SSC150 – Sistemas Computacionais Distribuídos

Transações Distribuídas

9ª aula
13/05/10

Profa. Sarita Mazzini Bruschi
sarita@icmc.usp.br

Slides baseados no material de:
Prof. Rodrigo Mello (USP / ICMC)
Prof. Edmilson Marmo Moreira (UNIFEI / IESTI)

Transações Atômicas

- Todas as técnicas estudadas até o momento são de baixo nível e exigem que o programador esteja envolvido com todos os detalhes de exclusão mútua, gerenciamento de regiões críticas, prevenção de *deadlock* e recuperação de falhas
- Deseja-se esconder do programador todos esses detalhes, através de um alto nível de abstração
- Essas abstrações são bastante utilizadas em sistemas distribuídos e são denominadas *Transações Atômicas*, ou simplesmente *transações*

Transações Atômicas

- Um processo anuncia que quer iniciar uma transação com um ou mais processos
- Eles podem executar várias operações por um certo período de tempo e depois o processo que iniciou a transação anuncia que quer que todos os outros confirmem (*commit*) o trabalho feito até então
- Se todos concordarem, o trabalho feito se torna permanente
- Se um ou mais processos se recusaram (ou falharam) a situação é revertida para o estado em que todos estavam antes da transação começar
- Este procedimento é chamado de “tudo ou nada” (*all-or-nothing*) e facilita o desenvolvimento dos programas pelo programador

Transações Atômicas

Exemplo

- Retirada de dinheiro de uma conta bancária e depósito em outra usando um PC com modem
- Operações em dois passos:
  - Retirar(quantia, conta1)
  - Depositar(quantia, conta2)
- Se a conexão cai depois do primeiro passo, mas antes do segundo, teremos como conseqüência o desaparecimento do dinheiro
- O problema é resolvido agrupando os dois passos em uma transação atômica
  - Ou os dois passos são completados ou nenhum deles será executado
  - Se a transação falhar, o sistema restaura o estado inicial

Transações Atômicas
Modelo

- O sistema consiste de um conjunto de processos independentes que podem falhar aleatoriamente
- A comunicação não é confiável, porém técnicas com *timeout* e protocolos de retransmissão podem recuperar mensagens perdidas

Transações Atômicas
Armazenamento Persistente

- O armazenamento possui 3 níveis:
  - Memória RAM: perde as informações quando existem falhas de energia ou de *hardware*
  - Disco: sobrevive a falhas de CPU mas pode ter perda de informação caso ocorra falha no disco, como por exemplo, cabeça do disco danificada
  - Armazenamento Estável: projetado para sobreviver a qualquer situação, exceto calamidades como inundações e terremotos
    - Apresenta alto grau de tolerância à falhas, o que é importante na implementação de transações atômicas
    - Pode ser implementado com um par de discos comuns, onde cada bloco do disco 2 é uma cópia do disco 1
    - Outra solução: RAID em outros níveis

Transações Atômicas
Primitivas

- A programação que utiliza transações exige primitivas especiais que devem ser suportadas pelo sistema operacional ou pelo mecanismo de execução da linguagem usada para implementação:
  - BEGIN_TRANSACTION: marca o início da transação
  - END_TRANSACTION: marca o fim da transação e tenta realizar o *commit*
  - ABORT_TRANSACTION: encerra a transação e restaura os valores anteriores ao início da transação
  - READ: lê dados de um arquivo (ou outro objeto)
  - WRITE: escreve dados em um arquivo (ou outro objeto)

Transações Atômicas
Primitivas

- A lista de primitivas depende do tipo de objeto que está sendo usado na transação
  - Em um sistema de e-mail deveria existir primitivas para enviar, receber e encaminhar e-mails
- BEGIN_TRANSACTION e END_TRANSACTION são usadas para delimitar o escopo de uma transação e as operações entre estas primitivas formam o corpo da transação
- Todo o corpo da transação deve ser executado ou nenhum será executado

Transações Atômicas
Propriedades

- As transações possuem quatro propriedades principais:
  - Atomicidade (*Atomic*): para o ambiente externo, a transação acontece de forma indivisível
  - Consistência (*Consistência*): a transação não viola nenhuma invariante do sistema
  - Isolamento (*Isolated*): transações concorrentes não interferem umas nas outras
  - Durabilidade (*Durable*): uma vez que a transação é realizada (*commit*), a mudança é permanente
- Essas propriedades são conhecidas como ACID

Transações Atômicas
Propriedades

- Consistência (exemplo)

```
BEGIN_TRANSACTION          BEGIN_TRANSACTION          BEGIN_TRANSACTION
  X = 0;                     X = 0;                     X = 0;
  X = X + 1;                 X = X + 2;                 X = X + 3;
END_TRANSACTION            END_TRANSACTION            END_TRANSACTION
```

tempo

Schedule 1	X=0; X=X+1; X=0; X=x+2; X=0; X=X+3;	Legal
Schedule 2	X=0; X=0; X=X+1; X=X+2; X=0; X=X+3;	Legal
Schedule 3	X=0; X=0; X=X+1; X=0; X=X+2; X=X+3;	Ilegal

Transações Atômicas
Implementação

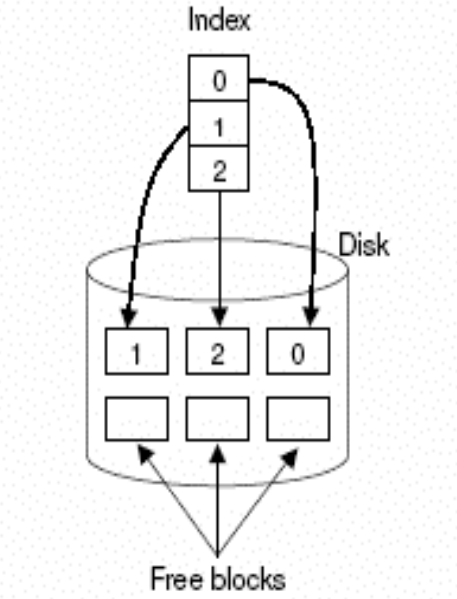
- Espaço de Trabalho (*Workspace*) Privado
  - Quando um processo começa uma transação, ele recebe um espaço privado contendo todos os arquivos (e objetos) que ele tem para acessar
  - Até que uma transação seja encerrada ou abortada, todas as leituras e escritas do processo são realizadas em um *workspace* privado
  - Problema: copiar tudo para uma área privada geralmente é proibitivo

