Resumo – Arquiteturas Centralizada e P2P.

Os sistemas distribuídos podem ser organizados de várias maneiras. Podemos distinguir entre a arquitetura de software e a arquitetura de sistema. A última considera onde os componentes que constituem um sistema distribuído estão **colocados nas várias máquinas**. A primeira se preocupa mais com a organização lógica do software: **como os componentes interagem**, de que maneiras podem ser estruturados, como podem ser tornados independentes, e assim por diante.

Uma palavra-chave ao falar sobre arquiteturas é **estilo arquitetônico**. Um estilo reflete o princípio básico que é seguido na organização da interação entre os componentes de software que compõem um sistema distribuído. Estilos importantes incluem **camadas, estilos orientados a serviços e estilos em que o tratamento de eventos** é proeminente, exemplificados pelos estilos conhecidos como *publish-subscribe* (publicação-inscrição).

Existem muitas organizações de sistemas distribuídos. Uma classe importante é aquela em que as máquinas são divididas em **clientes e servidores**. Um cliente envia uma solicitação a um servidor, que então produzirá um resultado que será retornado ao cliente. A arquitetura cliente-servidor reflete a forma tradicional de modularizar software, na qual um módulo chama as funções disponíveis em outro módulo. Colocando componentes diferentes em máquinas diferentes, obtemos uma distribuição física natural das funções em uma coleção de máquinas.

As arquiteturas **cliente-servidor são frequentemente altamente centralizadas**. Em arquiteturas **descentralizadas**, vemos muitas vezes um papel igual desempenhado pelos processos que constituem um sistema distribuído, também conhecido como sistemas **peer-to-peer (par-a-par).** Em sistemas peer-to-peer, os processos são organizados em uma rede de sobreposição, que é uma rede lógica na qual cada processo tem uma **lista local de outros pares** com os quais pode se comunicar. A rede de sobreposição **pode ser estruturada**, caso em que esquemas determinísticos podem ser utilizados para rotear mensagens entre processos. Em redes **não estruturadas**, a lista de pares é mais ou menos aleatória, o que implica que **algoritmos de busca** precisam ser utilizados para localizar dados ou outros processos.

Flooding:

Descrição: Cada nó que recebe uma solicitação de busca retransmite essa solicitação para todos os seus vizinhos, exceto para o nó de onde veio a solicitação. Este método é simples, mas pode gerar muito tráfego na rede.

Uso: Comum em redes não estruturadas, como Gnutella.

Random Walk:

Descrição: Em vez de enviar a solicitação para todos os vizinhos, o nó encaminha a solicitação para um vizinho selecionado aleatoriamente. Este processo pode ser repetido por um número definido de saltos.

Vantagens: Reduz o tráfego em comparação com o flooding, mas pode ser mais lento.

Uso: Adequado para redes não estruturadas.

Iterative Deepening:

Descrição: Este método combina o flooding com uma limitação de profundidade, começando com um pequeno limite e aumentando gradualmente até que o objeto seja encontrado.

Vantagens: Pode reduzir o tráfego ao limitar inicialmente o número de nós a serem contatados.

Routing Indices (RIs):

Descrição: Cada nó mantém índices de roteamento que fornecem informações sobre a localização aproximada dos dados. Ao receber uma solicitação de busca, o nó encaminha a solicitação para o vizinho que é mais provável de possuir o dado desejado.

Vantagens: Mais eficiente em redes estruturadas.

Uso: Implementado em redes estruturadas.

Kademlia:

Descrição: Kademlia usa um esquema de roteamento baseado em XOR para localizar nós e dados na rede. Cada nó é identificado por um ID e as solicitações são encaminhadas para os nós que têm IDs mais próximos do ID do dado solicitado.

Vantagens: Escalável e eficiente, usado em muitas redes P2P modernas.

Uso: Utilizado em redes como a BitTorrent DHT.

Chord:

Descrição: Chord organiza nós em um anel lógico onde cada nó é responsável por um intervalo de chaves. A busca é realizada navegando no anel até encontrar o nó que possui a chave desejada.

Vantagens: Proporciona balanceamento de carga e é eficiente em termos de saltos de rede.

Uso: Usado em sistemas distribuídos e DHTs (Distributed Hash Tables).

Pastry:

Descrição: Pastry organiza os nós em uma estrutura semelhante a uma árvore binária, onde a busca é feita encaminhando a solicitação para o nó que possui a chave mais próxima da chave desejada.

Vantagens: Oferece uma boa distribuição de carga e é eficiente em redes de grande escala.

Uso: Utilizado em sistemas DHT como o PAST.

CAN (Content Addressable Network):

Descrição: CAN organiza nós em um espaço de chave multi-dimensional, onde cada nó é responsável por uma zona deste espaço. A busca é realizada navegando por este espaço até encontrar o nó que possui a chave.

Vantagens: Simples e eficiente, com bom balanceamento de carga.

Uso: Adequado para redes estruturadas.