UNIP

UNIVERSIDADE PAULISTA

Sistemas Distribuídos

Autor: Prof. Michel Bernardo Fernandes da Silva
Colaboradoras: Profa. Vanessa Santos Lessa
Profa. Larissa Rodrigues Damiani

Professor conteudista: Michel Bernardo Fernandes da Silva

Formado em Engenharia Elétrica opção Telecomunicações pela Escola Politécnica da Universidade de São Paulo (Poli-USP) e mestre em Engenharia Elétrica pela mesma instituição. Concluiu os cursos de Pós-Graduação em Finanças e MBA Executivo no Insper. Atualmente, está cursando doutorado em Engenharia Elétrica na Poli-USP e desde 2013 é professor dos cursos superiores de tecnologia em diversos *campi* da Universidade Paulista (UNIP). Possui experiência profissional de mais de 12 anos na área financeira e já atuou como professor universitário e de pós-graduação em diversas instituições como FMU, Uninove e Anhanguera. É autor do livro *Cibersegurança: uma visão panorâmica de segurança de informação na internet*.

Dados Internacionais de Catalogação na Publicação (CIP)

S586s Silva, Michel Bernardo Fernandes da.

Sistemas Distribuídos / Michel Bernardo Fernandes da Silva. – São Paulo: Editora Sol, 2024.

112 p., il.

Nota: este volume está publicado nos Cadernos de Estudos e Pesquisas da UNIP, Série Didática, ISSN 1517-9230.

1. Sistemas distribuídos. 2. Arquitetura. 3. Cluster. I. Título.

CDU 681.3.021

U519.97 – 24

Sumário

Sistemas Distribuídos

APRESENTAÇÃO

O objetivo geral desta disciplina é a apresentação dos principais aspectos relacionados ao projeto e à implementação de um sistema distribuído. Nela, veremos quais as funções que um sistema operacional deve desempenhar em um sistema de computação, tais como gerenciamento do processador, gerenciamento de memória, memória virtual, sistemas de arquivos e sistemas de E/S, aspectos de segurança, entre outros.

Um objetivo específico desta disciplina é acentuar as principais características que o sistema operacional deve ter, os conceitos sobre sistemas operacionais, a arquitetura desses sistemas, os algoritmos utilizados nas diferentes atividades de gerência e formas de implementação para esses sistemas. Também são objetivos específicos a apresentação e a análise detalhada do sistema distribuído chamado cluster Beowulf.

Você consegue entender como é possível nos comunicarmos e executarmos serviços que são utilizados por milhares de usuários simultaneamente, tais como e-mails, sites de e-commerce e jogos online? São sistemas distribuídos que realizam essas operações de modo a oferecer maior agilidade, disponibilidade e flexibilidade para seus usuários.

As redes de computadores se espalharam por toda parte. A internet é uma delas, e certamente você utiliza outra: as redes de telefones móveis. Nas empresas, as redes corporativas interligam suas diversas áreas. Através de redes domésticas, é possível criar casas inteligentes com acionamento a distância de lâmpadas ou outros equipamentos eletrônicos, como ar-condicionado. Em fábricas industriais, existem diversas redes que interligam cada um dos componentes necessários para a produção do bem.

Até os veículos atuais são capazes de estabelecer redes com equipamentos multimídia e entre seus sistemas. Cada uma dessas redes compartilham as características básicas que as tornam assuntos relevantes para estudos denominados sistemas distribuídos.

O foco desta disciplina envolve as técnicas aplicadas, e não as tecnologias específicas, que podem ser substituídas por outras. Já as técnicas continuam sendo aplicadas mesmo com a substituição de tecnologia.

A área de sistemas distribuídos tem crescido muito rapidamente nos últimos anos em função de serviços web, aplicações par-a-par (ou peer-to-peer) e redes de sensores sem fio. Uma aplicação de sistemas distribuídos ocorre nas criptomoedas ou moedas digitais, sendo a Bitcoin a mais conhecida delas. Para as criptomoedas, é utilizada a tecnologia de Blockchain, que consolida um conjunto de informações que são conectadas através de criptografia. Dessa forma, transações financeiras como débitos e créditos e outras operações podem ser realizadas com segurança, com precisão e por milhares de usuários concorrentemente.

INTRODUÇÃO

Ao andar pelo centro de uma cidade, podemos ver diversos locais que possuem redes de computadores nas quais os dispositivos de computação estão interconectados de forma a compartilhar dados e recursos.

Além da internet, existem diversos outros tipos de redes, como a rede de telefonia móvel, redes em uma universidade, redes domésticas, entre outras. Elas têm características básicas de uma categoria denominada sistemas distribuídos.

Segundo Coulouris, Dollimore e Kindberg (2013, p. 1), "um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens".

Por sua vez, Tanenbaum e Steen (2007) elaboraram outra definição bastante difundida que conceitua o sistema distribuído como um conjunto de computadores independentes que se comporta como se fosse um único computador. Essa definição traz dois aspectos essenciais para o sistema computacional: do ponto de vista do hardware, as máquinas operam de forma autônoma e independente, e analisando o software, os usuários enxergam o sistema como uma única máquina.

Este livro-texto está dividido em três unidades. Na primeira, são abordados a definição, o histórico, os tipos e as arquiteturas de sistemas distribuídos. Na segunda, são descritos os conceitos e as técnicas aplicáveis à gerência de processos, tais como a comunicação e a sincronização entre processos. Outros tópicos dessa unidade são a nomenclatura, a confiabilidade e tolerância a falhas e a segurança de sistemas distribuídos. Na terceira, são apresentados os clusters de computadores e detalhes sobre seu projeto e gerenciamento. Haverá um detalhamento do cluster Beowulf. Adicionalmente, são mostrados os conceitos de programação paralela e computação em grade ou grid computing.

Aproveite a leitura e bons estudos!

Unidade I

1 O QUE SÃO SISTEMAS DISTRIBUÍDOS

Neste tópico, serão acentuados o conceito de sistemas distribuídos, o histórico, as características, os tipos e as arquiteturas desse tipo de sistema computacional.

1.1 Conceito de sistemas distribuídos

Um sistema distribuído (SD) pode ser entendido como uma coleção de computadores independentes que aparenta a seus usuários ser um sistema único e coerente. Neste conceito, há dois aspectos importantes: os computadores são independentes e os processos do sistema estão distribuídos em diferentes computadores que se comunicam através de redes de comunicação.

Assim, um sistema distribuído é constituído por diversas partes, nas quais diferentes programas estão sendo executados em computadores separados, cujos programas podem ser escritos em qualquer linguagem de programação e rodam sistemas operacionais diversos, e as partes desses sistemas se comunicam entre si de forma a interagir internamente, visando oferecer funcionalidades aos usuários.

A principal razão para construir e utilizar sistemas distribuídos decorre da aspiração de compartilhamento de recursos para usuários e para aplicações. O motivo crucial para o compartilhamento de recursos é a redução de custos, pois é mais barato comprar um sistema de armazenamento confiável e de qualidade a ser compartilhado do que comprar e manter o armazenamento para cada usuário individualmente. Por exemplo, em um coworking no qual existem diversas pequenas empresas, é realizado um contrato de serviço de internet único que será mais barato do que fazer um contrato individual para cada empresa. Como há diferentes padrões de comportamento de uso da rede, na maior parte do tempo os usuários não estarão utilizando a capacidade total da banda de internet contratada.

Com isso, uma das características mais relevantes para um sistema distribuído é sua abstração, porque os usuários não sabem em qual dispositivo está sendo executada uma parte específica do programa ou aplicação. A complexidade de um sistema distribuído pode variar bastante, sendo um exemplo simples uma impressora compartilhada em rede. Por outro lado, um sistema distribuído complexo pode ser composto de aglomerados de computadores dedicados à execução de uma aplicação em larga escala.

Os desafios para o desenvolvimento e a implantação de sistemas distribuídos decorrem de suas particularidades, tais como disponibilidade de recursos, abstração, sistema aberto e escalabilidade.

Um sistema distribuído pode ser bastante heterogêneo, já que pode ser composto de computadores de diferentes empresas, diversos sistemas operacionais e variadas plataformas. Uma forma de possibilitar

a operação com redes e computadores heterogêneos em sistemas distribuídos é a construção de middlewares, que operam como uma camada de abstração e possibilitam maior interoperabilidade. Assim, o middleware representa uma interface padronizada que ficará em uma camada superior ao sistema operacional.

A figura a seguir ilustra a organização em camadas:

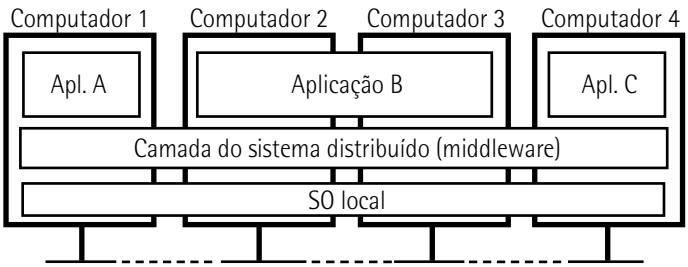

Figura 1 – Sistema distribuído como middleware

Fonte: Tanenbaum e Steen (2007, p. 4).

A figura ilustra quatro computadores interligados por rede e três aplicações, sendo que a aplicação B está distribuída entre os computadores 2 e 3. É oferecida a mesma interface para cada aplicação. De acordo com Tanenbaum e Steen (2007), o sistema distribuído possibilita formas para que os elementos de uma única aplicação distribuída possam se comunicar entre si e que diferentes aplicações se comuniquem. Simultaneamente, o sistema distribuído retira detalhes das diferenças entre o hardware e sistemas operacionais para cada aplicação.

Middleware é uma camada de software intermediária entre as aplicações que estão em diferentes computadores e os sistemas operacionais locais instalados nos computadores. Os serviços web (ou web services), o RMI Java e o CORBA são exemplos de middleware.

Representa-se o middleware por meio de processos ou objetos em um conjunto de máquinas que interagem entre si para suportar a comunicação e o compartilhamento de recursos em sistemas distribuídos. O objetivo do middleware é prover elementos para a construção de componentes de software que tenham interação em um sistema distribuído.

Lembrete

Middleware é uma camada de software intermediária entre as aplicações que estão em diferentes computadores e os sistemas operacionais locais instalados nos computadores. Os serviços web (ou web services), o RMI Java e o CORBA são exemplos de middleware.

Uma plataforma tem camadas de hardware e software de níveis mais baixos. As plataformas fornecem serviços para camadas que estão colocadas acima delas. Seu objetivo é entregar uma interface

de programação do sistema de forma a facilitar a comunicação e a coordenação entre processos. São exemplos de plataformas: PowerPC/Mac OS, Intel x86/Linux, Intel x86/Windows e ARM/Symbian.

Nos primeiros anos após a invenção dos sistemas computacionais até a década de 1980, os computadores eram dispositivos de grandes dimensões físicas e bastante caros: minicomputadores custavam dezenas de milhares de dólares. Assim, poucas empresas possuíam apenas alguns computadores e eles funcionavam de forma independente entre si, pois não havia uma forma de conectá-los. Esses sistemas eram chamados de sistemas centralizados e eram compostos de um único computador, seus dispositivos de E/S e, eventualmente, terminais remotos dos usuários.

A partir dos anos 1980, houve dois avanços tecnológicos que permitiram a mudança dessa situação. O primeiro, como resultado da evolução de microeletrônico, ocorreu com o desenvolvimento de poderosos microprocessadores, uma evolução que permitiu o aumento do poder computacional com redução significativa do preço em relação aos mainframes. Já a segunda melhoria foi a invenção de uma rede de computadores de alta velocidade. Redes locais – em inglês, local-area networks (LANs) – possibilitam que milhares de computadores dentro de um prédio estejam conectados de forma que pequenos volumes de dados possam ser enviados em microssegundos, sendo a velocidade da rede medida em bilhões de bits por segundo (bps). Para interligar milhares de dispositivos que estão em diferentes localidades em todo o mundo, são usadas redes globais ou wide-area networks (WANs).

Observação

Sistemas distribuídos são diferentes do software tradicional, pois os componentes estão dispersos por uma rede. Desconsiderar essa dispersão no projeto desses sistemas resulta em erros que deverão ser corrigidos posteriormente.