Transações Atômicas
Implementação

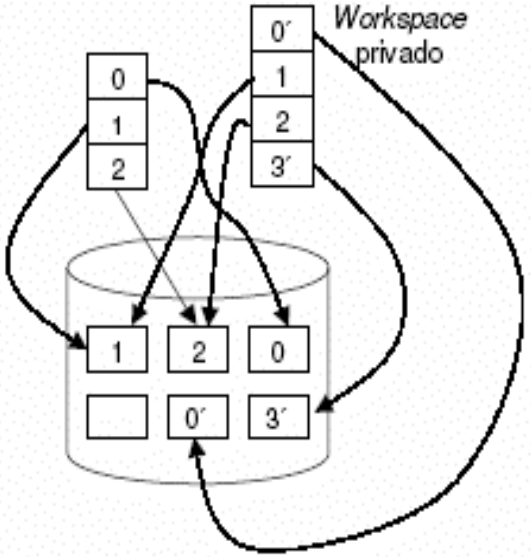
- Espaço de Trabalho (*Workspace*) Privado
  - Otimizações:
    - Quando um processo lê um arquivo não é necessário fazer uma cópia
    - Quando um processo escreve num arquivo, não é necessário copiar o arquivo de entrada, somente o arquivo de índice é copiado para o *workspace* privado

Transações Atômicas
Implementação

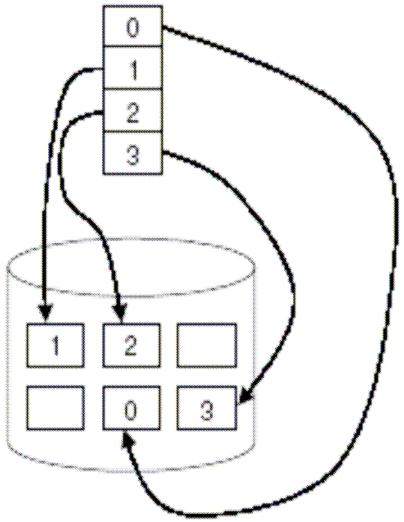
- Espaço de Trabalho (*Workspace*) Privado



O arquivo de índice e os blocos de disco para três blocos de arquivo.



Situação após a transação ter modificado o bloco 0 e adicionado o bloco 3



Situação após o *commit*

Transações Atômicas
Implementação

- Escrevendo um *log* (lista de intenções)
  - Neste caso, os arquivos são realmente modificados, mas antes de qualquer bloco ser modificado, é registrada, em um dispositivo de armazenamento estável, a transação que está fazendo a mudança; que arquivo e bloco estão sendo modificados; e os valores antigo e novo
  - Se a transação não for finalizada (*abort*), o *log* pode ser usado para voltar o sistema ao estado original
  - Começando do fim para o começo, cada registro do *log* é lido e a mudança desfeita (*rollback*)
  - O *log* também pode ser usado para recuperar o sistema de uma falha

Transações Atômicas
Implementação

- Escrevendo um *log* (lista de intenções)

```
X = 0;
Y = 0;
BEGIN_TRANSACTION
  X = X + 1;
  Y = Y + 1;
  X = Y * Y;
END_TRANSACTION
```

(a)

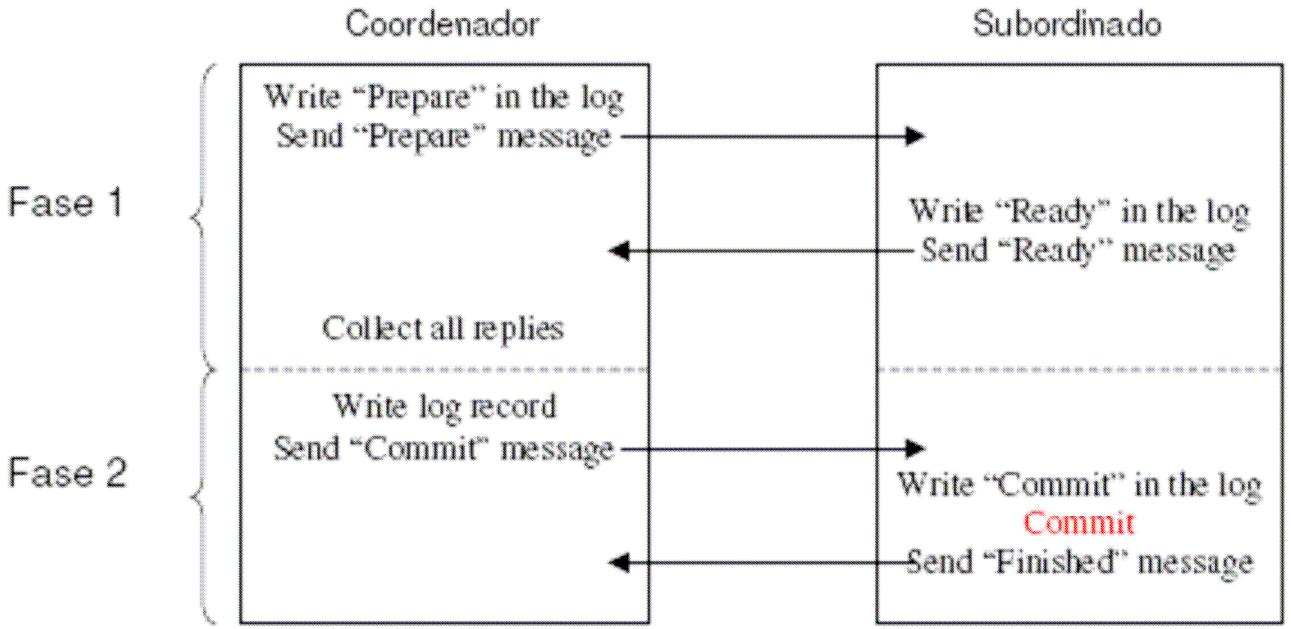
Log	Log	Log
X = 0/1	X = 0/1 Y = 0/2	X = 0/1 Y = 0/2 X = 1/4
(b)	(c)	(d)

