MC714

Sistemas Distribuídos

2° semestre, 2014

Migração de código

Migração de código

- Passagem de programas, não apenas de dados.
- Tradicionalmente: migração de processo
  - Mover processo em execução.
  - Custoso, complicado.
  - Justificativa: desempenho pode melhora ao mover processos.

- Ex.: cliente-servidor, servidor é banco de dados.
  - BD muito grande, aplicação com muitas requisições
  - Pode ser melhor mover parte da aplicação para servidor.
  - Processamento próximo à origem dos dados.

Migração de código

- Também pode-se pensar em mover parte do servidor para cliente
  - Exemplo de pré-processamento de formulário
- Migração de código pode ajudar a melhor explorar paralelismo
  - Ex. agente móvel em buscas
  - Mesmo código despachado várias vezes
  - Aumento linear sem utilizar programação paralela

Migração de código

- Além de melhorar desempenho, também pode aumentar flexibilidade.
- Ao invés de quebrar aplicação e decidir previamente onde cada parte executa, pode ser configurado dinamicamente.
  - Ex: fornecer interface/código em tempo real, antes da invocação.
  - Fig. 61.
  - Requer protocolo para baixar código/inicializar.
  - Necessário que código possa ser executado na máquina cliente.

Migração de código

- Baixar software dinamicamente não requer todo software instalado com antecedência.
  - Tanto recebido quanto descartado quando necessário.
  - Por ex., pode-se alterar protocolos sem necessidade de atualizações manuais de software.
  - Problemas de segurança: garantir que código baixado é confiável (pode acessar outros dados...)

Migração de código - modelos

- Processo dividido em três segmentos
  - Segmento de **código**: conjunto de instruções que compõem o programa em execução
  - Segmento de **recursos**: referências a recursos externos de que o processo necessita (arquivos, dispositivos)
  - Segmento de **execução**: armazena estado da execução do processo (dados privados, pilha, contador de programa)

Migração de código - modelos

- Mínimo essencial: mobilidade fraca
- Possível transferir segmento de código (+ alguns dados de inicialização).
- Programa transferido é iniciado de acordo com posições pré-definidas.
  - Ex. applets java iniciam sempre do começo.
- Requer somente que máquina alvo possa executar aquele código (torná-lo portável)

Migração de código - modelos

- Outro modelo: mobilidade forte
- Segmento de execução também pode ser transferido.
- Processo em execução pode ser parado e movido
  - Retoma do ponto que parou
- Mais geral que mobilidade fraca, mas mais difícil de implementar.

Migração de código - modelos

- Outra distinção: quem inicia migração
  - Remetente
  - Destinatário

- Remetente: migração iniciada pela máquina em que o código está em execução.
  - Transferir programas para um servidor de computação
  - Enviar programa de busca, por exemplo.
- Destinatário: iniciativa de migração tomada pela máquina-alvo.
  - Applets java.

Migração de código - modelos

- Destinatário:
  - Mais simples que pelo remetente.
  - Cliente toma iniciativa de migração.
- Contrário requer mecanismos de confiança entre cliente e servidor
  - Login/identificação
- Destinatário pode ser feita no anonimato
  - Servidor não se interessa pelos dados do cliente (?)
  - Migração serve para melhorar desempenho do cliente.

Migração de código - modelos

- Em mobilidade fraca: execução pelo processo-alvo ou novo processo.
  - Applets java são executados pelo browser.
  - Não há necessidade de novo processo.
  - Processo que executa precisa de alguma proteção em relação ao código a ser executado.

Migração de código - modelos

- Outro tipo de mobilidade forte: clonagem remota
  - Em contraste à migração de processos, produz cópia exata do processo original em máquina diferente.
  - Processo clonado: executa em paralelo com original.
  - Unix: bifurcação de processo e execução em máquina remota.

- Fig. 62.

Migração de código – segmento de recurso

- Segmento de recurso nem sempre pode ser transferido
  - Ex.: referência a uma porta TCP específica de uma conexão com outros processos.
  - Ao mover, precisa devolver a porta e requisitar uma nova.
- Mas pode não ser problema em outros casos
  - Referência a arquivo com URL absoluto.