Adicionalmente, os sistemas distribuídos oferecem maior funcionalidade em relação aos centralizados, já que permitem aplicações que não seriam possíveis em sistemas centralizados. Um exemplo de aplicação que somente é possível em um sistema distribuído é um aplicativo de mensagens instantâneas como o WhatsApp, pois para se comunicar cada pessoa precisa ter o aplicativo em seu dispositivo.

Outra vantagem de um sistema distribuído sobre os sistemas centralizados é a maior confiabilidade dos sistemas distribuídos. Como existe a distribuição de carga entre diversas máquinas, se um único dispositivo falhar, a operação continuará intacta.

1.2 Características de sistemas distribuídos

Um traço comum de sistemas distribuídos é a execução concorrente de processos. Em sistemas distribuídos, a coordenação de programas em execução concorrente e que compartilham recursos também é um assunto importante e regular, já que é possível um usuário trabalhar em seu computador e compartilhar recursos como arquivos ou páginas web, quando necessário.

Um recurso pode ser um componente de hardware, como discos e impressoras, ou elementos definidos pelo software, como arquivos, bancos de dados e objetos de dados de todos os tipos.

Os elementos de um sistema distribuído têm falhas independentes. Mesmo que um de seus componentes falhe, os outros ainda estarão em funcionamento, pois a falha deste não é imediatamente percebida pelos outros componentes com os quais se comunica. Falhas na rede resultam no isolamento dos computadores que estão conectados a ela, mas isso não significa que eles param de funcionar.

Quando os programas precisam de cooperação mútua, eles coordenam suas ações através da troca de mensagens, sendo necessária uma noção de tempo para ordená-la. Existe uma precisão limite para o sincronismo entre computadores em uma rede, e não existe suposição de relógio global. É essencial que o projetista do sistema esteja ciente de que é impossível que todos os dispositivos sejam sincronizados no mesmo horário e de que quanto maior o sistema, maior será a incerteza em relação ao relógio.

Um sistema distribuído aberto é aquele que disponibiliza serviços segundo regras padronizadas, as quais descrevem a sintaxe e a semântica de tais serviços. Por exemplo, as redes de computadores têm regras padronizadas para regular o formato, o conteúdo e o significado de mensagens transmitidas e recebidas. A formalização dessas regras nas redes de computadores ocorre através dos protocolos. Para sistemas distribuídos em geral, os serviços são especificados por meio de interfaces, descritas por meios de linguagens de definição de interface ou interface definition language (IDL), em inglês.

Por sua vez, um sistema distribuído usa algoritmos descentralizados, e cada máquina irá executar esse algoritmo, para que sejam capazes de tomar decisões com base em informações locais. Assim, nenhuma máquina terá a informação completa do estado do sistema. A falha do algoritmo em uma das máquinas não irá acarretar a queda do sistema, pois as outras que são independentes ainda estarão operando.

1.2.1 Escalabilidade

Outra característica que se aplica a sistemas distribuídos é a escalabilidade, que mede a capacidade do sistema em lidar facilmente com uma quantidade crescente de trabalho. Existem formas diferentes de medir a escalabilidade, tais como:

- número de processos e/ou usuários utilizados simultaneamente, que são métricas da escalabilidade de tamanho;
- distância máxima entre os nós, que representa a escalabilidade geográfica;
- número de domínios administrativos, isto é, diferentes redes corporativas, que mostra a escalabilidade administrativa.

Quando um sistema precisa ser escalável, surgem diferentes tipos de problemas que necessitam ser resolvidos. Em muitos serviços é adotada a arquitetura centralizada, e esse serviço é implementado por um único servidor que o executa em uma máquina específica. Em uma abordagem mais moderna,

pode haver um grupo de servidores colaborativos agrupados em um cluster de máquinas fracamente acopladas e fisicamente instaladas na mesma localidade.

Do ponto de vista administrativo, o sistema de uma empresa deve ser escalável. Quando as organizações crescem, é fundamental que o sistema tenha capacidade de suportar a nova demanda e que continue sendo fácil de gerenciar.

Há um gargalo potencial, que é como o servidor ou grupo de servidores irão se comportar com o aumento do número de pedidos para seus processos. As causas geradoras para o gargalo são a capacidade computacional, limitada pelas CPUs, e a capacidade de armazenamento, que inclui as taxas de transferência entre dispositivos de E/S. Mesmo que seja utilizado o computador mais avançado, com o aumento do número de usuários e processos, o sistema sofrerá alguns problemas.

Um exemplo típico dessa limitação de escalabilidade é um motor de pesquisas centralizado mal projetado. Nesse problema, é necessário combinar um texto a ser pesquisado em um conjunto de dados carregados. Mesmo com a utilização de técnicas avançadas de indexação, ainda persiste o problema da necessidade de processamento de uma grande quantidade de dados que excedem a capacidade de memória principal da máquina que está rodando o serviço. Como consequência, a maior parte do tempo de processamento será determinado pelo lento acesso ao disco e transferência de dados entre memória e disco. Mesmo que sejam adicionados discos de alta velocidade, ainda haverá muitos problemas com o aumento do número de requisições.

Além dos serviços centralizados, dados centralizados como uma única lista online ou algoritmos centralizados, como fazer o roteamento utilizando informações completas, estão sujeitos às limitações de escalabilidade.

1.2.2 Transparência

Um objetivo fundamental de um sistema distribuído é esconder do usuário final e das aplicações o fato de que os processos e recursos estão fisicamente distribuídos através de múltiplos computadores, os quais possivelmente estão separados por grandes distâncias geográficas. A transparência significa que para o usuário final é como se tivesse realizado em um único computador. Dessa forma, é possível compartilhar recursos, como um servidor.

O manual de referência Ansa (1989) trata de arquitetura de sistemas de redes avançadas e definiu oito formas de transparência.

A transparência de acesso permite que recursos locais e remotos sejam acessados com o uso de operações idênticas.

Com a transparência de localização, permite-se que os recursos sejam acessados sem conhecimento de sua localização física, edificação ou rede, através do endereço IP presente nas mensagens.

A transparência de concorrência possibilita que diversos processos operem concorrentemente, usando recursos compartilhados sem interferência entre eles.

Por meio da transparência de replicação, criam-se várias instâncias dos recursos para que sejam usadas para aumentar a confiabilidade e o desempenho, sem conhecimento das réplicas por parte dos usuários ou dos programadores de aplicativos.

Já a transparência de falhas permite a ocultação de falhas, possibilitando que usuários e programas aplicativos concluam suas tarefas, mesmo com falhas em outros componentes de hardware ou software do sistema distribuído.

A transparência de mobilidade existe para possibilitar a movimentação de recursos e clientes dentro de um sistema, sem afetar a operação de usuários ou de programas.

Para permitir que o sistema seja reconfigurado para melhorar o desempenho à medida que as cargas variam, existe a transparência de desempenho.

Por fim, a transparência de escalabilidade permite que o sistema e os aplicativos se expandam em escala, sem alterar a estrutura do sistema ou os algoritmos de aplicação.

Observação

As duas transparências mais importantes são a de acesso e a de localização; sua presença ou ausência afeta mais fortemente a utilização de recursos distribuídos. Às vezes, elas são referidas em conjunto como transparência de rede.

1.2.3 Qualidade de serviço

Segundo a ISO 13236 (1995), a qualidade de serviço ou quality of service (QoS) pode ser definida como o efeito coletivo do desempenho de um serviço que determina o grau de satisfação de um usuário do serviço. Após definir a funcionalidade necessária de um serviço, deve-se questionar a qualidade do serviço fornecido. Por exemplo, considere o serviço de provedor de internet. Quando a velocidade de download estiver baixa ou quando o filme que o usuário está assistindo fica travando, o usuário ficará insatisfeito e a qualidade do serviço de provimento de internet ficará deteriorada. Adicionalmente, se o usuário estiver sem internet, a qualidade de serviço será impactada.

De forma geral, as principais propriedades não funcionais dos sistemas que influenciam a qualidade do serviço experimentada pelos clientes e usuários são a confiabilidade, a segurança e o desempenho.

Os problemas de confiabilidade e de segurança são fundamentais no projeto da maioria dos sistemas computacionais. O aspecto do desempenho da qualidade de serviço foi definido originalmente em

termos da velocidade de resposta e do rendimento computacional, mas foi redefinido em termos da capacidade de satisfazer garantias de pontualidade.

A adaptabilidade para satisfazer as configurações de sistema variáveis e a disponibilidade de recursos tem sido reconhecida como um aspecto muito importante da qualidade do serviço.

1.3 Exemplos de sistemas distribuídos

Existem diversos exemplos de sistemas distribuídos, tais como a internet, pesquisas na web, o serviço de e-mail, o comércio eletrônico ou e-commerce, as redes sociais, o mobile banking, os serviços de streaming, bancos de dados em nuvem, entre outros.

Normalmente, são utilizados sistemas operacionais de rede como Windows, Mac OS e Linux, que permitem aos usuários a execução de forma independente dos demais recursos da rede. Entretanto, não existem sistemas operacionais para uso geral por dois motivos principais. O primeiro deles é o volumoso investimento realizado em software aplicativo, que será executado em um sistema operacional já existente. Adicionalmente, os usuários tendem a preferir autonomia em suas máquinas.

1.3.1 Jogos online

A indústria de jogos online ou massively multiplayer online games (MMOG) cresceu exponencialmente nos últimos anos e oferece uma experiência imersiva com a qual uma grande quantidade de usuários tem interação com o mundo virtual persistente pela internet. Um grave problema desse tipo de aplicação distribuída é a latência de rede via internet. Esses atrasos nos jogos causam a insatisfação dos usuários quando a imagem congelar ou houver pequenas pausas.

Essa dificuldade é agravada pelo fato de vários usuários estarem jogando simultaneamente, pois esse número pode chegar a centenas ou milhares de pessoas. Além disso, essas aplicações promovem eventos em tempo real para possibilitar que todos os jogadores tenham uma visão coerente e realista do cenário do jogo. São exemplos desses tipos de aplicação o EVE Online, da empresa finlandesa CCP Games, o EverQuest II, da Sony, e o Call of Duty: Modern Warfare, da Infinity Ward.

De acordo com Coulouris, Dollimore e Kindberg (2013), o jogo EVE Online foi construído com base no paradigma cliente-servidor, no qual uma única cópia do estado do jogo permanece armazenada em um servidor enquanto os clientes acessam essa base regularmente para se manter atualizados. No caso, o servidor é composto de um agregado de computadores, isto é, um cluster. A utilização desse modelo para a construção da aplicação possui vantagens no gerenciamento, e a cópia única facilita a implementação da coerência. Com isso, o principal dilema nesse caso é a otimização na comunicação, para garantir qualidade na entrega do serviço e propagação dos eventos em tempo real. Os autores especificam que existem sistemas estelares individuais, alocados para o particionamento da carga entre computadores que fazem parte do cluster. Esses sistemas sobrecarregados têm um computador dedicado, enquanto os outros compartilham computadores.

Saiba mais

O artigo a seguir apresenta modelos de distribuição, com o objetivo de disponibilizar um jogo com custo aceitável por empresas de menor porte de desenvolvimento de jogos:

BEZERRA, C. E.; CECIN, F.; GEYER, C. F. R. *Modelos de suporte distribuído para jogos online maciçamente multijogador*. Porto Alegre: Universidade Federal do Rio Grande do Sul, 2009. Disponível em: https://tinyurl.com/yu85j72h. Acesso em: 22 abr. 2024.

1.3.2 Pesquisas na web

A tarefa de pesquisa na web parece extremamente simples para o usuário final. Para realizar essa atividade, é necessário um mecanismo de pesquisa para indexar todo o conteúdo da rede world wide web, que inclui diversos tipos de informações diferentes, como páginas web, livros digitais, vídeos, imagens, mapas, documentos em PDF, entre outros.

A complexidade dessa atividade está na quantidade de páginas existentes na web, que chega a centenas de bilhões, com mais de um trilhão de endereços únicos. Realizar um processamento nesse banco de dados gigantesco representa um desafio para o projeto de sistemas distribuídos.