Transações Atômicas
Implementação

- *Commit*
  - A ação de encerramento de uma transação deve ser atômica, ou seja, “instantânea” e indivisível
  - Em um sistema distribuído, o *commit* exige cooperação de múltiplos processadores em diferentes máquinas
  - Um protocolo bastante conhecido é o *Two-Phase Commit*

Transações Atômicas
Implementação

- *Two-Phase Commit*



Transações Atômicas
Controle de Concorrência

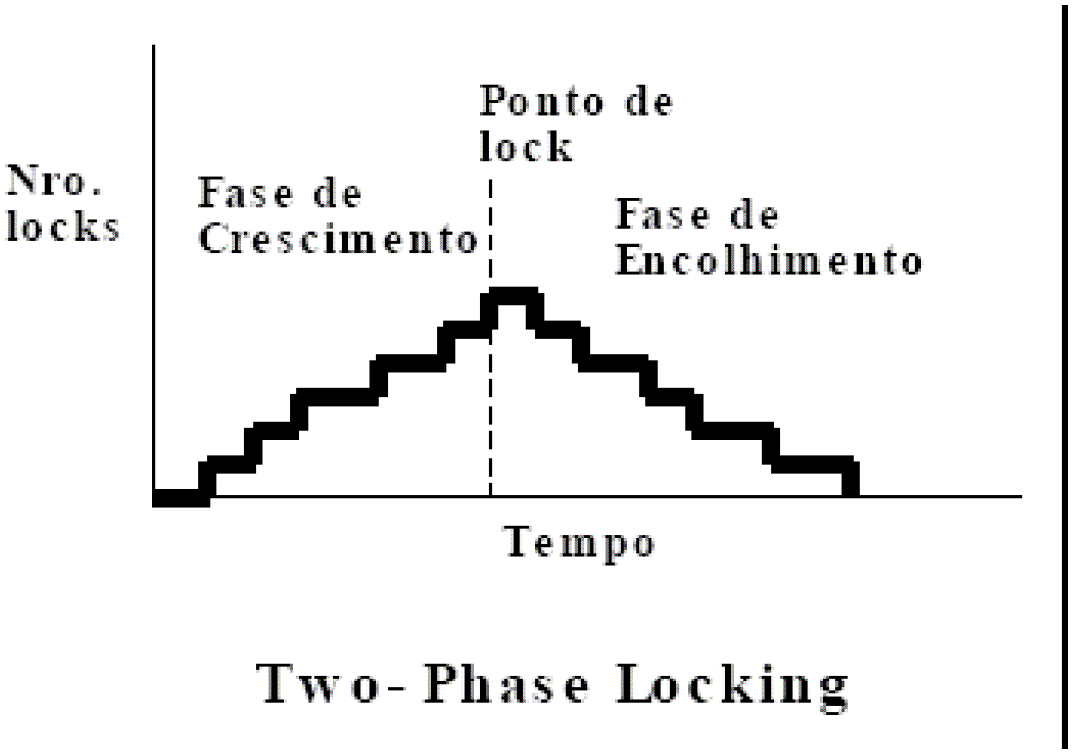
- Quando muitas transações estão sendo executadas simultaneamente em diferentes processos (ou diferentes processadores), algum mecanismo é necessário para garantir a propriedade *Isolated.* Este mecanismo é conhecido como ***concurrency control algorithm*** (algoritmo de controle de concorrência).

Transações Atômicas
Controle de Concorrência

- *Locking*
  - É o mais antigo e mais amplamente algoritmo de controle de concorrência utilizado
  - Quando um processo precisa ler ou escrever em um arquivo (ou qualquer outro recurso) como parte de uma transação, ele primeiro trava o arquivo
  - O gerente de *lock* mantém uma lista dos arquivos travados e rejeita qualquer tentativa de acesso por outros processos
  - Essa operação normalmente é feita pelo sistema e não requer a interferência do programador
  - Adquirir e liberar *locks*, precisamente no instante em que eles são necessários pode levar a inconsistências e deadlock
  - Por isso, a maioria das implementações das transações utiliza o *Two-Phase Locking*

Transações Atômicas
Controle de Concorrência

- *Two-Phase Locking*



Two- Phase Locking

Transações Atômicas
Controle de Concorrência

- Controle Otimmista
  - A idéia desta técnica é bastante simples: vá em frente com a execução e faça o que quiser sem prestar atenção ao que os outros estão fazendo
  - Se tiver algum problema, preocupa-se depois
  - Na confirmação são verificadas todas as outras transações para ver se algum dos seus arquivos foram modificados desde que a transação começou
  - Se houver modificação, a transação é abandonada, caso contrário, é confirmada (*commited*)
  - Permite o máximo de paralelismo
  - Em condições de muita carga no sistema, a probabilidade de falha é bem maior

Transações Atômicas
Controle de Concorrência

- *Timestamp*
  - A cada transação é atribuído um *timestamp* no momento de seu início
  - Usando o algoritmo de Lamport, assegura-se que os *timestamps* são únicos
  - Cada arquivo no sistema tem um *timestamp* de leitura e outro de escrita associados a ele, informando qual transação *commited* foi a última que leu ou escreveu nele
  - Quando um processo acessa um arquivo, o *timestamp* do arquivo será menor do que o da transação. Isto significa que o processamento está acontecendo na ordem certa
  - Caso a ordem esteja incorreta, a transação é abortada

SSC150 – Sistemas Computacionais Distribuídos

Processadores em Sistemas Distribuídos

9ª aula
06/05/10

Profa. Sarita Mazzini Bruschi
sarita@icmc.usp.br

Slides baseados no material de:
Prof. Rodrigo Mello (USP / ICMC)
Prof. Edmilson Marmo Moreira (UNIFEI / IESTI)

Modelos de sistemas

- Os processos são executados em processadores
- Em Sistemas Distribuídos, os processadores podem ser organizados de diversas formas, sendo as principais:
  - Modelo de Estações de Trabalho (Workstation Model)
  - Modelo Banco de Processadores (Processor Pool Model)

