fora de um intervalo especificado.
- Falha de resposta: resposta incorreta.
- Falha bizantina: arbitrariedade.

Modelos de falha

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure <i>Receive omission</i> <i>Send omission</i>	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure <i>Value failure</i> <i>State transition failure</i>	A server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Mascaramento por redundância

- Ocultar ocorrência de falhas através do uso de redundância.
 - Redundância de informação
 - Redundância de tempo
 - Redundância física

Processos faltosos

- Exemplo: acordo (consenso) distribuído com falha
- Eleger coordenador, validar ou não transação, repartir tarefas, sincronização...
- Objetivo: que todos os processos que não apresentam falha cheguem a um consenso sobre alguma questão e estabeleçam esse consenso dentro de um número finito de etapas.

Processos faltosos

		Message ordering				Communication delay
		Unordered		Ordered		
Process behavior	Synchronous			X		Bounded
	Asynchronous	X	X	X	X	Unbounded
				X	X	Bounded
				X	X	Unbounded
		Unicast	Multicast	Unicast	Multicast	
		Message transmission				

Circunstâncias sob as quais se pode chegar a acordo distribuído.

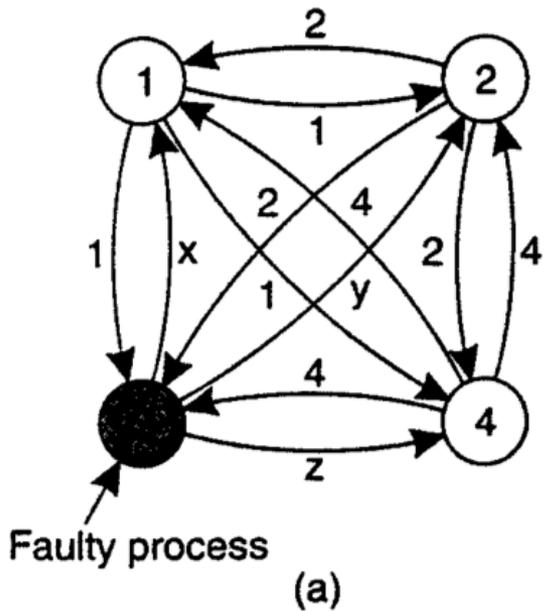
Acordo bizantino

- Lamport 1982.
- Síncrono, unicast, atraso limitado, entrega confiável.
- 4 etapas
- 1: processos enviam valores a todos os processos usando unicast confiável. Faltosos podem enviar qualquer coisa.
- 2: processos reúnem resultados em vetores.
- 3: cada processo transmite seu vetor
- 4: cada processo examina i -ésimo elemento dos vetores. Maioria é colocada no vetor resultado.

Acordo bizantino

- Lamport 1982.
- Síncrono, unicast, atraso limitado, entrega confiável.
- 4 etapas
- 1: processos enviam valores a todos os processos usando unicast confiável. Faltosos podem enviar qualquer coisa.
- 2: processos reúnem resultados em vetores.
- 3: cada processo transmite seu vetor
- 4: cada processo examina i -ésimo elemento dos vetores. Maioria é colocada no vetor resultado.

Acordo bizantino



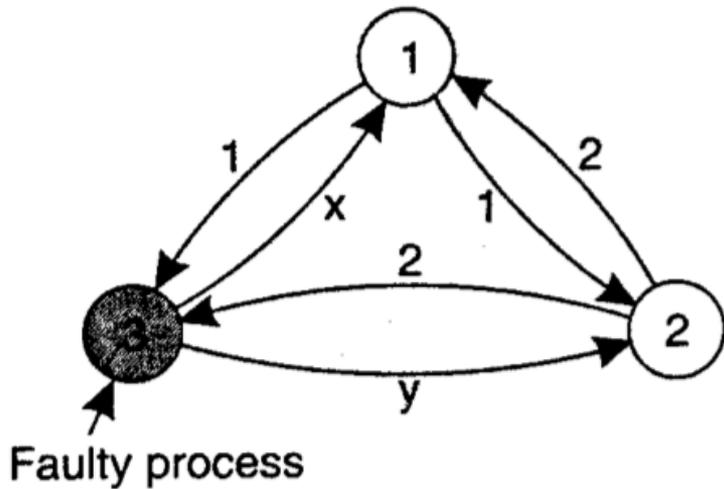
1 Got(1, 2, x, 4)
 2 Got(1, 2, y, 4)
 3 Got(1, 2, 3, 4)
 4 Got(1, 2, z, 4)

(b)

<u>1 Got</u>	<u>2 Got</u>	<u>4 Got</u>
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

Acordo bizantino



(a)

1 Got(1, 2, x)
 2 Got(1, 2, y)
 3 Got(1, 2, 3)

(b)

$\frac{1 \text{ Got}}{(1, 2, y)}$	$\frac{2 \text{ Got}}{(1, 2, x)}$
(a, b, c)	(d, e, f)

(c)

Acordo bizantino

- Se há K processos faltosos, necessários $2k+1$ processos funcionando corretamente para um total de $3k+1$.
 - Acordo só é possível se mais do que dois terços dos processos estiverem funcionando adequadamente.