O Google, líder de mercado em tecnologia de pesquisa na web, adotou o projeto de uma sofisticada infraestrutura de sistema distribuído que oferece suporte à pesquisa. Isso representa uma das maiores e mais complexas instalações de sistemas distribuídos da história da computação e, assim, exige um exame minucioso. Os destaques incluem a infraestrutura física, com inúmeros computadores interligados em redes, um sistema de arquivos distribuído, um serviço de bloqueio e um modelo de programação que permite cálculos paralelos e distribuídos.

Saiba mais

O site Google Data centers acentua a infraestrutura dos data centers do Google:

GOOGLE. *Sobre os data centers do Google*. [s.d.]. Disponível em: https://tinyurl.com/2abay42p. Acesso em: 12 abr. 2024.

Adicionalmente, o vídeo de introdução à infraestrutura do Google Cloud apresenta uma visão geral da nuvem do Google ou Google Cloud Plataform (GCP).

INTRODUÇÃO à infraestrutura de Google Cloud. 2021. 1 vídeo (21 min). Publicado pelo canal Google Cloud LATAM. Disponível em: https://tinyurl.com/34vata9p. Acesso em: 12 abr. 2024.

1.3.3 Bancos de dados distribuídos

A distribuição dos dados, processamento e armazenamento oferece algumas vantagens e desafios. Os principais fornecedores de sistemas gerenciadores de banco de dados (SGBDs) de mercado têm soluções distribuídas. SGBD é um sistema computacional que tem a função de armazenar, organizar, indexar e permitir acesso aos bancos de dados que estão interligados a ele.

A origem dos SGBDs distribuídos foi motivada, principalmente, após a internacionalização de grandes empresas, que tinha uma matriz e diversas filiais. Nesse contexto, surgiram alguns problemas referentes aos armazenamentos dos dados da matriz e das filiais.

Lembrete

SGBD é um sistema computacional que tem a função de armazenar, organizar, indexar e permitir acesso aos bancos de dados que estão interligados a ele.

Os bancos de dados distribuídos foram aplicados, nesse contexto, como solução para o aumento da disponibilidade dos dados, para manter a proximidade de uma filial com seus dados, facilitar a escalabilidade dos dados, incluir novas filiais e eliminar duplicação desnecessária de dados devido aos problemas decorrentes de empresas distribuídas geograficamente por vários países.

A figura a seguir mostra a arquitetura de banco de dados distribuídos, na qual quatro filiais (São Paulo, Rio de Janeiro, Campo Grande e São José do Rio Preto) de uma empresa estão conectadas por redes de computadores e utilizam banco de dados distribuídos.

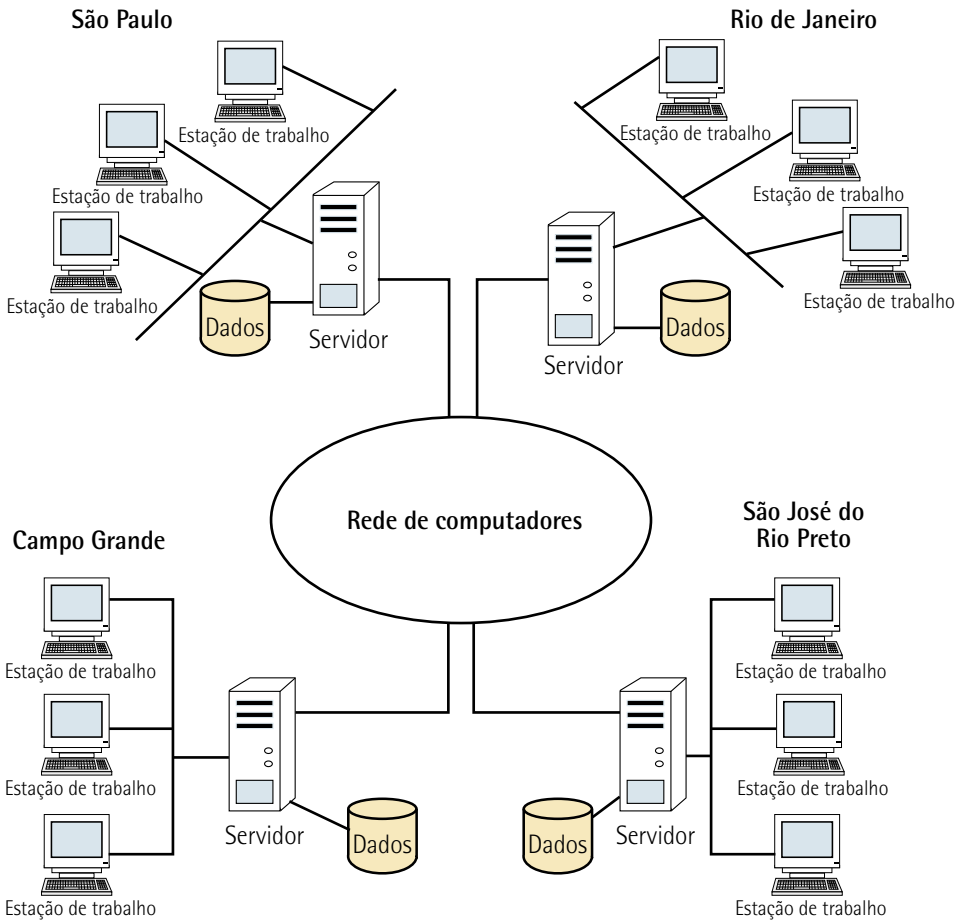

Figura 2 – Arquitetura de banco de dados distribuídos

1.3.4 Computação em nuvem ou cloud computing

Nas últimas duas décadas, as empresas comerciais têm procurado alternativas para manter em operação data centers próprios com o objetivo de oferecer seus recursos aos clientes. Com a crescente utilização de sistemas distribuídos, algumas empresas de tecnologia, como Google, Amazon e Huawei, estão promovendo o conceito dos recursos distribuídos, como uma commodity ou um serviço público, de forma análoga a serviços públicos, como água, gás encanado e eletricidade. Nesse modelo, os recursos são supridos por fornecedores de serviço apropriados e efetivamente alugados, em vez de pertencerem ao usuário final.

De acordo com Coulouris, Dollimore e Kindberg (2013), uma nuvem é entendida como um conjunto de serviços de aplicativo, armazenamento e computação baseados na internet, suficientes para suportar as necessidades da maioria dos usuários, permitindo, assim, que eles precisem, em grande medida ou totalmente, do software local de armazenamento de dados ou de aplicativo.

Desde 2006, a empresa Amazon começou a oferecer serviços de infraestrutura de TI por meio de serviços web e denominou seu serviço como Amazon Web Services. Segundo a AWS (2021), a computação em nuvem pode ser definida como fornecimento de aplicativos, armazenamento de banco de dados,

capacidade computacional e outros recursos computacionais virtualizados sob demanda por meio de uma plataforma de serviços em nuvem na internet, com definição de preço e com pagamento de acordo com a utilização. Esse provedor de serviço garante que a operação respeite uma série de requisitos customizados denominados **nível de serviço acordado** ou service level of agreement (SLA).

Esse tipo de sistema é composto de recursos físicos e serviços de software. Os recursos físicos podem ser de armazenamento ou de processamento e estão conectados à internet de modo a se tornarem disponíveis para clientes ligados em rede, eliminando a necessidade de utilizá-los nos computadores clientes. O serviço de armazenamento remoto pode ser usado como um backup ou para guardar arquivos pessoais, para que possam ser acessados por vários dispositivos conectados à conta do usuário, como ocorre nos serviços do Google Drive, Dropbox e Microsoft OneDrive.

Caso os serviços de processamento sejam necessários, os usuários podem ter acesso a data centers, isto é, recursos ligados em rede que têm acesso a extensos repositórios de dados utilizados por empresas ou à infraestrutura computacional de empresas como Google e Amazon.

A figura a seguir apresenta os principais elementos da computação em nuvem ou cloud computing:

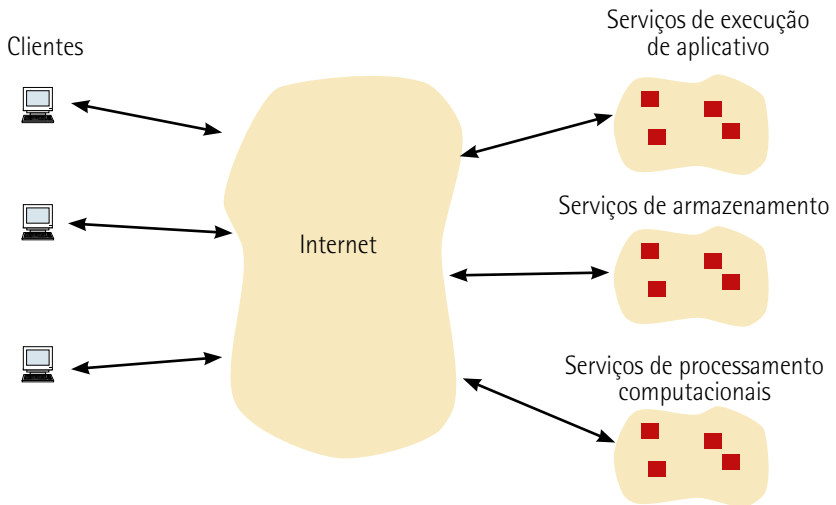

Figura 3 – Computação em nuvem

Fonte: Coulouris, Dollimore e Kindberg (2013, p. 14).

Saiba mais

O site da AWS oferece uma série de materiais explicativos, treinamentos virtuais e preparativos para certificação para operações com a AWS.

Disponível em: https://aws.amazon.com/pt/training/. Acesso em: 21 abr. 2024.

Pode-se usar uma estrutura em camadas com cinco componentes: aplicação, plataforma, infraestrutura virtual, virtualização e data centers com servidores e serviços de armazenamento. A figura a seguir ilustra essa estrutura em camadas:

Quadro 1 – Estrutura em camadas de computação em nuvem

Estrutura em camadas
Aplicação
Plataforma
Infraestrutura virtual
Virtualização
Servidores/Armazenamento/Datacenters

Fonte: Silva *et al.* (2020, p. 81).

A camada de aplicação é utilizada pelos usuários finais, e os clientes têm acesso às aplicações da nuvem com uma interface adequada. Através dessa camada, os usuários finais da internet obtêm as aplicações executadas pela estrutura da nuvem.

Na camada plataforma, estão especificados serviços como middleware, conectividade e integração. Os recursos dessa camada, a qual se localiza entre as camadas de aplicação e infraestrutura, são responsáveis por executar as aplicações.

A camada estrutural para o funcionamento da computação em nuvem é a infraestrutura, já que todos os serviços são disponibilizados através de uma infraestrutura completa. Com isso, o usuário tem o controle dos recursos virtualizados, com exceção dos dispositivos físicos do data center.

A infraestrutura é composta de diversos recursos, provendo a capacidade computacional necessária para o usuário e suas aplicações. Os principais recursos definidos nessa camada são servidores, elementos de rede, armazenamento, equipamentos de refrigeração e outros recursos de computação fundamentais. Esses recursos possibilitam guardar, manipular e enviar os dados através da internet, para garantir a operação dos serviços como um todo e controlar o ambiente por meio de uma interface única.

A virtualização é uma operação essencial para a computação em nuvem, pois retira a limitação existente entre aplicativos e servidores físicos. Isso ocorre devido aos servidores virtuais expandirem os recursos computacionais existentes. Adicionalmente, a virtualização permite gerenciar de forma centralizada os diversos servidores virtuais hospedados em um mesmo servidor físico.

A virtualização permite maximizar os recursos, reduzindo a necessidade de aquisição de equipamentos físicos, e impede a subutilização de recursos que ocorreria em uma infraestrutura física comum.

Analogamente à infraestrutura tradicional, o data center na nuvem tem a responsabilidade de processar dados e de armazenar os ativos de rede e dos sistemas que armazenam dados e informações do negócio, providos na forma de infraestrutura. Os serviços fornecidos pelo data center são:

- serviços de processamento;
- armazenamento;
- serviços de rede;
- serviços de segurança;
- serviços de aplicação;
- serviços de alta disponibilidade.

O provedor da nuvem oferece os recursos computacionais com três modelos de serviços diferentes:

- software como serviço ou software-as-a-service (SaaS);
- plataforma como serviço ou platform-as-a-service (PaaS);
- infraestrutura como serviço ou infrastructure-as-a-service (IaaS).