Migração de código – segmento de recurso

- Três tipos de vinculação de processo-recurso
  - **Vinculação por identificador:** processo se refere a um recurso por seu identificador e requer exatamente o recurso referenciado.
  - Ex. Referências URL, terminais locais de comunicação.
  - **Vinculação por valor:** somente o valor do recurso é necessário.
  - Execução normal se outro recurso fornece mesmo valor.
  - Ex. Bibliotecas padronizadas (C, java...).

Migração de código – segmento de recurso

- **Vinculação por tipo**: processo precisa de um recurso com um tipo específico.
- Ex. dispositivos locais (monitor, impressora).

- Migração de código pode necessitar mudanças em referências.
- Se e como a referência deve ser mudada depende se recurso pode ser movido com código.
- Considerar vinculações recurso-máquina.

Migração de código – segmento de recurso

- Alterar vinculações recurso-máquina:
- **Recursos não ligados:** podem ser movidos com facilidade entre máquinas diferentes: arquivos de dados associados somente ao programa migrado.
- **Recurso amarrado:** pode ser possível mover ou copiar, mas a custos mais altos (bancos de dados locais, sites web completos).
- **Recurso fixo:** vinculados a uma máquina ou ambiente específico, não podem ser movidos (dispositivos locais).

Migração de código – segmento de recurso

- Nove combinações de vinculação: 3 processo-recurso e 3 recurso-máquina.
- Fig. 63.
- Por identificador:
  - **Não ligado:** mover recurso ou referência global, por ex. URL se compartilhado com outros processos.
  - **Amarrado ou fixo:** criar referência global.

Migração de código – segmento de recurso

- Por valor:
  - **Não ligado:** copiar ou mover para novo destino. Se compartilhado, referência global.
  - **Amarrado:** bibliotecas: cópias geralmente disponíveis, caso contrário devem ser feitas antes da migração. Referência global quando muitos dados devem ser copiados.
  - **Fixo:** memória compartilhada entre processos. Referência global implicaria memória compartilhada distribuída.
- Por tipo:
  - Vincular novamente a recurso de mesmo tipo no local.
  - Se não estiver disponível, copiar ou mover, ou estabelecer referência global.

Migração de código – sistemas heterogêneos

- SDs: diferentes SOs, arquiteturas...
- Problemas similares aos da portabilidade
  - Assim como as soluções...
- Geração de código intermediário independente de máquina.
- Linguagens script e linguagens portáveis (java).
  - Máquina virtual ou intérprete de código fonte.
- Migração de ambientes inteiros (ex. migração de máquinas virtuais).

Migração de código – sistemas heterogêneos

- Migração de máquinas virtuais em tempo real.
  - Migrar toda a imagem da memória e vinculações a recursos locais.
- Migrar imagem da memória: 3 opções
  - Enviar as páginas de memória. Reenviar as que forem modificadas durante migração.
  - Parar máquina virtual, migrar memória e reiniciar na nova máquina.
  - Deixar a máquina “puxar” novas páginas conforme necessário (pós-cópia): copiar páginas sob demanda.

Migração de código – sistemas heterogêneos

- Parar máquina virtual: tempo ocioso pode ser inaceitável, dependendo do tipo de recurso fornecido pela máquina.
- Mover páginas sob demanda: pode prolongar período de migração; pode resultar em mau desempenho (demora para conjunto de dados de um processo migrar).
- Combinar primeira e segunda opções: pré-cópia. Fase de parar para copiar é muito mais breve.
  - Tempos de parada de 200ms ou menos.
- Fig 64.

Comunicação

Comunicação

- Comunicação entre processos: cerne de todo sistema distribuído.
- SDs: milhares a milhões de processos espalhados por uma rede com comunicação não confiável.
- Protocolos
- Modelos de comunicação: Remote procedure call - RPC, Message-Oriented Middleware – MOM; e fluxo de dados).
- Multicasting

Comunicação - fundamentos

Protocolos em camadas

- Ausência de memória compartilhada
  - Envio e recebimento de mensagens em baixo nível.
- Processo *A* Monta mensagem, faz chamada de sistema, SO envia mensagem ao destino *B.*
  - *A* e *B* precisam concordar com significado dos bits.
- Vários acordos: voltagem bits 0 e 1, qual o último bit da mensagem, detectar mensagens perdidas/danificadas, representação de dados...

