**Tema 11 - Comunicação em Sistemas Distribuídos**

A comunicação em sistemas distribuídos envolve trocas de mensagens devido à falta de memória compartilhada, exigindo que processos em diferentes máquinas troquem informações através de protocolos de rede. Esse processo, que parece simples, requer um acordo claro sobre a estrutura e o significado dos dados para evitar erros ou interpretações incorretas.

O Modelo de Referência OSI da Organização Internacional de Normalização (ISO) foi desenvolvido para facilitar a comunicação entre sistemas abertos, isto é, sistemas preparados para interagir com qualquer outro sistema aberto através de regras padronizadas que definem o formato, o conteúdo e o significado das mensagens. Este modelo divide a comunicação em sete camadas, cada uma abordando diferentes aspectos da comunicação e permitindo que o problema seja dividido em partes gerenciáveis, solucionadas independentemente. Cada camada oferece uma interface à camada acima, definindo serviços específicos para os usuários.

Existem dois tipos principais de protocolos: orientados à conexão e sem conexão. Nos protocolos orientados à conexão, uma conexão é explicitamente estabelecida e terminada após a comunicação, como no sistema telefônico. Nos protocolos sem conexão, nenhuma preparação é necessária; a comunicação é mais direta, como enviar uma carta.

As três camadas inferiores do modelo OSI são dedicadas às funções básicas de rede. A camada física lida com a transmissão de bits, estabelecendo padrões como a tensão usada para representar 0s e 1s e a capacidade de transmissão, além de padronizar interfaces elétricas e mecânicas para garantir a correta recepção dos bits, com normas como RS-232-C para comunicação serial.

A camada de enlace de dados estrutura os bits em quadros e usa um padrão especial de bits no início e no final de cada quadro para garantir a recepção correta, além de calcular um checksum pela soma dos bytes. Se o checksum corresponder ao esperado, o quadro é aceito; caso contrário, é solicitada a retransmissão. Os quadros são identificados por números de sequência.

Na camada de rede, o roteamento de mensagens pode envolver várias conexões. Ela escolhe a melhor rota com base no atraso, que pode variar de acordo com o tráfego e o acúmulo de mensagens. Os algoritmos de roteamento podem se adaptar às mudanças de carga ou usar médias de longo prazo. O protocolo IP (Protocolo de Internet), utilizado amplamente, opera sem conexão e permite que pacotes IP sejam roteados independentemente até seu destino.

A camada de transporte complementa a camada de rede ao garantir a entrega confiável e sem perdas de mensagens das aplicações. Divide as mensagens em pacotes, atribui números de sequência e gerencia a transmissão, incluindo o controle de pacotes enviados e recebidos e a capacidade de recepção. Ela pode operar sobre serviços de rede tanto orientados à conexão, onde pacotes são recebidos em sequência, quanto sem conexão, requerendo que a camada reordene pacotes para manter a integridade da mensagem.

Os protocolos da camada de transporte incluem TCP (Protocolo de Controle de Transmissão), que junto ao IP, forma o TCP/IP, padrão de internet que garante que todos os segmentos de dados sejam recebidos e ordenados corretamente. Para aplicações menos críticas, usa-se o UDP (Protocolo de Datagrama Universal), que adiciona funcionalidades mínimas ao IP, adequadas para aplicações que não necessitam de garantias de entrega confiável ou ordenada.

Outros protocolos, como o Protocolo de Transporte em Tempo Real (RTP), atendem a necessidades específicas, como a transferência de dados em tempo real, delineando formatos de pacotes para aplicações nesse contexto sem garantir a entrega, focando-se no formato e na gestão da transmissão. Acima da camada de transporte, a camada de sessão oferece controle de diálogo e facilidades de sincronização, permitindo a inserção de pontos de controle em transferências longas para facilitar a retomada após falhas.

A camada de apresentação trata da representação dos dados em alto nível, possibilitando a comunicação entre máquinas com diferentes representações internas através da definição de registros com campos estruturados e a adoção de sistemas padronizados de representação de caracteres, além de gerenciar a compressão e codificação de dados. A camada de aplicação acomoda uma variedade de aplicações de rede e expandiu-se para incluir todas as aplicações que não se ajustam perfeitamente às outras camadas. Ela engloba serviços como correio eletrônico, transferência de arquivos e execução remota, fornecendo serviços diretamente aos usuários finais.

No modelo OSI, os sistemas distribuídos são tratados como aplicações, sem uma distinção clara entre aplicações, protocolos específicos de aplicações e protocolos de uso geral. O FTP, por exemplo, define a transferência de arquivos entre cliente e servidor e não deve ser confundido com aplicações que realizam essa transferência e empregam o mesmo protocolo. O HTTP, desenvolvido para gestão e transferência de páginas da Web e implementado por navegadores e servidores web, é também utilizado em contextos não-web. Existem ainda protocolos de uso geral que são úteis para diversas aplicações, mas que não se classificam como protocolos de transporte, frequentemente categorizados como middleware.