Modelo de Estações de Trabalho

- Neste modelo, o sistema consiste de estações de trabalho em um edifício conectadas por uma LAN de alta velocidade
- Existem dois tipos de estações de trabalho:
  - com disco (*diskful workstation*)
  - sem disco (*diskless workstation*)
    - Há a necessidade de se implementar um servidor remoto de arquivos para o sistema de arquivos

Modelo de Estações de Trabalho

- Vantagens do modelo sem disco:
  - Ter um grande número de workstations equipadas com discos lentos é mais caro do que ter um ou dois servidores de arquivos com discos rápidos e com grande capacidade de armazenamento;
  - Facilidade de extensão, sendo facilmente adaptável às necessidades dos usuários;
  - Fácil manutenção, pois a atualização de um software não precisa ser feita em um grande número de máquinas;
  - Provê simetria e flexibilidade, pois todos os arquivos estão em um único lugar

Modelo de Estações de Trabalho

- Desvantagens do modelo sem disco
  - Servidores centralizados são pontos de falha e gargalos
  - Servidores compartilhados favorecem a utilização indevida (*hackers*)
  - Os processadores podem ser subutilizados, uma vez que várias estações costumam ficar desocupadas

Modelo de Estações de Trabalho

- O modelo de estações de trabalho com disco podem ser utilizadas das seguintes formas:
  - Paginação e arquivos temporários;
  - Paginação, arquivos temporários e binários;
  - Paginação, arquivos temporários, binários e caching de arquivos;
  - Sistema de arquivo local completo (baixa carga na rede)

Modelo de Estações de Trabalho

Utilização do disco	Vantagens	Desvantagens
Sem disco	Baixo custo; fácil manutenção de hardware e software; simetria e flexibilidade	Utilização pesada da rede, servidores de arquivos podem se tornar gargalos
Paginação e arquivos temporários	Reduz o overhead na rede sobre o modelo sem disco	Alto custo, devido ao grande número de discos necessários
Paginação e arquivos temporários e binários	Reduz ainda mais a carga na rede	Alto custo; complexidade adicional para atualização dos binários
Paginação, arquivos temporários, binários e cache	Baixa carga na rede; reduz também a carga nos servidores de arquivos	Alto custo, problemas de coerência de cache
Sistema de arquivos local	Quase nenhuma carga na rede; elimina a necessidade de servidoes de arquivos	Perda de transparência

Modelo de Estações de Trabalho

- Vantagens do modelo de Estações de Trabalho
  - Fácil de entender
  - Quantidade fixa de processamento dedicado, garantindo tempo de resposta
  - Programas gráficos sofisticados podem ser utilizados, pois possuem acesso direto à tela
  - Cada usuário tem um certo grau de autonomia e pode alocar os recursos das estações conforme for precisando
  - Discos locais tornam possível a continuidade do trabalho mesmo se o servidor de arquivos falhar
- Problema:
  - Estações ociosas

Modelo de Estações de Trabalho

- Utilizando estações ociosas:
  - Primeira tentativa: utilização do comando *rsh*
    - *rsh máquina comando*
    - Executa *comando* na *máquina* especificada
    - Problemas:
      - O usuário precisa achar as máquinas ociosas;
      - O ambiente da máquina remota pode ser diferente do ambiente local

Modelo de Estações de Trabalho

- Utilizando estações ociosas
  - A pesquisa de estações ociosas está baseada em três questões:
    1. Como uma estação ociosa é encontrada?
    2. Como um processo remoto pode executar de maneira transparente?
    3. O que acontece se o dono da estação ociosa voltar?

Modelo de Estações de Trabalho
1. Ociosidade

- O que é uma estação ociosa?
  - a que não tem ninguém “logado”
  - a que a carga de trabalho é baixa
    - o usuário pode logar e não tocar no teclado por horas
- Os algoritmos para localizar estações ociosas são divididas em duas categorias:
  - Dirigidos pelo servidor
  - Dirigidos pelo cliente

Modelo de Estações de Trabalho
1. Ociosidade

- Algoritmo dirigido pelo servidor:
  - Quanto uma estação está ociosa, ela anuncia sua disponibilidade, colocando o seu nome, endereço de rede, etc., em uma base de dados centralizada (registro)
  - Quando um usuário deseja executar um comando em uma estação ociosa, simplesmente digita: **remote comando**
  - O programa **remote** procura no registro uma estação ociosa mais apropriada
  - Outra possibilidade de se anunciar que uma estação está ociosa: coloca-se na rede uma mensagem de *broadcast* e cada máquina mantém uma cópia do registro
  - Problema: pode-se ocorrer condições de disputa

Modelo de Estações de Trabalho
1. Ociosidade

- Algoritmo dirigido pelo cliente
  - Quando o comando **remote** é invocado, faz-se um *broadcast* da requisição dizendo que programa deseja executar, quanto de memória será necessário, se serão necessárias operações de ponto-flutuante, etc.
  - Quando uma mensagem de resposta volta, o comando **remote** pega uma e inicia a operação

Modelo de Estações de Trabalho
2. Transparência

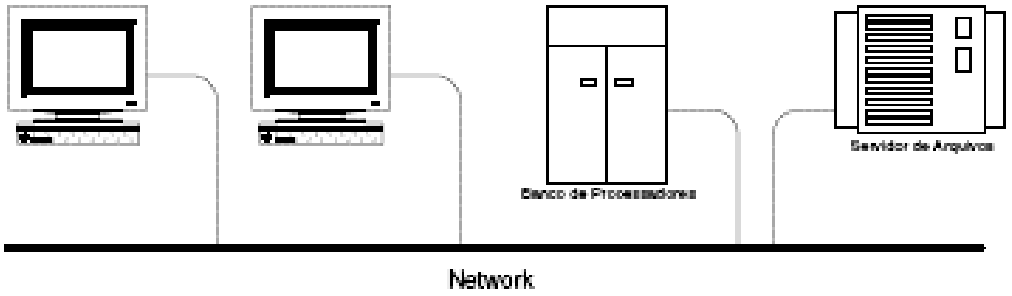
- Assim que a estação ociosa foi encontrada, é necessário agora que o processo seja movido para essa estação:
  - O processo precisa ter a mesma visão do sistema de arquivo, diretório de trabalho, variáveis de ambiente, etc.
  - Algumas chamadas de sistema precisam ser enviadas de volta para a máquina local, como por exemplo, leitura do teclado