- Modelo de referência International Organization for Standardization – ISO
- Modelo ISO OSI – Open Systems Interconnection Reference

Modelo ISO OSI

- Modelo OSI: até 7 camadas.
- Fig. 65
- Cada camada lida com um aspecto específico.
  - Problema quebrado em pedaços gerenciados independentemente.
- Cabeçalhos/trailers são adicionados em cada nível.
- Transmissão pela camada física.
- Cada camada retira e examina seu próprio cabeçalho.
- Fig. 66

Modelo ISO OSI

- Pilha de protocolos
  - Modelo de referência != protocolos utilizados
- Protocolos desenvolvidos para internet (TCP/IP) são os mais usados.

Modelo ISO OSI

- 3 camadas inferiores implementam funções básicas de rede
- Camada física
  - Transmitir bits
  - Quantos volts usar para 0 e para 1
  - Quantos bits por segundo
  - Ambas as direções simultaneamente?
  - Tamanho/forma do conector, número de pinos...
  - Padronização das interfaces elétrica, mecânica e de sinalização.

Modelo ISO OSI

- Camada de enlace
  - Agrupar bits em unidades (quadros)
  - Coloca padrão especial de bits no início e no final de cada quadro
  - Checksum
  - Número no cabeçalho para identificar sequencia
  - Remetente não localiza receptor: põe quadro na rede e o receptor o retira.

Modelo ISO OSI

- Camada de rede
  - Redes diferentes: mensagem pode fazer saltos
  - Escolher o melhor caminho
  - Roteamento: tarefa primária da camada de rede
  - Rota mais curta nem sempre a melhor
  - Mais utilizado: IP – internet protocol
  - Pacote – termo para uma mensagem na camada de rede
  - Roteamento independente, por pacote

Modelo ISO OSI

- Camada de transporte
  - Transforma rede em algo que um desenvolvedor pode usar
  - Serviço de transmissão entre remetente e receptor
  - Aplicação entrega mensagem com a expectativa que será entregue sem se perder.
  - Quebra mensagem, numera e envia.
  - Conexões de transporte confiáveis: rede orientada a conexão ou não – sequencia correta garantida ou não. Camada de transporte ordena.
  - Fornece comportamento de comunicação fim-a-fim.
  - Transmission Control Protocol – TCP – conexão
  - Universal Datagram Protocol – UDP – sem conexão

Modelo ISO OSI

- Camadas adicionais: internet agrupou camadas superiores (apresentação, sessão, aplicação) em uma única.
- Sessão: controle de diálogo – sincronização – pontos de verificação em transferências
- Apresentação: significado dos bits – definir campos da mensagem
  Aplicação: era para ser um conjunto de aplicações padronizadas (no modelo OSI).
  - Engloba qualquer aplicação que não se ajuste a uma das camadas subjacentes
- Da perspectiva do modelo OSI, middlewares são aplicações.

Protocolos de middleware

- Middleware: aplicação que reside logicamente, na maioria das vezes, na camada de aplicação, mas que contém protocolos de uso geral que justificam suas próprias camadas.
- Exemplos:
  - Protocolos de autenticação: não vinculados a uma aplicação específica.
  - Protocolos distribuídos de comprometimento (atomicidade).
  - Protocolo distribuído de bloqueio de recurso.

Protocolos de middleware

- Não substituem camadas do modelo de referência: aplicações podem, por exemplo, usar protocolos de transporte diferentes.

- Camada de middleware (protocolos) substitui camadas de sessão e apresentação na Fig. 65.

Tipos de comunicação

- Middleware: serviço intermediário na comunicação de nível de aplicação.
- Fig. 67
- Ex.: correio eletrônico
  - Comunicação é **persistente.**
  - Middleware armazena comunicação por tempo necessário
- **Comunicação transiente:**
  - Mensagem armazenada somente pelo tempo que a aplicação remetente e receptora estiverem executando.

Tipos de comunicação

- Comunicação assíncrona: remetente continua sua execução após ter apresentado a sua mensagem para transmissão.
- Comunicação síncrona: remetente bloqueado até saber que sua requisição foi aceita.
  - Sincronizar na apresentação da requisição (análoga: síncrona não bloqueante sem buffer no middleware)
  - Sincronizar na entrega da requisição (análoga: síncrona não bloqueante com buffer no middleware)
  - Sincronizar após processamento pelo servidor (análoga: síncrona bloqueante)

Chamada de Procedimento Remoto

RPC

- Permitir que programas chamem procedimentos localizados em outras máquinas.
- Processo A chama procedimento na máquina B
- A é suspenso
- Execução de procedimento ocorre em B
- Informações podem ser transportadas do chamador ao chamado, por parâmetros, e podem retornar no resultado do procedimento.
- Trocas de mensagem não são visíveis ao programador.