Software como serviço (SaaS)

Na abordagem software como serviço, o provedor da nuvem fornece aos clientes a utilização de aplicações em uma infraestrutura de nuvem, que pode ser acessada através de dispositivos conectados à internet. O usuário não controla a infraestrutura nem gerencia a nuvem. A rede, os servidores, os sistemas operacionais e o armazenamento são controlados pelo provedor da nuvem. O software também é desenvolvido pelo provedor de servidor. São exemplos de SaaS a Salesforce.com, que é uma solução para gerenciar o relacionamento entre empresas e clientes, e o serviço de e-mail Gmail.

Os aplicativos SaaS também são denominados softwares baseados na web, softwares hospedados ou sob demanda. Tais aplicativos rodam nos servidores das empresas provedoras, que são responsáveis pelo gerenciamento de acesso, manutenção da estrutura de segurança de dados, conectividade e servidores necessários para implementação do serviço.

Plataforma como serviço (PaaS)

No modelo de PaaS, as aplicações executadas como infraestrutura da nuvem são desenvolvidas ou adquiridas pelos usuários, partindo de linguagens de programação, ferramentas, bibliotecas e serviços suportados pelo fornecedor. Assim como no modelo SaaS, no PaaS o cliente não terá controle nem gerenciamento da infraestrutura da nuvem, que inclui redes, servidores, armazenamento, sistemas operacionais e, eventualmente, as configurações necessárias para o ambiente de hospedagem de aplicativos (Silva *et al.*, 2020). Como exemplos de provedores de PaaS, existem o Google App Engine e a Microsoft Azure Services.

Infraestrutura como serviço (IaaS)

No modelo IaaS, o cliente deve prover processamento, armazenamento, redes e outros recursos computacionais essenciais, para que sejam implantados e executados os aplicativos necessários para operações desse usuário. O usuário tem o controle sobre aplicativos instalados, sistemas operacionais e armazenamentos e, eventualmente, controla de forma limitada os componentes de rede, como firewalls de host. Entretanto, o consumidor não é responsável pelo gerenciamento e pelo controle da infraestrutura de nuvem subjacente. São exemplos de provedor IaaS a Amazon Web Services (AWS) e a Huawei Cloud.

De forma resumida, o IaaS apenas provê recursos computacionais. No PaaS, além dos recursos computacionais, são fornecidas tecnologias. No SaaS, são concedidas adicionalmente as ferramentas para desenvolvimento e execução dos serviços implementados.

A figura a seguir mostra uma visão em camadas dos modelos de serviço da computação em nuvem:

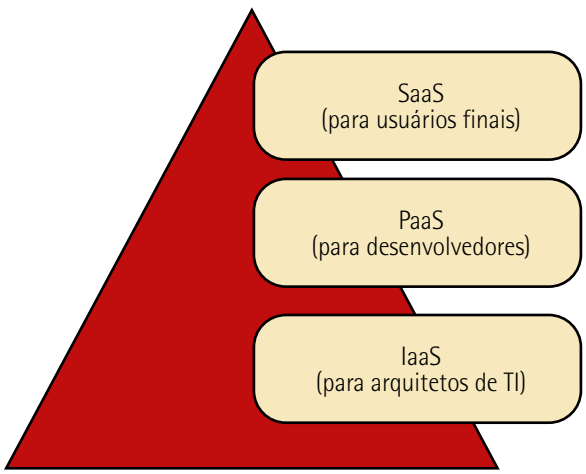

Figura 4 – Arquitetura de modelos de serviços

Fonte:Coulouris, Dollimore e Kindberg (2020, p. 174).

Dependendo do modelo de serviços utilizado, altera-se o responsável pelos componentes da pilha de camadas. A figura a seguir mostra os modelos de serviço em nuvem e a pilha de componentes envolvidos nos serviços IaaS (a), PaaS (b) e SaaS (c):

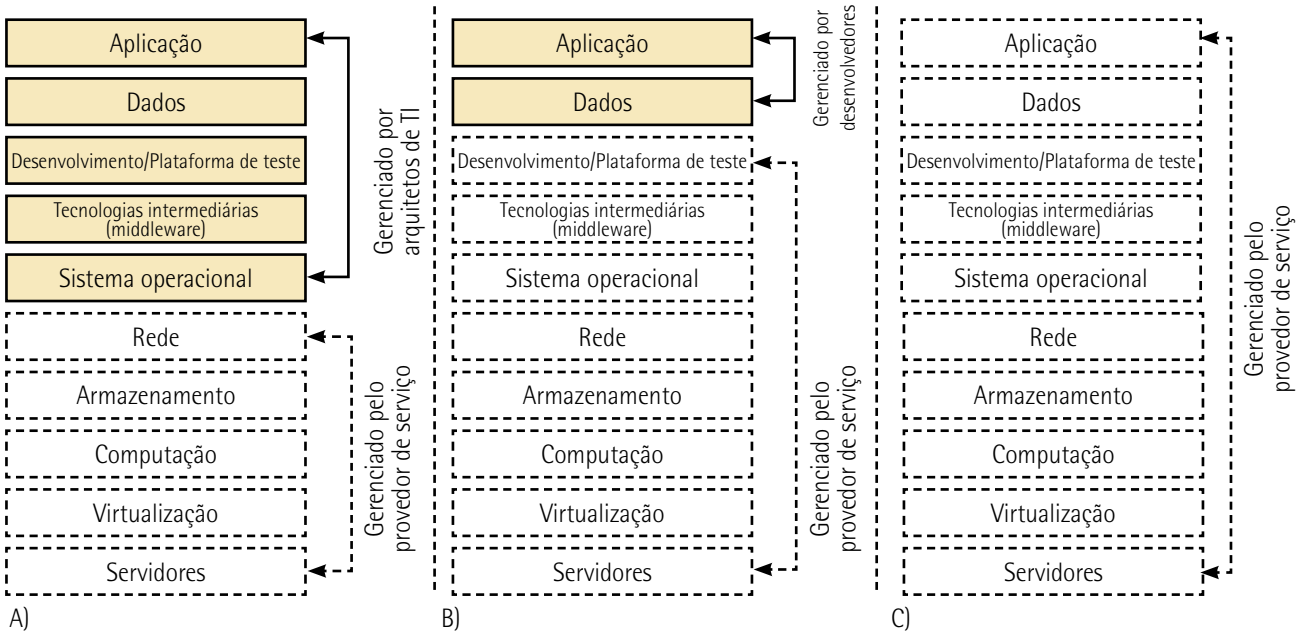

Figura 5 – Modelos de serviços e componentes: A) IaaS, B) PaaS e C) SaaS

Fonte: Silva *et al.* (2020, p. 175).

2 TIPOS E ARQUITETURA DE SISTEMAS DISTRIBUÍDOS

A construção do software de um sistema distribuído normalmente é uma atividade bastante complexa, e os componentes desse sistema estão espalhados por diversos hardwares. Neste tópico será apresentada a classificação de sistemas distribuídos e como eles estão organizados internamente. Eles devem ser confiáveis, gerenciáveis e adaptáveis, e a arquitetura pela qual eles estão estruturados deve contribuir para alcançar essas características, apesar da complexidade inerente aos sistemas distribuídos.

2.1 Tipos de sistemas distribuídos

Dependendo do uso, é possível classificar os sistemas distribuídos em sistemas de computação distribuídos de alto desempenho, computação em nuvem, sistemas de informação distribuídos e sistemas distribuídos pervasivos, que serão detalhados a seguir.

2.1.1 Sistemas distribuídos de alto desempenho

Existem dois subgrupos nessa classificação: computação de cluster e computação em grade ou grid computing. A **computação de cluster** possui uma característica de homogeneidade entre as suas máquinas, e o hardware consiste em um conjunto de estações de trabalho ou computadores semelhantes, conectados por uma rede local de alta velocidade. Todos os computadores estão ligados à mesma rede local e devem utilizar o mesmo sistema operacional.

A popularização dos sistemas de computação de cluster ocorreu quando a relação entre desempenho e preço dos computadores pessoais melhorou. Com isso, ficou conveniente, tanto técnica quanto financeiramente, a construção de um supercomputador que utilizasse tecnologia disponível no mercado por meio da conexão de uma série de computadores a uma rede de alta velocidade.

Normalmente, a computação de cluster é aplicada para programação paralela, em que um mesmo programa, com alta necessidade de processamento computacional, é executado de forma paralela em diversos dispositivos. Como existe muita interação entre as máquinas ou nós, esse sistema é fortemente acoplado.

Um exemplo conhecido na literatura é o cluster Beowulf, que é baseado no sistema operacional Linux. Nesse cluster, o conjunto de nós de computação é controlado e acessado através de um único nó mestre, no qual é realizada a manipulação da alocação dos nós ao programa executado paralelamente, a manutenção da fila de tarefas ou jobs a serem processados e a disponibilização de uma interface para os usuários do sistema. A figura a seguir mostra a configuração do sistema de cluster:

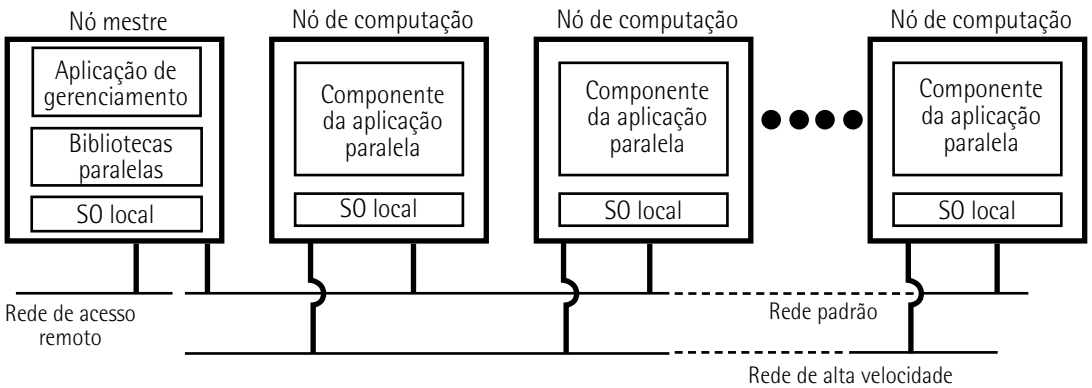

Figura 6 – Exemplo de um sistema de cluster

Fonte: Tanenbaum e Steen (2007, p. 10).

Nesse tipo de sistema, normalmente há o papel de nó mestre, que é responsável por alocar as tarefas aos nós e organizar a fila de tarefas e interface com usuários. Assim, quando ocorre uma sobrecarga em uma máquina do sistema, o nó mestre realiza a distribuição dessas tarefas entre os outros nós, retirando processos dessa máquina que estava sobrecarregada. Assim, o nó mestre provê uma forma de ocultar a migração dos processos, trazendo transparência de localização e migração.

No caso da **computação em grade** ou grid computing, as máquinas são heterogêneas. Cada sistema pode estar em um domínio de rede diferente e utilizar diferentes hardwares, softwares, padrões de rede e até sistemas operacionais. Um conjunto de empresas disponibiliza máquinas que serão conectadas em rede para permitir um processamento utilizando os recursos computacionais dessa rede de computadores.

Saiba mais

Um exemplo de computação em grade existe na PlanetLab, uma rede de pesquisa mundial para criar novos serviços de redes. Ela permite a colaboração de um grupo de pessoas e instituições com compartilhamento de capacidade de processamento e de memória.

Disponível em: http://www.planet-lab.com. Acesso em: 15 abr. 2024.

Com a utilização de recursos heterogêneos, permite-se a colaboração de um grupo de instituições por meio de organização virtual. A figura a seguir apresenta a arquitetura desse tipo de sistema, que é composto de quatro camadas: camada base, de conectividade, de recurso e camada coletiva:

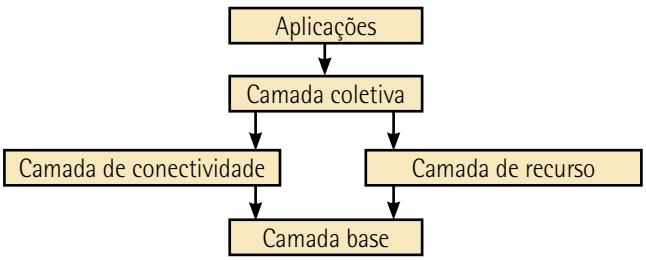

Figura 7 – Exemplo de um sistema de cluster

Fonte: Tanenbaum e Steen (2007, p. 12).