Middleware, geralmente localizado na camada de aplicação, engloba protocolos de uso geral que não dependem de aplicações específicas, tais como protocolos de autenticação, que fornecem prova de identidade reivindicada, e protocolos de autorização, que permitem acesso a recursos apenas para usuários e processos autenticados. Protocolos de commit distribuído também são exemplos de middleware, garantindo atomicidade em transações ao assegurar que operações sejam realizadas por todos os processos envolvidos ou por nenhum. Protocolos de bloqueio distribuído protegem recursos contra acessos simultâneos em múltiplas máquinas.

O middleware também oferece serviços de comunicação de alto nível, como chamadas de procedimento remoto e sincronização de streams para transferências de dados em tempo real, essenciais em aplicações multimídia, além de serviços confiáveis de multicast que podem escalar para milhares de receptores em redes de grande área. Embora alguns protocolos de middleware possam ser adequados para a camada de transporte, frequentemente são mantidos em camadas superiores para atender requisitos específicos de aplicações, como multicasting confiável.

Para ilustrar como o middleware pode servir como um serviço adicional em computação cliente-servidor, considere um sistema de correio eletrônico, operando como um serviço de comunicação de middleware. Cada host executa um agente do usuário que permite compor, enviar e receber e-mails. O agente do usuário remetente envia o e-mail ao sistema de entrega, que se encarrega de entregá-lo ao destinatário. O agente do usuário receptor verifica novos e-mails e os transfere para visualização e leitura pelo usuário.

A comunicação em sistemas de middleware pode ser persistente ou transitória, assíncrona ou síncrona. Na comunicação persistente, as mensagens são armazenadas pelo tempo necessário para serem entregues, independentemente de as aplicações remetente e receptora estarem em execução. Na comunicação transitória, as mensagens só são armazenadas enquanto as aplicações estão ativas. Na comunicação assíncrona, o remetente prossegue imediatamente após submeter uma mensagem, enquanto na comunicação síncrona, ele fica bloqueado até que sua solicitação seja aceita, podendo a sincronização ocorrer em diversos pontos do processo.

Há várias combinações de persistência e sincronização utilizadas na prática. Uma das mais comuns é a persistência com sincronização no momento da submissão da solicitação, frequentemente adotada em sistemas de enfileiramento de mensagens. Analogamente, a comunicação transitória com sincronização após o processamento completo da solicitação também é amplamente empregada, equivalendo às chamadas de procedimento remoto.

Além de persistência e sincronização, é importante diferenciar entre comunicação discreta e contínua. A comunicação discreta envolve a troca de mensagens, com cada mensagem constituindo uma unidade completa de informação. Já a comunicação contínua implica no envio sequencial de múltiplas mensagens relacionadas temporalmente ou pela ordem de envio.

Nos sistemas distribuídos que baseiam-se em troca explícita de mensagens entre processos, surge a necessidade de transparência de acesso, o que pode ser alcançado através da Chamada de Procedimento Remoto (RPC), conforme sugerido por Birrell e Nelson. No RPC, quando um processo em uma máquina A chama um procedimento em B, o processo chamador em A é suspenso, e a execução ocorre em B. Informações são transportadas através dos parâmetros e retornadas nos resultados do procedimento sem que o programador perceba a passagem de mensagens.

Entender a chamada de procedimento convencional é essencial para compreender o RPC. Em uma chamada de procedimento típica em C, como **count = read(fd, buf, nbytes);**, os parâmetros são organizados na pilha em ordem inversa, e após a execução, o procedimento retorna o valor em um registro, limpa o endereço de retorno e o controle é devolvido ao chamador. Parâmetros podem ser passados por valor ou por referência, e essa distinção é crucial no RPC devido às diferenças nos espaços de endereçamento entre as máquinas.

O RPC visa tornar as chamadas remotas indistinguíveis das locais, proporcionando transparência na execução. Por exemplo, ao fazer uma chamada ao procedimento read, o programador pode estar interagindo com um servidor remoto sem perceber. No servidor, um "stub do servidor" recebe a mensagem, desempacota os parâmetros e executa o procedimento como se fosse local. Após a execução, o resultado é enviado de volta ao cliente pelo stub do servidor. Quando o cliente recebe a resposta, o stub do cliente desempacota o resultado e o retorna ao chamador original. Esse processo mantém os detalhes da comunicação de rede ocultos, permitindo que as aplicações cliente e servidor funcionem como se a interação fosse local, garantindo a integridade e a transparência das chamadas.