Modelo de Estações de Trabalho
3. Volta do dono da estação

- Quando o usuário dono da estação ociosa volta, pode-se:
  - Não fazer nada, porém isso vai contra a idéia de se ter uma estação de trabalho pessoal;
  - Matar o processo intruso, perdendo o trabalho já realizado;
  - Alertar o processo de que será morto, dando tempo para realizar algumas tarefas de modo a não perder o trabalho
  - Migrar o processo para outra máquina, o que não é fácil pois precisa mover todas as estruturas de dados e pode-se ter arquivos abertos

Modelo Banco de Processadores

- Neste modelo, o sistema é constituído por um gabinete com vários processadores em uma sala central, os quais podem ser alocados dinamicamente para os usuários
- No lugar de estações de trabalho, tem-se terminais gráficos de alta performance



Modelo Banco de Processadores

- Vantagens:
  - Alta capacidade de executar processamento numérico intensivo
  - Adequado para o desenvolvimento de aplicações paralelas
  - Melhor aproveitamento de recursos
- Desvantagens:
  - Não recomendado para atividades interativas
  - Escalabilidade mais difícil

Modelo Híbrido

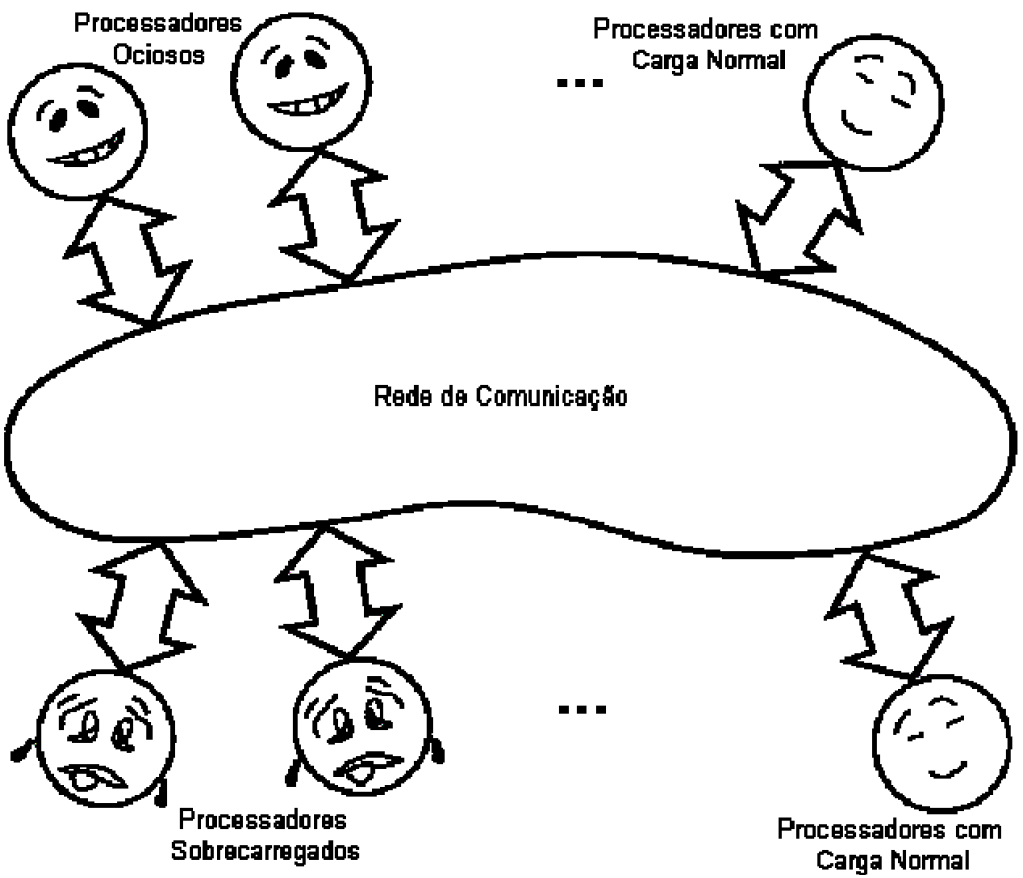
- Neste modelo, cada usuário tem uma estação de trabalho pessoal e tem também um banco de processadores
- Possui as vantagens dos dois modelos
- Desvantagem de ser mais caro

Alocação dos Processadores

- Um sistema distribuído pode estar organizado como uma coleção de estações, um banco de processadores ou alguma forma híbrida
- Em qualquer um dos casos é necessário decidir qual processo deve ser executado em qual máquina:
  - Modelo estação de trabalho: quando deve-se executar um processo localmente ou procurar uma estação ociosa
  - Modelo banco de processadores: uma nova decisão precisa ser tomada para cada novo processo

Alocação dos Processadores

- O problema consiste em como distribuir (ou escalonar) os processos entre os elementos de processamento visando atingir alguns objetivos, tais como: minimizar o tempo de execução e os atrasos causados pela comunicação e/ou maximizar a utilização dos recursos do sistema



Alocação dos Processadores

- As estratégias de alocação podem ser classificadas em:
  - Não-migratórias
    - O processo é criado e é tomada a decisão de onde colocá-lo
    - Este ficará no processador designado até o final
  - Migratórias
    - Um processo pode ser movido mesmo que já tenha começado a sua execução
    - Permite um maior balanceamento de carga no sistema

Alocação dos Processadores
Tópicos de Projeto

- Decisões que devem ser tomadas nos algoritmos de alocação de processadores:
  - Algoritmo determinístico x heurístico
  - Algoritmo centralizado x distribuído
  - Algoritmo ótimo x sub-ótimo
  - Algoritmo local x global
  - Algoritmo inicializado pelo enviador x inicializado pelo receptor

Alocação dos Processadores
Tópicos de Projeto

- Determinístico x Heurístico
  - Determinístico: quando é conhecido antecipadamente tudo sobre o comportamento do processo. Pode-se tentar todas as possibilidades de alocação e escolher a melhor
  - Heurístico: quando a carga do sistema é imprevisível, podendo variar drasticamente.
- Centralizado x Distribuído
  - Coletar todas as informações em um único lugar permite uma melhor decisão, mas é menos confiável e pode carregar excessivamente a máquina central
- Ótimo x Sub-ótimo
  - Soluções ótimas podem ser obtidas para sistemas centralizados ou distribuídos, mas são mais custosas que as sub-ótimas, pois coletam mais informações
- A maioria dos algoritmos utilizam heurísticas, são distribuídos e sub-ótimos