RPC

- Máquinas diferentes, espaços de endereçamento diferentes.
- Passar parâmetros e resultados: pode demandar conversões se máquinas não forem idênticas.
- Máquinas podem falhar.

- É possível lidar com muitos desses problemas: RPC é bastante utilizado.

RPC – operação básica

- **Primeiro:** rever chamada de procedimento convencional em C
- Count = read(fd, buf, nbytes)
  - fd inteiro que indica um arquivo
  - buf é um vetor de caracteres no qual dados são lidos
  - nbytes inteiro que informa quantos bytes ler
- Pilha fig. 68 (chamada pelo programa principal)

RPC – operação básica

- Parâmetro por valor: parâmetro é variável local com valor definido. Modificações não alteram valor original.
- Parâmetro por referência: ponteiro – i.e. endereço da variável
  - Chamada read: segundo parâmetro por referência (vetor).
  - Pilha: endereço do vetor de caracteres.
  - Modificar algo no vetor, modifica original.

RPC – operação básica

- Outro mecanismo (não usado em C): chamada por copiar/restaurar.
- Copiar valor para pilha, como na chamada por valor, e então copiá-lo de volta após a chamada, sobrescrevendo valor original.
- Mesmo efeito que passagem por referência em muitas situações. Mas não em algumas situações, como ... ?
- Qual mecanismo usar: projeto da linguagem.

RPC – operação básica

- Idéia da RPC: fazer com que uma chamada remota pareça uma local → seja transparente.
- Chamada local de *read:*
  - Rotina extraída da biblioteca pelo linker e inserida no programa objeto.
  - Interface entre o código de usuário e o sistema operacional local.

RPC – operação básica

- RPC consegue transparência de modo análogo:
  - Se *Read* é procedimento remoto, por ex. máquina servidor de arquivos.
  - Versão de read diferente: apêndice de cliente
  - Mesma sequencia de chamada da fig 68.
  - Também chamada SO local.
  - Não pede dados ao SO: empacota parâmetros em uma mensagem e requisita que seja enviada.
  - Chama send e depois receive, bloqueando até receber resposta.
  - Fig. 69.

RPC – operação básica

- Mensagem chega ao servidor: apêndice de servidor
- Equivalente ao apêndice cliente
  - Transforma requisições que vêm pela rede em chamadas de procedimentos locais.
  - Normalmente, terá chamado *receive* e bloqueado esperando mensagens.
  - Desempacota parâmetros e chama procedimento local.
- Servidor não precisa saber que chamador é remoto.
  - No exemplo do read, buffer será interno ao apêndice de servidor.

RPC – operação básica

- Apêndice retoma controle depois de read efetuado.
- Empacota resultados (i.e., o buffer) em uma mensagem e chama send.
- Apêndice de servidor, usualmente, chama *receive* novamente.
- No cliente, mensagem é copiada ao buffer que está esperando e processo cliente é desbloqueado.
- Apêndice de cliente desempacota resultado, copia ao cliente e retorna da maneira usual.
- Chamador não sabe que era remoto.

RPC – operação básica

- Ao cliente: chamada comum, não send e receive.
- Resumo:

1. Procedimento cliente chama apêndice cliente.
2. Apêndice cliente constrói uma mensagem e chama SO local.
3. SO cliente envia mensagem ao SO remoto.
4. SO remoto dá a mensagem ao apêndice servidor.
5. Apêndice servidor descompacta parâmetros e chama o servidor.

RPC – operação básica

6. Servidor faz serviço e retorna resultado para apêndice.
7. Apêndice de servidor empacota resultado em uma mensagem e chama SO local.
8. SO do servidor envia mensagem ao SO cliente.
9. SO cliente dá a mensagem ao apêndice de cliente
10. Apêndice desempacota resultado e retorna ao cliente.

- Efeito líquido: nem cliente nem servidor ficam cientes das etapas intermediárias ou da existência da rede.