O objetivo da camada base é prover interfaces para recursos locais em um site exclusivo, e as interfaces permitem o compartilhamento de recursos. Na camada de conectividade, são definidos os modos como os componentes estão interligados e são armazenados os protocolos de comunicações utilizados para transações da computação em grade.

O gerenciamento de recursos provê funções de informações de configuração sobre um recurso específico e é realizado pela camada de recurso. A camada coletiva manipula o acesso a múltiplos recursos, aloca e escalona tarefas para múltiplos recursos.

2.1.2 Computação em nuvem ou cloud computing

Desde a década de 2010, a computação em grade está sendo substituída gradativamente pela computação em nuvem ou cloud computing. Esta pode ser entendida como a entrega de recursos de TI sob demanda através da internet e com preço definido pela utilização. Desse modo, as empresas não precisam construir e manter data centers e servidores físicos, pois basta o acesso a serviços conforme a necessidade usando um provedor da nuvem com a Amazon Web Services (AWS), o Microsoft Azure, o Google Cloud, entre outros fornecedores (AWS, 2021).

A cloud computing permite o uso de um recurso de computação como uma máquina virtual (VM), um serviço de armazenamento ou processamento de uma aplicação. É como o uso de um serviço terceirado, no qual as empresas irão pagar pelos serviços computacionais que consumirem. Por exemplo, diversas empresas utilizam soluções em nuvem para serviços de backup, em vez de armazenarem em servidores de arquivos internos.

Um dos benefícios da utilização de computação em nuvem é a flexibilidade, já que os usuários podem dimensionar os serviços para o atendimento de suas necessidades, acessar os serviços em quaisquer locais ou personalizar aplicações. Outra vantagem da nuvem é a eficiência, pois as empresas podem disponibilizar seus aplicativos mais rapidamente no mercado, sem gastos com compra de equipamentos e com manutenção.

2.1.3 Sistemas de informação distribuídos

São sistemas corporativos para integrar diversas aplicações. Depois do surgimento de tablets e smartphones, foi necessário que esses sistemas pudessem ser utilizados em diversas plataformas. Eles podem ser implementados como um sistema de processamento de transações ou uma integração de aplicações empresariais. Um exemplo de sistema de informação distribuído é o sistema bancário, no qual o usuário pode acessar as informações de sua conta através de internet banking, mobile banking, redes de caixas eletrônicos ou ATMs, nas agências bancárias em correspondentes bancários etc.

Um sistema de banco de dados normalmente realiza operações através de transações, e as transações têm quatro propriedades. A primeira é que, para o mundo externo, as transações são indivisíveis ou atômicas. Outra propriedade é que elas não violam invariantes no sistema, isto é, são consistentes. Adicionalmente, as transações são isoladas e não interferem umas nas outras. Por fim, as alterações das transações são permanentes, pois uma vez concluída a transação não é possível desfazer os resultados dessa operação. Esse conjunto de propriedade das transações são conhecidas pela sigla Acid: atômicas, consistentes, isoladas e duráveis.

O quadro a seguir apresenta exemplos de primitivas de transações:

Quadro 2 – Exemplos de primitivas para transações

Primitiva	Descrição
BEGIN_TRANSACTION	Marque o início de uma transação
END_TRANSACTION	Termine a transação e tente comprometê-la
ABORT_TRANSACTION	Elimine a transação e restaure os valores antigos
READ	Leia dados de um arquivo, tabela ou de outra forma
WRITE	Escreva dados para um arquivo, tabela ou de outra forma

Fonte: Tanenbaum e Steen (2007, p. 12).

Através das chamadas de procedimentos remoto, em inglês, remote procedure calls (RPC), é realizada uma comunicação para receber atenção do servidor remoto. Normalmente, as chamadas de procedimento são encapsuladas em uma transação, sendo denominadas RPC transacional.

Existem softwares para monitorar o processamento de transações, que permite que uma aplicação tenha acesso a diversos servidores e bancos de dados de forma a oferecer um modelo transacional.

Em relação aos sistemas de informação com integração de aplicações corporativas, estes utilizam middleware de comunicação por chamada remota de procedimento (RPC), invocações de método remoto (RMI) e middleware orientado a mensagem (MOM).

O RPC é um procedimento escrito que pode ser acessível por outro cliente. No RMI, são usadas linguagens orientadas a objetos.

As aplicações já existentes trocam informações entre elas. O miiddleware facilita a integração de aplicações empresariais, a reutilização de códigos e aplicações.

2.1.4 Sistemas distribuídos pervasivos

Eles têm a instabilidade como comportamento de sistemas. São realizados por dispositivos de computação móvel ou embarcada de pequeno porte e alimentados por bateria; usam conexão sem fio e permitem mobilidade. Uma aplicação desse tipo é denominada eHealth, na qual ocorre o uso de computação móvel, que emprega informações vindas de sensores e, por meio de aplicativos, as processa para tratamento de saúde.

Outro tipo é chamado de redes de sensores sem fio, em que os nós se comunicam entre si. Diversas medições são realizadas através de redes sem fio, como levantamento de informações climáticas e pluviométricas, informações de tráfego e informações de sensores de presença em um sistema de segurança.

Na figura a seguir, é apresentada uma rede de sensores sem fio que enviam os dados diretamente para o operador do sistema que irá armazenar e processar as informações dos sensores (a) ou armazenar e processar os dados em cada sensor (b).

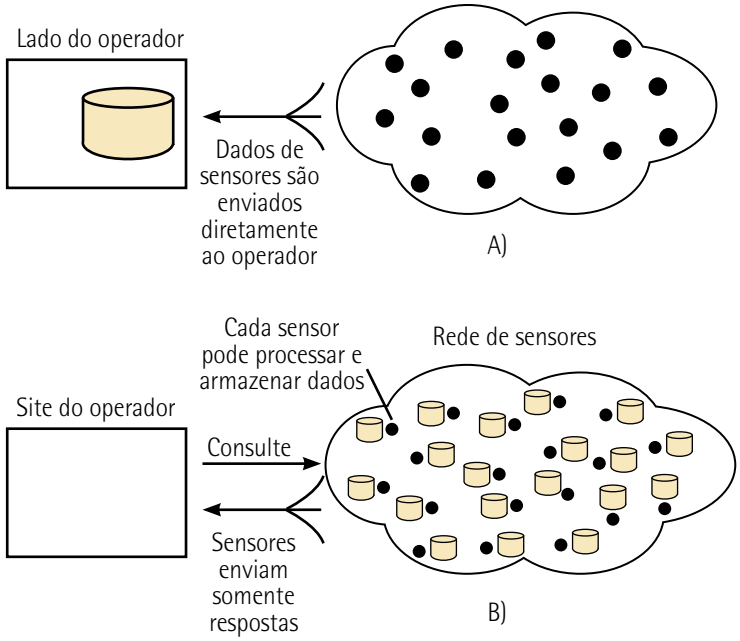

Figura 8 – Organização de um banco de dados de uma rede de sensores sem fio

Fonte: Tanenbaum e Steen (2007, p. 18).

A Internet das Coisas ou Internet of Things (IoT) é uma das categorias de redes de sensores sem fio. Nela, os componentes como sensores, atuadores, lâmpadas ou "coisas" são interligados via internet.

2.2 Arquitetura de sistemas distribuídos

A arquitetura de um sistema representa como é sua estrutura em relação a componentes especificados separadamente e das inter-relações entre essas partes. De forma semelhante ao que é realizado em um projeto arquitetônico para a construção de um edifício residencial em uma cidade, o sistema distribuído também terá uma arquitetura. No projeto do prédio, são definidos aspectos de sua aparência exterior, estilo arquitetônico e sua estrutura geral e interna, como os sistemas de água e elétrico, fornecendo um padrão de referência para o projeto.

Um elemento essencial para os sistemas distribuídos são as entidades que irão se comunicar. Normalmente, essas entidades são processos que possuem linhas de execução denominadas threads. Em sistemas mais primitivos, como redes de sensores e um sistema de monitoramento para segurança, as entidades que se comunicam são chamadas de nós.

As linguagens orientadas a objetos introduziram o conceito de objetos distribuídos. Nessa forma de computação, diversos objetos interagem entre si e são unidades para decompor o domínio do problema. Para acessar esses objetos são aplicadas interfaces com a linguagem de definição de interface, ou interface definition language (IDL), em inglês.

Outra entidade que pode ser usada para comunicação são os componentes, que são semelhantes aos objetos: eles são acessados por interfaces e fornecem abstrações voltadas ao problema para a construção de sistemas distribuídos (Coulouris, Dollimore e Kindberg, 2013). Por outro lado, os componentes detalham as dependências. O middleware baseado em componentes frequentemente provê suporte adicional para áreas importantes, como a implantação e o suporte para programação no lado do servidor.

Podemos entender os componentes como unidades modulares com interfaces requeridas e fornecidas bem definidas que são substituíveis dentro de seu ambiente. Assim, um componente do sistema pode ser substituído, desde que sejam mantidas suas interfaces.

Os serviços web também são um padrão importante para o desenvolvimento de sistemas distribuídos. Além de adotarem interfaces como objetos e componentes, aplicam serviços web, integrados à world wide web, e aplicam padrões da web para representar e descobrir serviços.

Um serviço web é um aplicativo de software identificado por um URI, cujas interfaces e vínculos podem ser definidos, descritos e descobertos como artefatos da XML. Ele suporta interações diretas com outros agentes de software, usando trocas de mensagens baseadas em XML por meio de protocolos de internet.

Outra questão relevante para um sistema distribuído é qual o padrão de comunicação utilizado. Existem três tipos de paradigmas de comunicação: comunicação entre processos, comunicação indireta e invocação remota.

A comunicação entre processos representa o nível mais básico de comunicação e inclui primitivas de passagem de mensagens, suporte para multicast – isto é, envio para múltiplos destinatários – e acesso direto à API.

O paradigma mais comum em sistemas distribuídos é a invocação remota, que contempla diversas técnicas de comunicação bilateral entre os componentes e resulta na chamada de um procedimento em dispositivo remoto.

Outro padrão relevante são os protocolos de requisição-respostas utilizados para suportar operações cliente-servidor. Tais protocolos envolvem uma troca de pares de mensagens entre cliente e servidor. Inicialmente, a primeira mensagem é formada por uma codificação da operação que o servidor irá executar. A segunda mensagem será composta pela resposta à requisição inicial, codificada como um vetor de bytes. Parte dos sistemas distribuídos usa chamadas de procedimento remoto ou invocação de método remoto como estratégias para troca do tipo requisição-resposta.

Uma chamada de procedimento remoto ou remote procedure call (RPC) trata o procedimento remoto com chamadas como se estivesse em um espaço de armazenamento local.

Na invocação de método remoto ou remote method invocation (RMI), um objeto chamador pode invocar um método em um objeto remoto e está mais voltada para objetos distribuídos.

Outro ponto de atenção é o modo como as entidades são mapeadas na infraestrutura física, que pode ocorrer por meio de muitos dispositivos conectados por uma rede arbitrária. A posição das entidades é essencial para a determinação de propriedades do sistema distribuído, como desempenho, confiabilidade, segurança e transparência.

É necessário considerar os padrões de comunicação entre as entidades, a carga de trabalho atual, a qualidade da comunicação entre as diferentes máquinas, o nível de serviço exigido (QoS), a confiabilidade das máquinas, a segurança do sistema e outras características para determinar o posicionamento das entidades no sistema distribuído.

2.3 Estilos arquitetônicos

Para entender a organização lógica dos sistemas distribuídos, estes devem ser divididos em componentes de software. O estilo arquitetônico é estabelecido com base em seus componentes, na forma como eles estão interligados entre si, nos dados intercambiados entre eles e também no modo como são configurados para formação de um sistema.