Em chamadas de procedimento convencionais, um cliente que faz uma RPC fica bloqueado até que uma resposta seja retornada. Contudo, esse comportamento estrito de solicitação-resposta é desnecessário quando não há resultado a retornar, apenas levando ao bloqueio do cliente que poderia continuar trabalhando após fazer a solicitação. RPCs assíncronos são úteis nesses casos, permitindo que o cliente continue imediatamente após emitir a solicitação. Nesse sistema, o servidor envia uma resposta assim que recebe a solicitação, apenas para indicar que processará a RPC. O cliente prossegue sem bloqueios assim que recebe o reconhecimento do servidor.

Problemas de representação de dados entre máquinas distintas, como formatos de caracteres e numéricos, podem surgir, tornando necessário adaptar os dados ao sistema do receptor. A passagem de ponteiros é complexa em RPC, pois um ponteiro só tem significado dentro do espaço de endereço do processo que o utiliza. Em muitos casos, os dados referenciados pelo ponteiro são copiados para a mensagem enviada ao servidor, substituindo a passagem por referência por uma operação de copiar/restaurar, frequentemente adequada.

Para ocultar RPCs, é crucial que o chamador e o chamado concordem com o formato e o protocolo das mensagens. Diferenças em como os dados são representados, como inteiros em complemento de dois ou caracteres em Unicode, devem ser padronizadas. Essa padronização garante que mensagens sejam interpretadas sem ambiguidades. As RPCs são uma base amplamente adotada para middleware e sistemas distribuídos. DCE (Distributed Computing Environment), desenvolvido pela Open Software Foundation, é um exemplo de sistema RPC, projetado para funcionar como uma camada de abstração entre sistemas operacionais de rede existentes e aplicações distribuídas. DCE permite que aplicações distribuídas operem em uma variedade de sistemas operacionais sem perturbar aplicações não distribuídas existentes. DCE oferece serviços como sistema de arquivos distribuídos, diretório, segurança e tempo distribuído, facilitando o acesso e a gestão de recursos distribuídos globalmente e a sincronização de relógios em máquinas distintas.

Para que um cliente possa chamar um servidor, é necessário que o servidor esteja registrado e preparado para aceitar chamadas. O registro do servidor possibilita que o cliente localize e se conecte a ele. A localização do servidor ocorre em duas etapas: primeiro, localiza-se a máquina do servidor e, depois, o processo correto nessa máquina. Essencialmente, para se comunicar com o servidor, o cliente precisa saber um ponto final (ou porto) na máquina do servidor para enviar mensagens. Esse ponto final é usado pelo sistema operacional do servidor para distinguir mensagens de entrada para diferentes processos. No DCE, uma tabela de pares (servidor, ponto final) é mantida em cada máquina servidor por um processo chamado daemon DCE. Antes de estar disponível para solicitações, o servidor deve solicitar um ponto final ao sistema operacional e registrar esse ponto final com o daemon DCE. Este registra a informação na tabela de pontos finais, incluindo quais protocolos o servidor utiliza. O servidor também se registra no serviço de diretório fornecendo o endereço de rede da máquina do servidor e um nome sob o qual pode ser localizado.

A vinculação de um cliente a um servidor procede como segue: o cliente fornece o nome do servidor ao servidor de diretório, que retorna o endereço de rede da máquina onde o servidor de vídeo opera. Em seguida, o cliente contata o daemon DCE na máquina do servidor para obter o ponto final do servidor de vídeo listado na tabela de pontos finais. Com essa informação, a chamada de procedimento remoto (RPC) é possível. Para chamadas subsequentes, essa pesquisa inicial não é necessária.

A execução do RPC é realizada de maneira transparente e rotineira. O stub do cliente empacota os parâmetros e os envia utilizando o protocolo escolhido durante o processo de vinculação. Quando uma mensagem chega ao servidor, é roteada para o servidor correto com base no ponto final contido na mensagem. A biblioteca de tempo de execução entrega a mensagem ao stub do servidor, que desempacota os parâmetros e aciona o servidor. A resposta segue o caminho inverso.

RPC e invocações de objetos remotos são essenciais para ocultar a comunicação em sistemas distribuídos, melhorando a transparência de acesso. No entanto, esses mecanismos podem não ser adequados, especialmente quando não se pode assegurar que o lado receptor esteja ativo quando uma solicitação é feita. Em tais casos, serviços alternativos de comunicação são necessários. A natureza intrinsecamente síncrona dos RPCs, que bloqueia um cliente até que sua solicitação seja processada, às vezes precisa ser substituída por outras formas de mensagens.