Alocação dos Processadores
Tópicos de Projeto

- Local x Global (Política de transferência)
  - Quando um processo é criado, é decidido se ele pode ser executado na máquina que foi gerado ou não
  - Se esta máquina estiver muito carregada, o novo processo é transferido para outro lugar
  - A questão é se a transferência é baseada inteiramente na informação local. Caso a carga da máquina esteja abaixo de um certo limiar, pode-se optar por manter o novo processo, caso contrário, se livrar dele
  - Outra abordagem é coletar informação global a respeito da carga em outras máquinas antes de decidir onde o processo vai ser executado

Alocação dos Processadores
Tópicos de Projeto

- Algoritmo inicializado pelo enviador x inicializado pelo receptor (Política de locação)
  - Uma vez que foi decidido executar o processo em outra máquina, a política de locação tem que determinar para onde enviar esse processo
  - A política de locação obviamente não pode ser local, ela necessita de informação sobre a carga do sistema
  - Tem-se duas abordagens:
    - O processo que envia começa a troca de informação;
    - O receptor inicia a troca de informação

Alocação dos Processadores
Questões de implementação

- Todos os algoritmos assumem que as máquinas sabem sua própria carga e podem dizer se estão ociosas ou sobrecarregadas
- Medir a carga não é uma tarefa simples. Pode-se considerar:
  - Número de processos em cada máquina
  - Contar somente os processos que estão sendo executados ou prontos
  - Fração que a CPU está ocupada

Alocação dos Processadores
Questões de implementação

- Overhead para coletar as informações e migrar os processos
  - Deve-se considerar: tempo de CPU, memória utilizada, *bandwith* da rede
- Complexidade do software considerado para coletar as medidas, além de desempenho, corretude e robustez
- Eager *et al.* (1986) estudou 3 algoritmos

Alocação dos Processadores
Questões de implementação

- Algoritmo 1
  - Pega uma máquina aleatoriamente e envia um novo processo para ela.
  - Se a máquina receptora estiver sobrecarregada, ela envia o processo para outra máquina aleatoriamente.
  - O procedimento continua até que alguém aceite o processo ou um contador seja excedido

Alocação dos Processadores
Questões de implementação

- Algoritmo 2
  - Pega uma máquina aleatoriamente e envia uma mensagem para ela perguntando se está sobrecarregada ou não
  - Se não estiver, ela ganha o novo processo; caso contrário, outra máquina é sorteada
  - O procedimento continua até que uma máquina é encontrada ou um contador é excedido. Neste caso o processo permanece onde foi criado

Alocação dos Processadores
Questões de implementação

- Algoritmo 3
  - Pergunta para *k* máquinas as suas cargas
  - O processo é enviado para a máquina com a menor carga

- O algoritmo 3 tem o melhor desempenho, mas o ganho sobre o algoritmo 2 é pequeno

SSC150 – Sistemas Computacionais Distribuídos

Deadlock

9ª aula
06/05/10

Profa. Sarita Mazzini Bruschi
sarita@icmc.usp.br

Slides baseados no material de:
Prof. Edmilson Marmo Moreira (UNIFEI / IESTI)

Deadlock

- Um conjunto de processos está em *deadlock* se cada processo do conjunto está aguardando por um evento que somente um outro processo do mesmo conjunto pode causar
- Exemplo:
  - dois processos querendo imprimir um arquivo grande armazenado num DVD
    - O processo A requisita permissão para utilizar a impressora e recebe autorização
    - O processo B, em seguida, requisita utilizar o DVD e também consegue permissão
    - O processo A pede para utilizar o DVD mas a requisição é negada até que B libere
    - O processo B pede para utilizar a impressora mas a requisição é negada até que A libere

Deadlock

- Recursos
  - Um recurso é qualquer dispositivo de hardware ou informação que pode ser utilizado por somente um único processo em um determinado instante de tempo
  - Podem ser:
    - Preemptivos
      - Podem ser retirados dos processos sem prejuízos
      - Exemplo: memória
    - Não-preemptivos
      - Não pode ser retirado do processo sem causar falhas no processamento
  - Normalmente, tem-se *deadlocks* com recursos não-preemptíveis

Deadlocks
Condições para que ocorra Deadlock

- Exclusão mútua
  - Cada recurso só pode estar associado a um processo em um determinado instante
- Posse e espera
  - Processo de posse de recursos podem requisitar por novos recursos
- Não-preempção
  - Recursos previamente garantidos não podem ser retirados de um processo
- Espera circular
  - Deve existir uma cadeia circular de 2 ou mais processos, cada um aguardando por recursos que estão alocados para membros de cada cadeia

Deadlocks
Abordagens para o tratamento de dealocks

- Algoritmo do Avestruz
- Prevenção de deadlock
- Impedimento de deadlock
- Detecção de deadlock
- Recuperação de deadlock

Deadlocks em Sistemas Distribuídos

- Deadlocks em sistemas distribuídos são similares aos que ocorrem em sistemas com um único processador
- São difíceis de prevenir, impedir, ou mesmo detectar
- Pode-se dividir em dois tipos:
  - Deadlock de comunicação
  - Deadlock de recursos

Deadlocks em Sistemas Distribuídos

- Deadlock de comunicação:
  - Ocorre quando, por exemplo, um processo A está tentando enviar uma mensagem para um processo B que, por sua vez, está tentando enviar uma mensagem para um processo C, que está tentando enviar uma mensagem para A. Neste caso, poderia ocorrer deadlock se não houvesse *buffers* disponíveis

Deadlocks em Sistemas Distribuídos

- Deadlock de recursos:
  - Ocorre quando processos estão disputando acessos exclusivos à dispositivos de I/O, arquivos, etc.
- Os deadlocks de comunicação e recursos podem ser tratados de maneira única, pois canais de comunicação e/ou *buffers* são também recursos

Deadlocks em Sistemas Distribuídos

- Todas as estratégias de tratar deadlock em sistemas monoprocessados podem ser aplicadas em sistemas distribuídos
- O algoritmo do Avestruz é uma boa abordagem para a maioria dos sistemas gerais, tais como automação de escritórios, controle de processos, etc. Esses sistemas não possuem mecanismos para tratamento de deadlock, embora aplicações individuais, tais como banco de dados distribuídos possam implementar os seus próprios mecanismos

Deadlocks em Sistemas Distribuídos

- A detecção e recuperação também são abordagens bastante utilizadas, principalmente porque prevenir e evitar são abordagens mais complexas
- A prevenção também é possível, embora mais complexa do que as formas de prevenção utilizadas em sistemas com um único processador
- As abordagens de impedimento, em geral, não são utilizadas em sist. distribuídos, principalmente porque é muito difícil saber quais recursos que cada processo irá eventualmente precisar (algoritmo do banqueiro).