De acordo com Coulouris, Dollimore e Kindberg (2013), para a compreensão dos elementos de um sistema distribuído, deve-se ter as respostas para as quatro perguntas a seguir:

- Quais são as entidades que irão se comunicar no sistema distribuído?
- Qual é o paradigma de comunicação aplicado ou como essas entidades se comunicam?
- Quais funções e responsabilidades estão relacionadas a eles na arquitetura global?
- Qual é a localização de cada componente do sistema?

Os componentes de um sistema são unidades modulares que têm interfaces bem definidas e que podem ser substituídas dentro de seu ambiente por um módulo semelhante. Existem interfaces requeridas, ou dos serviços que são necessários para esse componente, e interfaces fornecidas pelos componentes, isto é, os serviços que esse componente provê para outros.

Os componentes podem ser interligados entre si através de conexões, de forma a ter uma aplicação que contemple esses componentes. Um conector é entendido como um mecanismo que funciona para mediar a comunicação ou cooperação entre componentes.

Adicionalmente, é necessário definir como os dados são trocados entre os componentes. Essa troca pode ocorrer por uma memória ou repositório compartilhado ou por troca de mensagens entre os componentes como se fosse uma caixa postal.

Um último aspecto está relacionado à forma de configuração dos componentes. Os parâmetros de configuração podem ser definidos quando os componentes são instanciados/inicializados ou são definidos durante a execução da aplicação.

Alterando-se componentes e conectores, é possível criar diferentes configurações, que foram denominadas estilos arquitetônicos. Para sistemas distribuídos, os estilos arquitetônicos mais relevantes são:

- arquitetura em camadas;
- arquitetura baseada em objetos;
- arquitetura baseada em dados;
- arquitetura baseada em eventos.

2.3.1 Arquitetura baseada em camadas

Vejamos uma arquitetura de comunicação em camadas do protocolo TCP/IP. Nesse modelo, cada camada tem uma função específica e é possível alterar uma sem modificar o conteúdo das outras. Na figura a seguir é ilustrada a organização conceitual de protocolos em camadas:

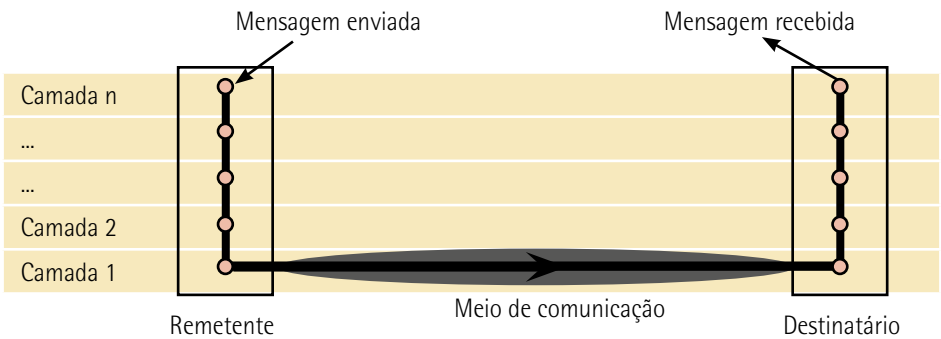

Figura 9 – Organização de protocolo em camadas

Fonte: Coulouris, Dollimore e Kindberg (2013, p. 92).

A arquitetura em camadas também é aplicável a sistemas distribuídos, nos quais seus componentes são organizados em camadas. Um componente da camada N tem permissão para chamar componentes da camada imediatamente abaixo (N-1). Entretanto, a camada inferior não tem permissão para acessar a camada acima.

A figura a seguir apresenta o estilo arquitetônico em camadas, os fluxos de requisição, que descem a hierarquia de camadas, e o fluxo de resposta, que vai da camada inferior para a superior.

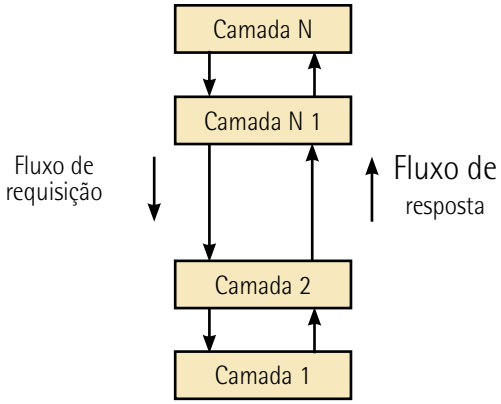

Figura 10 – Arquitetura em camadas

Fonte: Tanenbaum e Steen (2007, p. 21).

2.3.2 Arquitetura baseada em objetos

Na arquitetura baseada em objetos, cada componente será representado por um objeto. Esses objetos são conectados entre si por uma chamada de procedimento remoto, ou remote procedure call (RPC), em inglês. É bastante explorada na arquitetura cliente-servidor, sendo muito aplicada em sistemas de software de grande porte.

A figura a seguir acentua os elementos da arquitetura baseada em objetos:

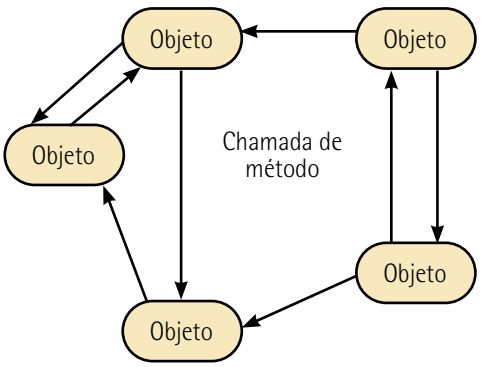

Figura 11 – Estilo arquitetônico baseado em objetos

Fonte: Tanenbaum e Steen (2007, p. 21).

2.3.3 Arquitetura baseada em dados

Na arquitetura baseada em dados, os componentes se comunicam por meio de um repositório de dados compartilhado, semelhante a uma caixa postal. Trata-se de um sistema fracamente acoplado, já que não há interação direta entre os componentes. Envolve sistemas pervasivos, como redes de sensores sem fio e Internet das Coisas (IoT).

2.3.4 Arquitetura baseada em eventos

Na arquitetura baseada em eventos, a comunicação entre processos ocorre com a propagação de eventos, os quais podem transportar dados. Esses sistemas são conhecidos como publicar/subescrever (publish/subscribe), nos quais os processos publicam eventos e o middleware garante que apenas os processos que se subscrevem a esses eventos são os que irão receber.

É possível fazer uma analogia com a publicação de eventos do Facebook. Quando um usuário se inscreve para acessar um conteúdo de uma conexão na rede, ele irá receber todos os eventos referentes àquela conexão.

Pode-se combinar as arquiteturas baseadas em eventos e as arquiteturas baseadas em dados de forma a resultar em espaços compartilhados de dados.

2.4 Modelo de interação

Os sistemas distribuídos são formados por uma coleção de processos que interagem entre si por troca de mensagens. O comportamento e o estado são descritos através de um algoritmo distribuído para cada processo do sistema distribuído, inclusive a transmissão de mensagens entre os processos.

Em situações normais, não é possível prever o tempo de execução de cada processo e a sincronização das mensagens trocadas entre eles. Em sistemas distribuídos, existem dois fatores que alteram de forma relevante a interação de processos: o desempenho da comunicação e a impossibilidade de obter uma noção global de tempo unificada no sistema.

Um dos fatores que limitam a interação em um sistema é o desempenho da comunicação, que é afetada em algumas características, como latência, largura de banda e jitter.

O atraso que ocorre entre o começo do envio de uma mensagem em um processo remetente e o início do recebimento pelo destinatário é denominado **latência**. A latência total compreende o atraso de transmissão, o atraso no acesso à rede e o tempo de processo dos serviços de comunicação do sistema operacional nos processos de transmissão e recepção de mensagens.

A **largura de banda** ou bandwidth de uma rede de comunicações representa a quantidade de informação que pode ser transmitida pela rede em determinado instante de tempo. Essa largura de banda disponível é compartilhada por todos os elementos que realizam comunicações naquela rede.

Segundo Coulouris, Dollimore e Kindberg (2013, p. 63), "**jitter** é a variação estatística do atraso na entrega de dados em uma rede, que produz uma recepção não regular dos pacotes". A variação do atraso é essencial para elementos multimídia, pois caso amostras consecutivas de um arquivo de áudio forem reproduzidas em intervalos de tempo irregulares, o som produzido sairá com distorções.

A temporização é uma questão relevante para sistemas distribuídos, já que cada computador tem um clock ou relógio interno próprio. Desse modo, dois processos executados em computadores

distintos vão associar carimbos de tempo ou time stamps aos seus eventos. Existe uma taxa de desvio do relógio ou drift que mede a variação de tempo entre um relógio qualquer e o relógio de referência ideal. Assim, mesmo que fosse feito um ajuste inicial do tempo, posteriormente os relógios não estariam sincronizados.

Em sistemas distribuídos síncronos, é definido um limite de tempo inferior e superior para que cada mensagem enviada em um canal seja recebida e cada processo possua um relógio local com taxa de desvio com uma limitação de valor máximo.

Por outro lado, os sistemas distribuídos assíncronos, como a internet, não fazem nenhuma consideração em relação aos intervalos de tempo envolvidos em qualquer tipo de execução, como a velocidade de execução do processo, os atrasos na transmissão da mensagem e as taxas de variação do relógio. Nesse tipo de sistema, não é possível sincronizar os relógios dos diferentes processos existentes.

2.5 Arquitetura de sistemas

Analisando o papel de cada componente, as arquiteturas do sistema podem ser divididas nos seguintes tipos:

- Centralizada:
  - cliente-servidor;
  - multidivididas.
- Descentralizada:
  - par-a-par (peer-to-peer) estruturada;
  - par-a-par não estruturada;
  - gerenciamento de topologia de redes de sobreposição;
  - superpares.
- Híbridas:
  - sistemas de servidor de borda;
  - sistemas distribuídos colaborativos.

Como exemplos de arquiteturas centralizadas de modelo cliente-servidor, podemos apontar sites de notícias, o sistema bancário com mobile banking ou internet banking e serviços de streaming.

As arquiteturas descentralizadas utilizam sistemas par-a-par ou peer-to-peer. O algoritmo e protocolo Chord é um exemplo de arquitetura descentralizada.

As arquiteturas híbridas combinam características das arquiteturas centralizadas e descentralizadas. Aplicações como BitTorrent, Skype e WhatsApp são exemplos.

2.6 Arquitetura centralizada

É a mais usada em sistemas distribuídos. Está baseada no modelo cliente-servidor, no qual os processos estão divididos em dois grupos. O processo **servidor** é responsável pela implementação de um serviço específico, como um serviço de banco de dados, de e-mail ou de streaming. Já o processo **cliente** realiza uma requisição de serviço ao servidor e, na sequência, aguarda uma resposta do servidor. Essa forma de interação é conhecida como requisição-resposta.

A figura a seguir apresenta a interação entre cliente e servidor em função do tempo. Inicialmente, o cliente que requisita um serviço empacota uma mensagem para o servidor identificando o serviço que deseja e, também, envia os dados necessários. Posteriormente, o servidor, que sempre espera a chegada de uma requisição, a receberá, realizará o seu processamento e empacotará os resultados em uma mensagem de resposta enviada para o cliente (Monteiro *et al.*, 2020).

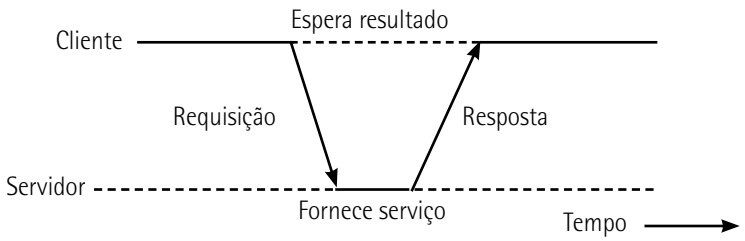

Figura 12 – Interação entre cliente e servidor

Fonte: Tanenbaum e Steen (2007, p. 22).

Um modelo cliente-servidor é semelhante a serviços fora do domínio da web. No sistema de telefonia, uma central telefônica em qualquer área cumpre o papel de um servidor. Por sua vez, um assinante da empresa pode ser visto como cliente e, ao telefonar, pede um número de telefone específico para a central. A central deve estar preparada e disponível o maior tempo possível. O assinante pode realizar chamadas por um curto intervalo de tempo, dependendo de sua necessidade do serviço.