Muitos sistemas e aplicações distribuídos baseiam-se diretamente no modelo de comunicação orientado a mensagens fornecido pela camada de transporte. Interfaces de sockets, introduzidas nos anos 1970 no UNIX de Berkeley, e a XTI, desenvolvida pela AT&T, são exemplos cruciais de interfaces padronizadas que permitem aos programadores empregar uma variedade de protocolos de mensagens através de um conjunto simples de primitivas.

Um socket é um ponto final de comunicação onde uma aplicação pode escrever dados para serem enviados pela rede subjacente e de onde os dados recebidos podem ser lidos. Os servidores tipicamente executam as primitivas socket, bind, listen e accept nesta ordem. O comando bind associa um endereço local ao socket criado, informando ao sistema operacional que o servidor deseja receber mensagens apenas no endereço e porta especificados. O comando listen, usado em comunicações orientadas à conexão, permite ao sistema operacional reservar buffers suficientes para um número máximo especificado de conexões que o chamador está disposto a aceitar. O comando accept bloqueia o chamador até que uma solicitação de conexão chegue. Quando isso ocorre, o sistema operacional cria um novo socket com as mesmas propriedades do original e o retorna ao chamador.

No lado do cliente, também se cria um socket utilizando a primitiva socket, mas geralmente não é necessário vincular explicitamente o socket a um endereço local, pois o sistema operacional pode alocar dinamicamente uma porta quando a conexão é estabelecida. A primitiva connect exige que o chamador especifique o endereço de nível de transporte para o qual uma solicitação de conexão deve ser enviada. O cliente fica bloqueado até que a conexão seja estabelecida com sucesso, permitindo então que ambos os lados comecem a trocar informações através das primitivas send e receive. Fechar uma conexão é simétrico ao usar sockets, realizado quando tanto o cliente quanto o servidor chamam a primitiva close.

Com o desenvolvimento de multicomputadores de alto desempenho, os desenvolvedores procuram primitivas orientadas à mensagem que permitam a criação de aplicações altamente eficientes. As primitivas devem ter um nível de abstração conveniente para facilitar o desenvolvimento de aplicações e minimizar a sobrecarga. Os sockets foram considerados inadequados por suportarem apenas primitivas de envio e recebimento simples e por serem projetados para protocolos de uso geral como o TCP/IP, não sendo adequados para os protocolos proprietários de redes de interconexão de alta velocidade usados em clusters de servidores de alto desempenho. Estes protocolos exigem uma interface que possa lidar com funcionalidades mais avançadas, como diferentes formas de buffering e sincronização. Resultado disso é que a maioria das redes de interconexão e multicomputadores de alto desempenho vêm com bibliotecas de comunicação proprietárias que oferecem uma variedade de primitivas de comunicação eficientes e de alto nível. No entanto, essas bibliotecas são mutuamente incompatíveis, criando um problema de portabilidade para os desenvolvedores de aplicações.

A necessidade de independência de hardware e plataforma incentivou a criação de um padrão para passagem de mensagens, conhecido como Interface de Passagem de Mensagens (MPI). Projetado para aplicações paralelas, o MPI é adaptado para comunicação transitória, operando diretamente sobre a rede subjacente e assumindo que falhas graves, como falhas de processo ou partições de rede, são fatais e não necessitam de recuperação automática. O MPI organiza a comunicação dentro de um grupo conhecido de processos, cada um identificado por um par (ID do grupo, ID do processo), que identifica exclusivamente a origem ou destino de uma mensagem.

As primitivas de mensageria do MPI suportam a comunicação transitória, com a primitiva MPI\_bsend permitindo que o remetente submeta uma mensagem para transmissão, geralmente primeiro copiada para um buffer local no sistema MPI. Assim que a mensagem é copiada, o remetente continua suas atividades, enquanto o sistema MPI local gerencia a transmissão assim que um receptor solicitar o recebimento. Existe também uma operação de envio bloqueante, MPI\_send, cuja semântica varia de acordo com a implementação. Esta primitiva pode bloquear o chamador até que a mensagem especificada tenha sido copiada para o sistema MPI do lado do remetente ou até que o receptor inicie uma operação de recebimento. A comunicação síncrona, em que o remetente é bloqueado até que seu pedido seja aceito para processamento adicional, é realizada através da primitiva MPI\_ssend. A forma mais forte de comunicação síncrona é oferecida pela primitiva MPI\_sendrecv, na qual o remetente envia um pedido ao receptor e fica bloqueado até que uma resposta seja retornada, assemelhando-se a uma RPC convencional.

