Arquiteturas Paralelas e Distribuídas

RPC e paradigmas de interação entre processos

Eduardo Furlan Miranda

Baseado em: PEREIRA, Caique Silva. Sistemas distribuídos. Londrina: EDE, 2019. ISBN: 978-85-522-1443-4.

Paradigmas de interação entre processos

- Paradigmas são fundamentais para entender a comunicação e coordenação em sistemas distribuídos e concorrentes

- Cada um dos paradigmas a seguir descreve de forma precisa os paradigmas de interação entre processos, especialmente no contexto de
  - RPC (Chamada de Procedimento Remoto)
  - Sistemas distribuídos

Chamadas de Procedimento Remoto (RPC)

- Permite que um programa execute procedimentos em outro sistema remoto como se fosse uma chamada local
  - A sub-rotina pode estar, p.ex., em outro processo
- Abstrai os detalhes da comunicação de rede
  - Fornece uma interface simples e transparente para o desenvolvedor
- Utiliza módulos “stub” (cliente e servidor) que gerenciam a serialização e desserialização de dados
  - Cliente: quem chama
  - Servidor: a sub-rotina chamada

Chamadas de Procedimento Remoto (RPC)

- Quando o cliente chama o servidor, o RPC deve cuidar de
  - Pegar todos os parâmetros passados para a sub-rotina e transferi-los para o nó remoto
  - Executar a sub-rotina no nó remoto
  - Transferir de volta todos os parâmetros retornados para a rotina de chamada

- Pode ser implementado de forma
  - Síncrona (mais comum)
    - O cliente envia uma requisição e aguarda pela resposta antes de continuar sua execução
    - Esse modelo garante que os dados retornados sejam usados imediatamente, mas pode causar bloqueios e aumentar a latência
  - Assíncrona (menos comum)
    - Continua executando outras tarefas enquanto a resposta é processada

Troca de Mensagens (*Message Passing*)

- Os processos se comunicam enviando e recebendo mensagens explicitamente
- Pode ser implementado de forma
  - Síncrona
    - O remetente espera até que a mensagem seja recebida
  - Assíncrona
    - O remetente continua sua execução sem aguardar a confirmação
- Útil em cenários onde os processos precisam se comunicar de maneira flexível e desacoplada (ex.: entre nós)
- Amplamente utilizada em sistemas distribuídos e concorrentes, como MPI (*Message Passing Interface*)

```
%%writefile mpi-example.f90
PROGRAM mpi_example
    USE mpi
    IMPLICIT NONE
    INTEGER :: rank, size, ierr, msg, status(MPI_STATUS_SIZE)
    CALL MPI_INIT(ierr)                                  ! Inicializa o MPI
    CALL MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)   ! Obtém o número de processos
    CALL MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)   ! Obtém o identificador do processo
    IF (size /= 2) THEN
        PRINT *, "Este programa requer exatamente 2 processos."
    ELSE
        IF (rank == 0) THEN
            msg = 42  ! Mensagem a ser enviada
            CALL MPI_SEND(msg, 1, MPI_INTEGER, 1, 0, MPI_COMM_WORLD, ierr)
            PRINT *, "Processo 0 enviou a mensagem:", msg
        ELSE IF (rank == 1) THEN
            CALL MPI_RECV(msg, 1, MPI_INTEGER, 0, 0, MPI_COMM_WORLD, status, ierr)
            PRINT *, "Processo 1 recebeu a mensagem:", msg
        END IF
    END IF
    CALL MPI_FINALIZE(ierr)  ! Finaliza o MPI
END PROGRAM mpi_example
```

```
Overwriting mpi-example.f90
```

```
! mpif90 mpi-example.f90 -o mpi-example.out
```

```
! mpirun -np 2 ./mpi-example.out
```

```
 Processo 0 enviou a mensagem:          42
 Processo 1 recebeu a mensagem:          42
```

Memória compartilhada distribuída (*Distributed Shared Memory* - DSM)

- Os processos interagem compartilhando um espaço de memória virtual que é acessível por todos os processos envolvidos, independentemente de suas localizações físicas

- Abstrai a complexidade da comunicação de rede, permitindo que os processos acessem e modifiquem dados compartilhados como se estivessem em uma única máquina

- Para garantir consistência, mecanismos como coerência de cache são frequentemente empregados
  - Cada processo pode ter sua própria cópia local de dados em caches individuais

- Pode introduzir desafios de desempenho devido à latência de rede e à necessidade de sincronização

Publicação/Assinatura (*Publish/Subscribe*)

- Os processos interagem através de um sistema de eventos
- Um processo (publicador) envia mensagens para um tópico específico, e outros processos (assinantes) recebem essas mensagens apenas se estiverem inscritos no tópico
- Altamente escalável e desacoplado, pois os publicadores e assinantes não precisam conhecer uns aos outros diretamente

Transmissão de Dados em Fluxo (*Streaming*)

- Envolve o envio contínuo de dados entre processos, permitindo que os dados sejam processados em tempo real à medida que chegam

- Amplamente utilizado em sistemas de *streaming* de vídeo, análise de dados em tempo real, e comunicações bidirecionais, como chats ao vivo

Coordenação Baseada em Eventos (*Event-Driven*)

- Os processos reagem a eventos específicos, como cliques do usuário, mudanças de estado, ou atualizações de dados
- Esses eventos podem ser gerados por interações do usuário, sensores, ou até mesmo por outros processos
- O sistema é projetado para ser altamente responsivo, executando ações apenas quando eventos relevantes ocorrem

Passagem de Tokens (*Token Passing*)

- Um *token* é usado como um mecanismo de controle para
  - Coordenar o acesso a recursos compartilhados
  - Regular a comunicação entre processos
- O *token* é passado sequencialmente entre os processos em um sistema distribuído
  - Apenas o processo que possui o *token* pode realizar determinadas operações
    - Como acessar um recurso crítico ou enviar mensagens

Chamadas de Métodos Remotos (*Remote Method Invocation* - RMI)

- Paradigma de interação que estende o conceito de RPC para linguagens orientadas a objetos
- Permite que um objeto em uma máquina invoque métodos de um objeto localizado em outra máquina remota
  - Como se estivessem no mesmo ambiente de execução
- A principal diferença entre RMI e RPC é que o RMI opera em termos de objetos e interfaces, utilizando serialização para transmitir objetos pela rede
- O RMI é específico para linguagens orientadas a objetos (como Java)

Comunicação Baseada em Troca de Arquivos (*File Sharing*)

- Ocorre quando processos interagem transferindo arquivos que contêm dados ou instruções
- Esses arquivos podem ser usados para armazenar informações intermediárias, resultados de processamento ou configurações que guiam o comportamento dos processos
- Embora seja uma abordagem menos dinâmica em comparação com paradigmas como RPC, ou troca de mensagens, ela é altamente eficaz em cenários onde a comunicação não precisa ser em tempo real, como em
  - Sistemas de lote (*batch*)
  - Pipelines de processamento de dados (sequência de etapas)
  - Backups distribuídos