Um ponto crítico para essa arquitetura se refere à distinção entre servidor e clientes. Considere um banco de dados distribuído: um servidor pode agir constantemente como cliente quando repassa requisições para um servidor de sistema de arquivos, que implementa as tabelas do banco de dados.

Para resolver esses dilemas na definição de clientes e servidores, considerando que muitas aplicações cliente-servidor têm como objetivo suportar o acesso de usuários ao banco de dados, utiliza-se um sistema em camadas com os seguintes níveis:

- nível de interface de usuário;
- nível de processamento;
- nível de dados.

A figura a seguir apresenta um organograma simplificado de um mecanismo de busca com três camadas distintas.

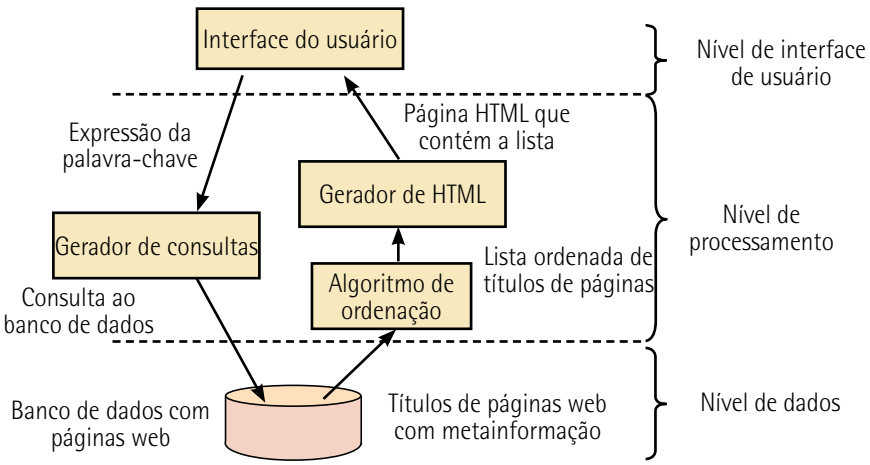

Figura 13 – Organização em três camadas de mecanismo de busca

Adaptada de: Tanenbaum e Steen (2007, p. 24).

Para distinguir entre os níveis lógicos, surgem algumas possibilidades para distribuição física de uma aplicação cliente-servidor por diferentes máquinas. Em uma organização com duas máquinas, há dois tipos de máquinas:

- uma máquina cliente, com apenas a interface de usuário;
- uma máquina servidor, com as camadas de processamento e de dados.

A figura a seguir acentua possíveis configurações do modelo cliente-servidor, alterando os níveis que são implementados no servidor ou no cliente. Em (a), todas as tarefas são manipuladas pelo servidor e o cliente terá apenas uma parte do software da interface com o usuário. Em (b), todo o software da interface de usuário é colocado no cliente, mas ainda não há nenhum processamento no cliente. Em (c), uma parte da aplicação é colocada no cliente, como um formulário de uma página que faz uma checagem de consistência nos dados. Em (d), toda a aplicação será processada pelo cliente. Em (e), apenas a parte de dados será disponibilizada no servidor, e o restante será executado no cliente.

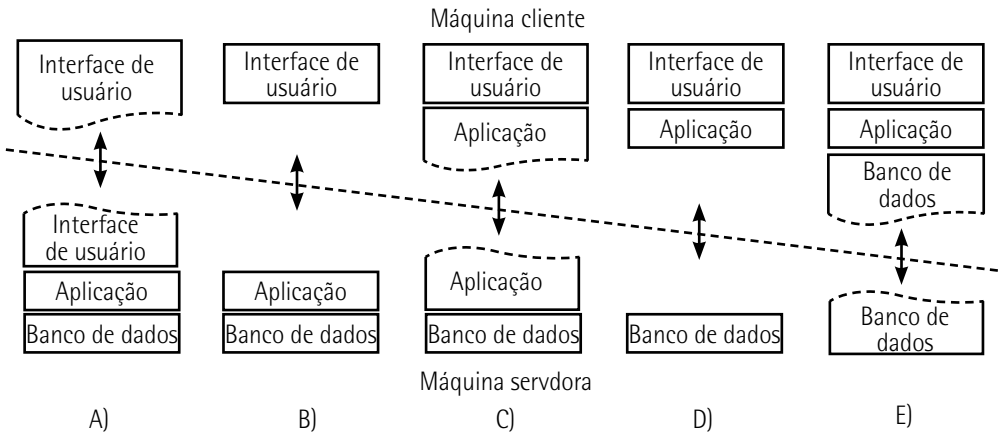

Figura 14 – Alternativas no modelo cliente-servidor

Fonte: Tanenbaum e Steen (2007, p. 25).

Quando o cliente implementa uma quantidade menor de níveis, como as letras de (a) a (c) da figura, esse processo é um cliente magro ou thin client. Quando o cliente usa mais níveis, de (d) a (e), ele é um cliente gordo ou fat client.

Existem situações em que um servidor precisa agir como cliente, o que irá resultar em uma arquitetura de três divisões em termos físicos e pode ser entendido como um modelo arquitetural em camadas. Nessa arquitetura, os programas do nível de processamento estarão armazenados em um servidor apartado, e ela é conhecida como arquitetura de três divisões. A figura a seguir demonstra essa variação do modelo cliente-servidor:

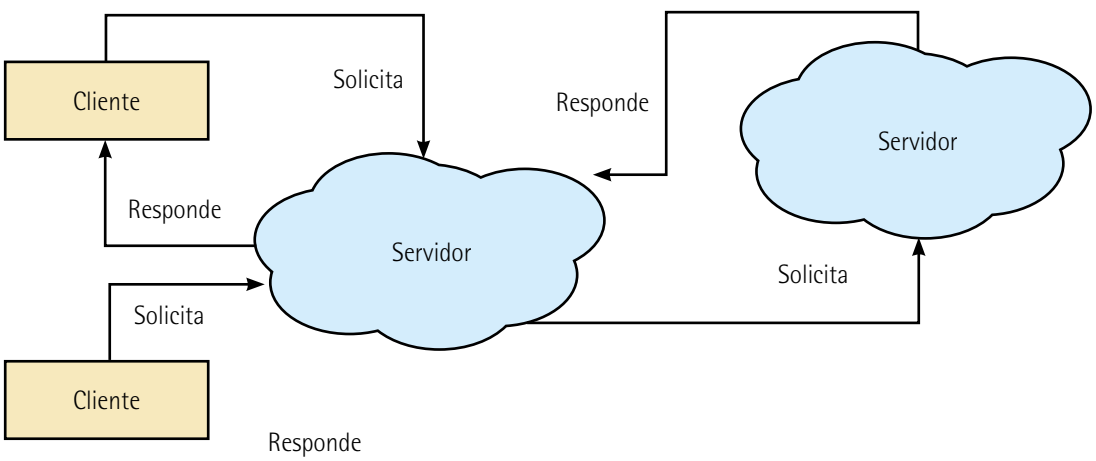

Figura 15 – Variação do modelo cliente-servidor: servidores atuando como clientes

Fonte: Monteiro *et al.* (2020, p. 65).

O processamento de transações é realizado através da arquitetura de três divisões. Outro exemplo de aplicação desse modelo consiste em um servidor web, o qual normalmente é um cliente de um servidor de arquivos local, responsável pelo gerenciamento de arquivos que contêm páginas web.

Os mecanismos de busca também usam essa variação de arquitetura e funcionam considerando múltiplas threads concorrentes, algumas atendendo aos clientes e outras executando os rastreadores de rede ou web crawlers. Estes operam como clientes de servidores web e coletam informação do conteúdo de sites publicados e desempenham papéis de indexadores e robôs ou bots. Essas funções são executadas para manter a base de dados dos motores de busca atualizados.

A figura a seguir mostra o comportamento do servidor que também atua como cliente em outro instante de tempo:

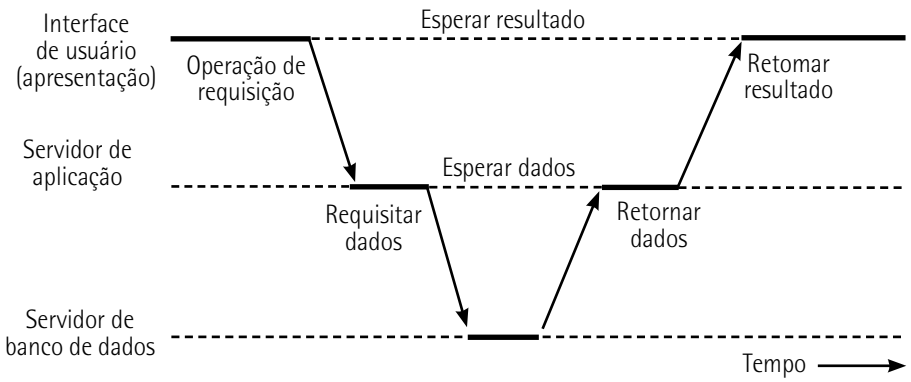

Figura 16 – Exemplo de servidor que age como cliente

Fonte: Tanenbaum e Steen (2007, p. 26).

Uma limitação do modelo cliente-servidor, que opera em uma estrutura centralizada, ocorre quando o número de solicitações começa a comprometer toda a capacidade dos servidores. A centralização de recursos impõe uma limitação de crescimento ao sistema, sendo o gargalo do modelo cliente-servidor. Assim, foram desenvolvidas arquiteturas que realizam a descentralização de recursos da rede, as quais serão abordadas no próximo tópico.

3 ARQUITETURA DE SISTEMAS: DESCENTRALIZADA E HÍBRIDA

Arquiteturas centralizadas usam o modelo cliente-servidor e realizam uma distribuição vertical, isto é, seus componentes logicamente diferentes estão em máquinas distintas. Para atender às necessidades de novas aplicações, em arquiteturas modernas, é usada também a distribuição horizontal, na qual é considerada a distribuição dos clientes e dos servidores. Os sistemas reais podem combinar as arquiteturas.

3.1 Arquiteturas descentralizadas

O cliente-servidor possui duas distribuições: vertical e horizontal. Na vertical, os componentes logicamente distintos são implementados por máquinas diferentes. Cada máquina irá executar um conjunto específico e pré-definido de funções; dessa forma, as funções de cada uma das máquinas estão bem delimitadas.

Na distribuição horizontal, o cliente ou servidor pode ser fisicamente subdividido em partes lógicas equivalentes, que podem atuar de forma colaborativa. Nessa estrutura, é possível realizar um balanceamento da carga: caso o servidor esteja sobrecarregado, outros servidores serão acionados para a distribuição da carga.

Os papéis entre os nós não ficam muito bem definidos e não há uma hierarquia entre os nós da rede, por isso eles são chamados de pares ou peers. Assim, essa arquitetura descentralizada também é denominada par-a-par ou peer-to-peer, representada pela sigla (P2P).

Nessa arquitetura, não é necessário que um processo-servidor seja executado ininterruptamente e aguarde pela conexão dos processos clientes. Todos os nós das redes ou peers terão a responsabilidade compartilhada, de modo que um par pode prover serviços em dado momento e receber serviços em outro (Forouzan; Mosharraf, 2013).

Os processos entre clientes e servidores são todos iguais e flat. Cada processo se comporta simultaneamente como servidor e como cliente, atuando como **servente**. O cliente é quem inicia a requisição, e as demandas geradas por clientes podem ser atendidas por outros clientes, sem ficar dependendo da capacidade de um único servidor.

Uma vantagem dessa arquitetura é sua escalabilidade, pois o sistema pode crescer em quantidade de nós e se expandir sem perda de desempenho. Contudo, é necessário lidar com a entrada e a saída de pares ao longo do tempo, trazendo uma complexidade maior para o projeto do sistema distribuído.