Os sistemas de enfileiramento de mensagens, também conhecidos como Middleware Orientado a Mensagens (MOM), proporcionam suporte substancial para comunicação assíncrona persistente. Eles são capazes de armazenar mensagens de maneira intermediária, permitindo que tanto o remetente quanto o destinatário não necessitem estar ativos durante a transmissão. Esses sistemas são especialmente adequados para suportar transferências de mensagens que podem levar minutos, diferenciando-se de sistemas mais rápidos, como os e-mails da Internet.

A funcionalidade básica desses sistemas é que as aplicações comunicam-se inserindo mensagens em filas designadas. Estas mensagens são então encaminhadas por uma rede de servidores de comunicação e entregues ao destinatário, independentemente de sua disponibilidade no momento do envio. A maioria dos servidores de comunicação está diretamente conectada, facilitando o envio direto das mensagens ao servidor destino. Cada aplicação gerencia sua própria fila privada para receber mensagens, embora seja possível para várias aplicações compartilharem uma única fila.

Importante destacar que as mensagens depositadas nas filas permanecem até serem explicitamente removidas, proporcionando independência operacional entre remetente e destinatário. Isso implica que não é necessário que o destinatário esteja ativo no momento em que uma mensagem é enviada para sua fila, e igualmente o remetente não precisa estar ativo quando sua mensagem é recebida.

A interface de usuário para esses sistemas é intuitiva e simplificada: utiliza-se a primitiva put para adicionar mensagens a uma fila específica, enquanto a primitiva get permite que um processo autorizado remova a mensagem mais antiga da fila, com bloqueio ocorrendo apenas se a fila estiver vazia. Opções adicionais de chamada permitem buscar mensagens específicas com base em critérios como prioridade ou padrões de correspondência, e a variante não bloqueante, implementada pela primitiva poll, permite que o processo continue operando mesmo se a fila estiver vazia ou a mensagem desejada não for encontrada. Adicionalmente, muitos sistemas de enfileiramento permitem que um processo instale um manipulador como função de retorno de chamada, que é acionado automaticamente sempre que uma mensagem é adicionada à fila. Esse mecanismo também pode iniciar automaticamente um processo para buscar mensagens da fila quando nenhum processo estiver ativo, geralmente por meio de um daemon no lado do receptor que monitora continuamente a fila para novas mensagens e as processa conforme necessário.

Finalmente, é essencial que as mensagens sejam inseridas apenas em filas locais ao remetente, tais como filas na mesma máquina ou em máquinas próximas numa LAN acessível via RPC. Essas são chamadas de filas de origem. As mensagens inseridas devem especificar uma fila de destino para onde serão transferidas, cabendo ao sistema de enfileiramento a responsabilidade de fornecer filas para remetentes e destinatários e de assegurar a transferência correta das mensagens das filas de origem para as de destino.

Os gestores de filas são fundamentais na gestão de sistemas de enfileiramento de mensagens, interagindo diretamente com as aplicações que enviam ou recebem mensagens. Alguns gestores atuam como roteadores ou retransmissores, encaminhando mensagens para outros gestores de filas, o que permite o crescimento do sistema de enfileiramento e a formação de uma rede de sobreposição de aplicações sobre uma rede de computadores existente.

Gestores de filas especiais, conhecidos como relays, são úteis por várias razões. Por exemplo, em muitos sistemas, a ausência de um serviço de nomeação dinâmica que mantenha as correspondências atualizadas entre filas e localizações de rede torna a topologia da rede de enfileiramento estática. Neste cenário, cada gestor de fila precisa ter uma cópia do mapeamento de filas para locais de rede. Utilizar alguns roteadores que conhecem a topologia da rede pode simplificar a gestão da rede ao encaminhar mensagens diretamente para o roteador mais próximo e, a partir daí, para o destino pretendido.

Os relays também permitem o processamento secundário de mensagens, como o registro para segurança ou tolerância a falhas, e podem atuar como gateways, transformando mensagens em formatos compreensíveis pelos destinatários. Para propósitos de multicast, um relay pode colocar uma mensagem recebida em cada fila de envio correspondente. À medida que a rede cresce, a configuração manual torna-se impraticável, tornando essencial a adoção de esquemas de roteamento dinâmico, similares aos usados em redes de computadores.

Integrar aplicações existentes e novas em um único sistema de informação distribuído requer que as aplicações entendam as mensagens que recebem, o que frequentemente implica em enviar mensagens no mesmo formato que o destinatário espera. À medida que novas aplicações são adicionadas ao sistema, necessitando de formatos de mensagem distintos, todos os receptores potenciais devem ajustar-se para produzir o novo formato. Uma alternativa é adotar um formato de mensagem comum, mas isso muitas vezes é inviável para sistemas de enfileiramento de mensagens devido ao nível de abstração em que operam. Se as aplicações que compõem um sistema de informação distribuído são altamente diversas, o formato comum mais adequado pode ser apenas uma sequência de bytes. Embora existam alguns formatos de mensagem padrão para domínios de aplicação específicos, a abordagem usual é aceitar a diversidade de formatos e simplificar as conversões tanto quanto possível.