Detecção de Deadlocks em Sistemas Distribuídos – Detecção

- Quando um deadlock ocorre em um SO convencional, a forma de resolver o problema é “matando” um ou mais processos envolvidos no deadlock
- Quando um deadlock é detectado em um sistema baseado em transações atômicas, ele é resolvido abortando uma ou mais transações

Detecção de Deadlocks em Sistemas Distribuídos – Detecção

- Quando uma transação é abortada porque ela contribuiu para um deadlock, o sistema é primeiramente restaurado para o estado em que ele se encontrava antes da transação começar
- Neste ponto, a transação pode recomeçar e com um pouco de sorte pode ter sucesso na segunda vez
- A diferença é que as conseqüências de abortar um processo em sistemas que utilizam transações atômicas são menos severas do que em sistemas que não utilizam transações atômicas

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Centralizada

- Primeira tentativa: utilizar um algoritmo de detecção centralizado
- Embora cada máquina possua um grafo de alocação de recursos para os seus processos e recursos, um coordenador central mantém um grafo de recursos para as entradas no sistema, que é a união de todos os grafos individuais

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Centralizada

- Ao contrário do sistema centralizado, onde todas as informações estão automaticamente disponíveis em um mesmo local, em um sistema distribuído estas informações devem ser enviadas explicitamente
- Cada máquina mantém o grafo para o seus processos e recursos e diversas possibilidades podem ser implementadas objetivando a atualização do grafo centralizado

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Centralizada

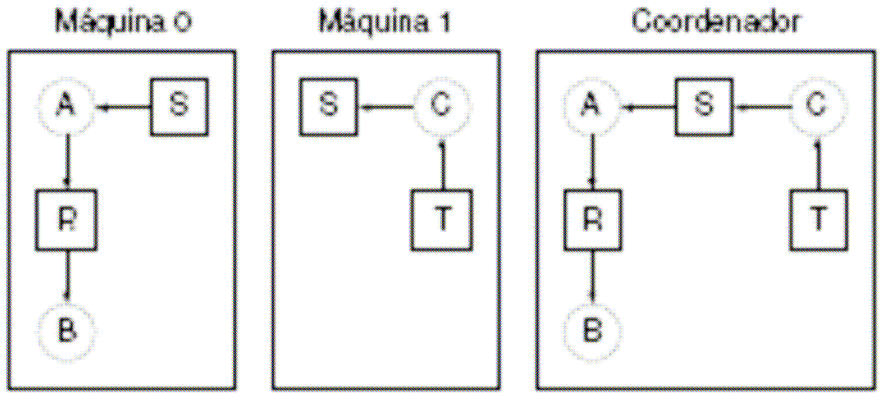
- Primeira:
  - Sempre que um arco é adicionado ou removido do grafo de alocação de recursos, uma mensagem pode ser enviada para o coordenador providenciar a atualização
- Segunda:
  - Periodicamente, todo processo pode enviar uma lista de arcos adicionados ou removidos desde a última atualização
  - Menos mensagens que a primeira abordagem
- Terceira:
  - O coordenador pede informação quando ele precisar

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Centralizada

- Infelizmente, nenhuma das abordagens trabalha bem
- Exemplo:
  - Sistema com os processos A e B executando na máquina 0 e o processo C executando na máquina 1
  - Três recursos: R, S e T

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Centralizada

- Inicialmente:
  - O processo A mantém o recurso S e deseja R;
  - O processo B mantém o recurso R;
  - O processo C detém o recurso T e deseja o recurso S

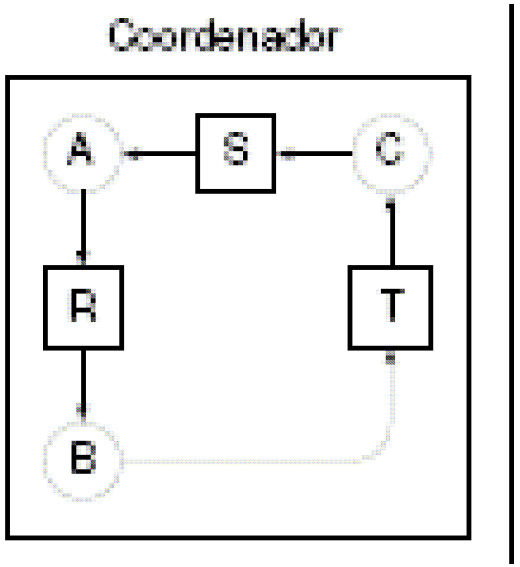


Detecção de Deadlocks em Sistemas Distribuídos – Detecção Centralizada

- Execução segura, pois não existe ciclo:
  - Assim que B encerrar, o processo A obtém R e pode terminar a execução, liberando o recurso S para o processo C
- Após um tempo, B libera R e requisita T, uma operação perfeitamente segura
- A máquina 0 envia uma mensagem para o coordenador anunciando a liberação de R e a máquina 1 envia uma mensagem para o coordenador anunciando que agora B está esperando por T

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Centralizada

- Infelizmente, a mensagem da máquina 1 chega primeiro, levando o coordenador a construir o grafo abaixo, levando a um *falso deadlock*