Em uma integração avançada de aplicativos empresariais, um corretor não apenas converte mensagens, mas também é responsável por compatibilizar aplicações com base nas mensagens que estão sendo trocadas. Nesse modelo, chamado de publicar/assinar, as aplicações enviam mensagens na forma de publicações. Elas podem publicar uma mensagem sobre um determinado tópico, que é então enviada ao corretor. Aplicações que declararam interesse em mensagens sobre esse tópico recebem essas mensagens do corretor. Formas mais avançadas de mediação também são possíveis. No centro de um corretor de mensagens está um repositório de regras e programas que podem transformar uma mensagem de um tipo para outro.

As mensagens enviadas entre sistemas em sistemas de enfileiramento de mensagens requerem integração para que as aplicações compreendam as mensagens que recebem. Em muitos casos, é necessário que as mensagens de saída do remetente estejam no mesmo formato que o receptor pode processar. No entanto, estabelecer um formato comum de mensagem como nos protocolos de rede tradicionais frequentemente não é viável devido à diversidade das aplicações dentro de um sistema de informação distribuído. Frequentemente, o melhor formato comum pode ser apenas uma sequência de bytes, embora existam formatos de mensagem comuns para certos domínios de aplicação.

Dado esse desafio, os sistemas de enfileiramento de mensagens muitas vezes incluem corretores de mensagens, ou \*message brokers\*, que atuam como gateways no nível da aplicação. Esses corretores de mensagens são responsáveis por converter mensagens de entrada para que possam ser compreendidas pela aplicação de destino. Por exemplo, se uma mensagem de entrada contém uma tabela de um banco de dados com um delimitador de final de registro específico e campos de comprimento fixo, mas a aplicação de destino espera um delimitador diferente e campos de comprimento variável, um corretor de mensagens pode ser utilizado para converter a mensagem para o formato esperado pela aplicação de destino. Em configurações mais avançadas, um corretor de mensagens pode atuar como um gateway entre diferentes aplicações de banco de dados, lidando com a conversão entre formatos de dados incompatíveis.

Além disso, os corretores de mensagens desempenham um papel crucial na integração avançada de aplicativos empresariais (EAI), coordenando as comunicações baseadas em mensagens entre aplicações que podem não ter um formato de mensagem comum. Isso é frequentemente realizado por meio de um modelo de publicar/assinar, onde as aplicações "publicam" mensagens sobre tópicos específicos que são então encaminhadas aos inscritos através do corretor de mensagens. Os corretores também podem realizar funções mais complexas de mediação e transformação de mensagens, o que é essencial quando as aplicações envolvidas não podem ser diretamente compatibilizadas devido a diferenças nos formatos de dados ou nos protocolos de comunicação.

Esses sistemas de enfileiramento e os corretores de mensagens facilitam a configuração de comunicações assíncronas e persistentes entre processos que podem estar executando aplicações muito diferentes, manipulando acessos a bases de dados ou realizando cálculos diversos, sem a necessidade de estarem ativos ao mesmo tempo. Isso contrasta com os sistemas de e-mail, que, embora também utilizem um modelo de enfileiramento de mensagens, geralmente são configurados para fornecer suporte direto ao usuário final e não necessariamente para comunicação entre processos em um sistema distribuído.

A comunicação em sistemas distribuídos geralmente envolve a troca de unidades completas e mais ou menos independentes de informação, como solicitações de procedimentos, respostas a essas solicitações e mensagens entre aplicações, como nos sistemas de enfileiramento de mensagens. Neste tipo de comunicação, o momento específico em que ocorre não afeta a correção, embora possa influenciar a performance do sistema. No entanto, existem formas de comunicação onde o tempo desempenha um papel crucial, especialmente ao lidar com informações dependentes de tempo, como streams de áudio e vídeo.

Para mídias contínuas, como áudio e vídeo, a relação temporal entre os dados é fundamental. Por exemplo, um stream de áudio que utiliza Modulação por Código de Pulso (PCM) para amostras de 16 bits deve reproduzir cada amostra em intervalos exatos para manter a qualidade do som original. Da mesma forma, a representação de movimento em vídeo requer que imagens sucessivas sejam exibidas em espaços uniformes de tempo. A reprodução correta exige não apenas que as imagens sejam mostradas na ordem correta, mas também em uma frequência constante. Essas necessidades diferem significativamente das mídias discretas, como texto ou imagens estáticas, onde as relações temporais entre os itens de dados não são fundamentais para a interpretação correta dos dados.

A comunicação em sistemas distribuídos tem focado na troca de unidades de informação completas e relativamente independentes, como pedidos para invocar procedimentos e mensagens trocadas entre aplicações, como nos sistemas de enfileiramento de mensagens. Este tipo de comunicação não depende de um momento específico para ocorrer. No entanto, existem formas de comunicação onde o tempo é crucial. Um exemplo é um stream de áudio, que requer que amostras sejam reproduzidas em intervalos específicos para manter a fidelidade do som original. Assim, os sistemas distribuídos devem oferecer suporte para a troca de informações dependentes de tempo, como streams de áudio e vídeo.

O suporte para a troca de informações dependentes de tempo é muitas vezes descrito como suporte para mídias contínuas. Mídias contínuas, como vídeo e áudio, exigem que as relações temporais entre os dados sejam preservadas para garantir uma reprodução correta. Isso contrasta com mídias discretas, onde tais relações temporais não são fundamentais. Os streams de dados podem ser discretos, como conexões TCP/IP, ou contínuos, como a reprodução de um arquivo de áudio. Para capturar os aspectos de temporização, distingue-se entre modos de transmissão assíncronos, síncronos e isócronos. Em modos assíncronos, os itens são transmitidos sequencialmente sem restrições de tempo específicas; em modos síncronos, define-se um atraso máximo de ponta a ponta; e em modos isócronos, os dados devem ser transferidos dentro de limites de tempo máximo e mínimo, conhecidos como jitter limitado.

Os fluxos de dados contínuos, ou streams, podem ser simples ou complexos, consistindo em várias substreams relacionadas que também dependem do tempo. Por exemplo, o áudio estéreo é transmitido por meio de duas substreams sincronizadas, garantindo o efeito estéreo. Essa sincronização é crucial e, se falhar, a reprodução também falhará.

Na perspectiva de sistemas distribuídos, vários elementos são necessários para apoiar streams. Concentrando-nos em streams de dados armazenados, ao invés de dados ao vivo, discutimos uma arquitetura cliente-servidor geral para suportar streams multimídia contínuos. Essa arquitetura revela questões importantes como compressão de dados para reduzir a necessidade de armazenamento e capacidade de rede, e o controle da qualidade da transmissão e questões de sincronização.

Requisitos de tempo e outras exigências não funcionais são geralmente expressos como requisitos de Qualidade de Serviço (QoS), que descrevem o que é necessário do sistema distribuído subjacente e da rede para garantir que as relações temporais em um stream sejam preservadas. QoS para streams de dados contínuos geralmente envolve tempo, volume e confiabilidade. Embora muito tenha sido discutido sobre como especificar QoS necessário, na prática, isso frequentemente se resume a especificar propriedades importantes, como a taxa de bits necessária, o atraso máximo até que uma sessão seja estabelecida, o atraso máximo de ponta a ponta, a variância máxima de atraso e o atraso máximo de ida e volta. No entanto, ao lidar com comunicação orientada por stream baseada no protocolo de Internet, devemos aceitar que a base da comunicação é formada por um serviço de datagrama extremamente simples e com esforço mínimo: o IP. Quando a situação se complica, como pode acontecer na Internet, a especificação do IP permite que uma implementação do protocolo descarte pacotes sempre que necessário.

Ao abordar a comunicação em sistemas distribuídos, é crucial considerar os diferentes modos de transmissão para dados contínuos e discretos. No modo assíncrono, os dados são transmitidos sem restrições de tempo específicas, adequados para fluxos de dados discretos. Já o modo síncrono impõe um atraso máximo de ponta a ponta, utilizado, por exemplo, em aplicações de sensoriamento. O modo isócrono, necessário para a transmissão de multimídia distribuída, garante que os dados sejam transmitidos dentro de um intervalo de tempo específico, evitando variações que podem afetar a qualidade da reprodução de áudio e vídeo.

Estes conceitos são fundamentais para o suporte de mídias contínuas em sistemas distribuídos, onde a preservação das relações temporais entre diferentes unidades de dados é essencial para a interpretação correta da mídia, como na reprodução de áudio ou vídeo. Por exemplo, o fluxo de áudio deve ser transmitido de forma que as amostras sejam reproduzidas em intervalos regulares para replicar fielmente o som original. A gestão desses fluxos envolve técnicas avançadas de sincronização e controle de qualidade de serviço para garantir a entrega adequada e oportuna dos dados.

O suporte para a troca de informações dependentes do tempo é frequentemente expresso como requisitos de Qualidade de Serviço (QoS). Estes requisitos descrevem as necessidades do sistema distribuído subjacente e da rede para assegurar que as relações temporais em um fluxo de dados sejam mantidas. Os requisitos de QoS para fluxos de dados contínuos envolvem geralmente pontualidade, volume e confiabilidade, especificando propriedades importantes como a taxa de bits necessária, o atraso máximo até o estabelecimento de uma sessão e a variância máxima de atraso.