Detecção de Deadlocks em Sistemas Distribuídos – Detecção Centralizada

- Solução para esse problema: utilizar o algoritmo de Lamport para prover um relógio global:
  - Uma vez que a mensagem da máquina 1 foi disparada por uma mensagem da máquina 0 (a requisição de T por B), a mensagem da máquina 1 irá possuir um *timestamp* maior do que o *timestamp* da máquina 0 para o coordenador

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Centralizada

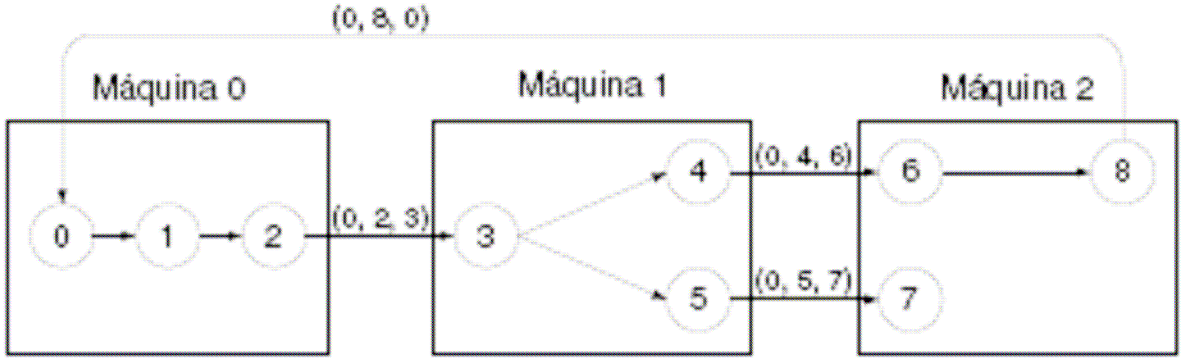
- Quando o coordenador recebe a mensagem da máquina 1 que sinaliza um *deadlock* suspeito, ele pode enviar uma mensagem para todas as máquinas no sistema dizendo: “eu acabei de receber uma mensagem com *timestamp* T que sinaliza um *deadlock*. Se alguém possuir uma mensagem para mim com um *timestamp* menor, favor enviá-la imediatamente.”
- Quando todas as máquinas responderem, positiva ou negativamente, o coordenador irá verificar que a aresta de *R* para *B* foi removida, e concluirá que o estado do sistema ainda é seguro.

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Distribuída

- Existem vários algoritmos
- Um deles é o algoritmo de Chandy-Misra-Hass, em homenagem aos seus criadores
- Neste algoritmos, os processos estão autorizados a solicitar vários recursos de cada vez, ao invés de um
- Com esta alteração, os processos podem aguardar por dois ou mais recursos simultaneamente

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Distribuída

- Modificação no grafo, onde somente os processos são apresentados
  - Cada arco passa por um recurso, mas este foi omitido para simplificar
  - O processo 3 está esperando por 2 recursos, um em posse do processo 4 e outro em posse do processo 5
  - Alguns processos estão aguardando por recursos locais (processo 1, máquina 0), enquanto outros estão aguardando por recursos alocados em outras máquinas (processo 2, máquina 0)



Detecção de Deadlocks em Sistemas Distribuídos – Detecção Distribuída

- O algoritmo de Chandy-Misra-Hass é invocado quando um processo deve aguardar por algum recurso, como por exemplo, o processo 0 bloqueando o processo 1
- Uma mensagem *probe* (investigação) é gerada e enviada para o processo (ou processos) que mantêm os recursos necessários

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Distribuída

- A mensagem consiste de 3 números:
  - O processo que está bloqueado,
  - O processo que está enviando a mensagem
  - O processo para quem a mensagem se destina
- A mensagem inicial do processo 0 para o processo 1 contém a tupla (0,0,1)
- Quando uma mensagem chega, o receptor verifica se está aguardando por algum recurso
- Se estiver, a mensagem é atualizada, mantendo-se o primeiro campo, mas alterando o segundo para o número do processo receptor e o terceiro para o número do processo que o processo receptor está aguardando

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Distribuída

- Se o processo aguarda vários recursos, ele envia mensagens para cada um dos processos que detém os recursos desejados
- Se a mensagem retornar ao processo que iniciou o procedimento, o ciclo estará detectado e, consequentemente, o deadlock

Detecção de Deadlocks em Sistemas Distribuídos – Detecção Distribuída

- Várias formas de quebrar o deadlock:
  - O processo que iniciou a detecção comete "suicídio"
    - Problema: vários processos iniciarem a investigação ao mesmo tempo, e todos cometerem o suicídio, quando não seria necessário
  - Acrescentar a sua identidade na mensagem de investigação e depois verificar qual possui a maior identificação e matá-lo

Detecção de Deadlocks em Sistemas Distribuídos – Prevenção

- A prevenção de deadlock consiste em projetar cuidadosamente o sistema para que os estes sejam, estruturalmente, impossíveis de acontecer
- Algumas técnicas:
  - Exigir que os processos manipulem somente um recurso por vez;
  - Exigir que os processos requisitem todos os seus recursos no início do processamento;
  - Fazer com que os processos liberem todos os recursos quando eles solicitarem por um novo
- Estas não funcionam muito bem

Detecção de Deadlocks em Sistemas Distribuídos – Prevenção

- Uma técnica que pode funcionar é ordenar os recursos disponíveis no sistema e exigir que os processos requisitem os recursos estritamente na ordem crescente
- Isso significa que um processo nunca poderá manter um recurso e solicitar um outro com menor número
- Em sistemas distribuídos com tempo global e transações atômicas, outras duas técnicas são possíveis

Detecção de Deadlocks em Sistemas Distribuídos – Prevenção

- Ambas são baseadas na idéia de associar cada transação com um *timestamp* global no momento em que se inicia
- A idéia por trás do algoritmo é que quando um processo irá bloquear, aguardando por um recurso que outro processo está usando, uma *checagem* é iniciada para verificar qual possui o *timestamp* maior (ou seja, o mais jovem). Pode-se, então, permitir a espera somente se o processo possuir um *timestamp* menor (é mais velho) que o processo por quem ele deve esperar.
- Desta maneira, seguindo uma lista de processos bloqueados, o *timestamp* estará sempre crescendo e, portanto, ciclos serão impossíveis.

Detecção de Deadlocks em Sistemas Distribuídos – Prevenção