Apesar de algumas redes, como a Internet, fornecerem meios para diferenciar classes de dados e priorizar pacotes, as limitações dos serviços de entrega baseados em esforço podem impedir a garantia total de QoS. Mecanismos como buffers são utilizados para reduzir o jitter, armazenando temporariamente os pacotes para permitir uma reprodução regular no receptor. No entanto, problemas como a perda de pacotes podem ocorrer, exigindo técnicas de correção de erros, como a correção de erro para frente (FEC), onde pacotes são codificados de modo que a recepção de qualquer subconjunto suficiente de pacotes permite a reconstrução dos dados.

A sincronização de streams é outro aspecto crítico, especialmente em sistemas multimídia onde streams de vídeo e áudio devem ser precisamente alinhados para garantir, por exemplo, a sincronização labial em vídeos. A sincronização pode ser realizada em vários níveis, desde a sincronização de unidades de dados individuais até a coordenação de streams complexos que envolvem múltiplas substreams. Métodos de sincronização podem ser distribuídos ao longo da rede, com mecanismos implementados tanto no lado do envio quanto do recebimento para garantir que os dados sejam entregues e reproduzidos de forma coerente.

Um tópico importante na comunicação em sistemas distribuídos é o suporte para enviar dados para vários destinatários, conhecido como comunicação multicast. Historicamente, esse tópico tem sido dominado por protocolos de rede, onde várias soluções de nível de rede e de transporte foram implementadas e avaliadas. O desafio majoritário nessas soluções tem sido estabelecer caminhos de comunicação para a disseminação de informações, o que frequentemente envolve um grande esforço de gestão e, em muitos casos, intervenção humana. Além disso, a falta de convergência nas propostas faz com que os provedores de serviços de internet relutem em apoiar o multicast.

Com o surgimento da tecnologia peer-to-peer e, em particular, a gestão estruturada de overlays, tornou-se mais fácil configurar caminhos de comunicação. Como as soluções peer-to-peer são tipicamente implementadas na camada de aplicação, várias técnicas de multicast no nível de aplicação foram introduzidas. A ideia básica no multicast de nível de aplicação é que os nós se organizem em uma rede de overlay, que é usada para disseminar informações entre seus membros. É importante observar que os roteadores de rede não estão envolvidos na gestão da adesão ao grupo. Consequentemente, as conexões entre nós em uma rede de overlay podem cruzar vários links físicos, tornando o roteamento de mensagens dentro do overlay subótima em comparação ao que poderia ser alcançado por meio de roteamento de nível de rede.

Um problema crucial é a construção da rede de overlay. Existem basicamente duas abordagens: os nós podem se organizar diretamente em uma árvore, onde há um caminho único de overlay entre cada par de nós, ou podem organizar-se em uma rede mesh, onde cada nó tem vários vizinhos e, geralmente, existem múltiplos caminhos entre cada par de nós. A principal diferença é que a última geralmente oferece maior robustez; se uma conexão falhar, ainda haverá oportunidade de disseminar informações sem a necessidade de reorganizar imediatamente toda a rede de overlay.

As redes de overlay permitem que informações sejam enviadas a múltiplos destinatários de forma eficiente e gerenciável, apesar dos desafios inerentes na configuração e manutenção dessas redes. Essas redes oferecem um meio crucial para aplicações distribuídas interagirem em larga escala, fornecendo a infraestrutura necessária para uma comunicação eficaz e distribuída em um ambiente globalmente conectado.

Em resumo, a comunicação em sistemas distribuídos é essencial para a funcionalidade e eficiência de redes modernas, abrangendo desde a transferência simples de dados até operações complexas mediadas por camadas sofisticadas do Modelo OSI e sistemas de enfileiramento de mensagens. A padronização dos protocolos através do Modelo OSI e o desenvolvimento de soluções como RPC e MPI são cruciais para garantir a intercompatibilidade e eficácia na transmissão de informações. Os sistemas de enfileiramento de mensagens, por sua vez, facilitam comunicações robustas e flexíveis entre aplicações que operam independentemente do estado ativo dos processos, suportando a integração empresarial complexa e promovendo a escalabilidade e a resiliência em ambientes empresariais. Através da exploração dessas tecnologias e metodologias, os sistemas distribuídos continuam a evoluir, oferecendo soluções inovadoras que endereçam os desafios emergentes em comunicação e processamento de dados em redes globais.

**Referências**

Livro Sistemas Distribuídos Princípios e Paradigmas - Tanembaum (capítulo 4)