A figura a seguir apresenta a arquitetura peer-to-peer:

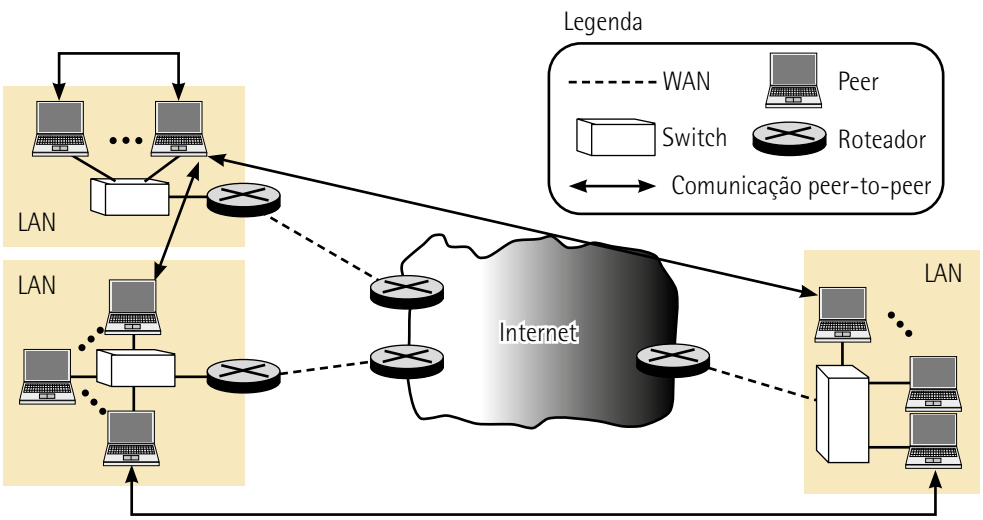

Figura 17 – Arquitetura par-a-par

Adaptada de: Forouzan e Mosharraf (2013, p. 37).

Os nós são formados pelos processos e os enlaces que representam os canais de comunicação formam uma rede de sobreposição. Existe uma topologia que representa essa rede; dependendo da rede de sobreposição do sistema, a arquitetura descentralizada pode ser classificada em estruturada ou não estruturada.

3.1.1 Arquiteturas descentralizadas estruturadas

A rede de sobreposição é construída usando um mecanismo determinístico e se comporta como se cada nó da rede tivesse um identificador único ou uma chave.

Observação

Em um algoritmo determinístico, dada uma certa entrada, o algoritmo sempre resultará na mesma saída e a sequência de estados da máquina será a mesma.

Um mecanismo utilizado frequentemente é chamado de tabela de hash distribuído – em inglês, distributed hash table (DHT). Os nós e dados dessa rede recebem uma chave aleatória, que possui entre 128 e 160 bits.

O desafio para essa estrutura é o mapeamento do identificador de um nó conhecendo a chave de um dado. De forma semelhante, os nós do sistema recebem uma chave aleatória. Nesse caso, a requisição é roteada entre os nós até alcançar o nó com o dado especificado.

Esse tipo de rede é amplamente usado para distribuição de conteúdo, como filmes, arquivos e músicas. O ponto principal dessa arquitetura é que quando consulta um item de dado, o endereço de rede do nó responsável por aquele item de dado é encontrado de forma única e eficiente.

Uma implementação dessa arquitetura é o sistema Chord, no qual os nós estão organizados logicamente em um anel. Caso um item de dado tenha uma chave qualquer (n), é mapeado para um nó que tem o menor identificador (id) superior a n. Tal nó é conhecido como o sucessor da chave n, como representado na figura a seguir:

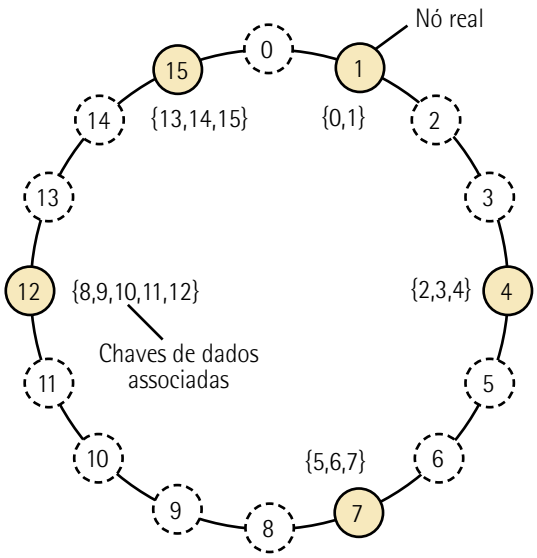

Figura 18 – Mapeamento dos itens de dados para pares no Chord

Fonte: Tanenbaum e Steen (2007, p. 27).

3.1.2 Arquiteturas descentralizadas não estruturadas

Nesse tipo de arquitetura, são usados algoritmos aleatórios para a construção da rede de sobreposição. Com isso, os nós não têm uma visão geral da rede, mas somente mantêm uma lista de nós vizinhos ou adjacentes, com uma visão parcial da rede.

Os dados são armazenados de forma aleatória, fato que dificulta uma busca na rede. Para pesquisa na rede, é realizada uma tempestade de broadcast, de modo que todos os nós da rede recebem a instrução da busca e indicam se possuem o conteúdo pesquisado. Essa forma de pesquisa é bastante custosa em termos de desempenho.

3.1.3 Arquiteturas descentralizadas: superpares

Com o aumento do tamanho da rede e da quantidade de dados existentes, a busca por itens de dados em sistemas distribuídos não estruturada torna-se problemática.

Para facilitar essa busca, alguns nós são escolhidos como superpares ou superpeers e irão atuar como intermediários. Esses nós manterão um índice do conteúdo da rede e a pesquisa será realizada nos superpares. Somente onde a pesquisa do superpar tiver sucesso os pares comuns desse superpar irão realizar a busca pelo dado. Desse modo, há uma organização hierárquica da rede. A figura a seguir ilustra essa estrutura.

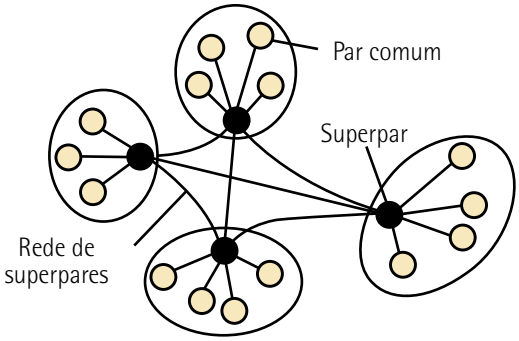

Figura 19 – Organização hierárquica em uma rede não estruturada superpares

Fonte: Tanenbaum e Steen (2007, p. 27).

Sempre que um nó comum for adicionado à rede, ele será conectado a um dos superpares. Um problema fundamental dessa arquitetura é a escolha dos superpares quando um superpar não está mais presente na rede. Um dos requisitos para ser escolhido como superpar é que o nó deve ter longo tempo na rede e possuir alta disponibilidade.

Uma solução para esse comportamento é não utilizar a associação fixa entre cliente-superpar. Por exemplo, pense em uma rede de compartilhamento de arquivos: é recomendado que um cliente se conecte a um superpar que guarde um índice de arquivos.

3.2 Arquiteturas híbridas

Nesse tipo de arquitetura, soluções de cliente-servidor são combinadas como forma de funcionamento da arquitetura descentralizada. A arquitetura centralizada é utilizada para servir como diretório da informação e a arquitetura descentralizada tem a função de distribuição de conteúdo.

Diversas aplicações usadas no nosso dia a dia são formadas por arquiteturas híbridas, tais como os aplicativos WhatsApp, Skype e BitTorrent.

Os sistemas distribuídos colaborativos utilizam inicialmente um esquema de busca pelo cliente-servidor tradicional. Posteriormente, o nó se junta para dar um mecanismo descentralizado de colaboração.

3.2.1 Sistemas de servidor de borda

Em arquiteturas híbridas, existe uma classe de sistemas distribuídos denominada sistemas de servidores de borda. Na internet, existem servidores que são postos na "borda" da rede, que é composta pelas fronteiras entre redes corporativas e a própria internet, disponibilizada por um **provedor de serviço de internet** ou internet service provider (ISP). A figura a seguir acentua a visão da internet como rede formada por servidores de borda:

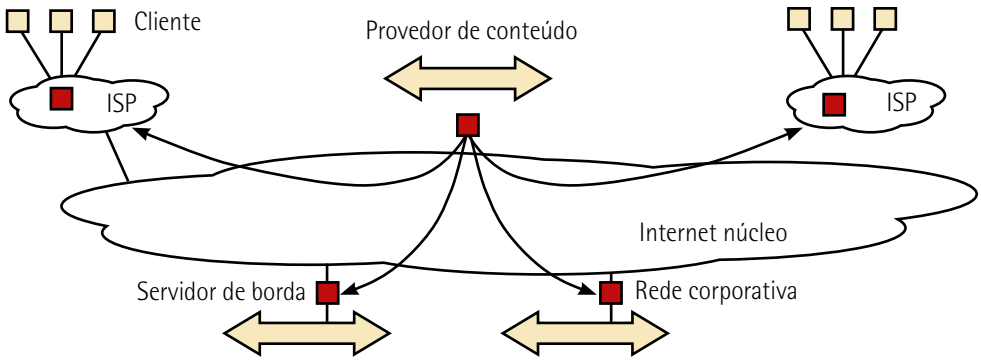

Figura 20 – Servidores de borda e a internet

Fonte: Tanenbaum e Steen (2007, p. 31).

Os usuários finais estão conectados à internet através de um servidor de borda cujo objetivo é servir conteúdo, provavelmente após etapas de filtragem e codificação.

3.2.2 Sistemas distribuídos colaborativos

Em sistemas distribuídos colaborativos, quando um novo nó é inserido no sistema, ele poderá usar uma estrutura descentralizada para colaboração. Um exemplo é o sistema de compartilhamento de arquivos BitTorrent. Quando um usuário final procurar um arquivo, ele transfere partes do arquivo de outros usuários até que essas partes possam ser montadas em conjunto, formando um arquivo completo.

Visando ter um arquivo, o usuário precisa acessar um diretório global daquela rede. Esse diretório contém as referências para os arquivos.torrent, os quais têm as informações específicas para a transferência daquele arquivo. O rastreador é um servidor que mantém a contabilização dos nós ativos que possuem o arquivo desejado.

A figura a seguir ilustra a operação do BitTorrent, que tem a colaboração entre os usuários como característica marcante:

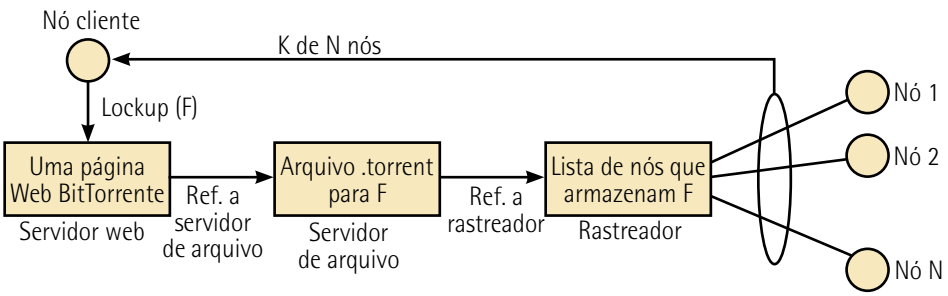

Figura 21 – Funcionamento do BitTorrent

Fonte: Tanenbaum e Steen (2007, p. 32).

3.3 Arquiteturas *versus* middleware

O middleware pode ser configurado de forma estática, quando são instanciados, ou dinâmica, isto é, em tempo de execução. Os middleware como CORBA são baseados em objetos.

Uma solução para reconfigurar a necessidade da aplicação são os interceptadores. Esses componentes interrompem o fluxo de controle usual da aplicação.

Um interceptador é um software que fica entre a aplicação, e o middleware é uma solução para reconfigurar a necessidade de aplicação. Esse software interrompe o fluxo de controle normal e permite a execução de outro código, específico para aplicação.

Utilizando interceptadores, um objeto A pode chamar um método que pertence a um objeto B enquanto este estiver em uma máquina diferente de A. A invocação remota a objeto ocorre em três etapas. Inicialmente, é oferecida a mesma interface de B para o objeto A. Depois, a chamada por A é transformada em uma invocação a objeto genérico oferecido pela máquina em que A é executado. Na última etapa, a invocação a objeto genérico é transformada em uma mensagem a ser transmitida através da rede na camada de transporte disponibilizada pelo sistema operacional local de A.

Observe a seguir o uso de interceptadores que manipulam invocações de um objeto remoto:

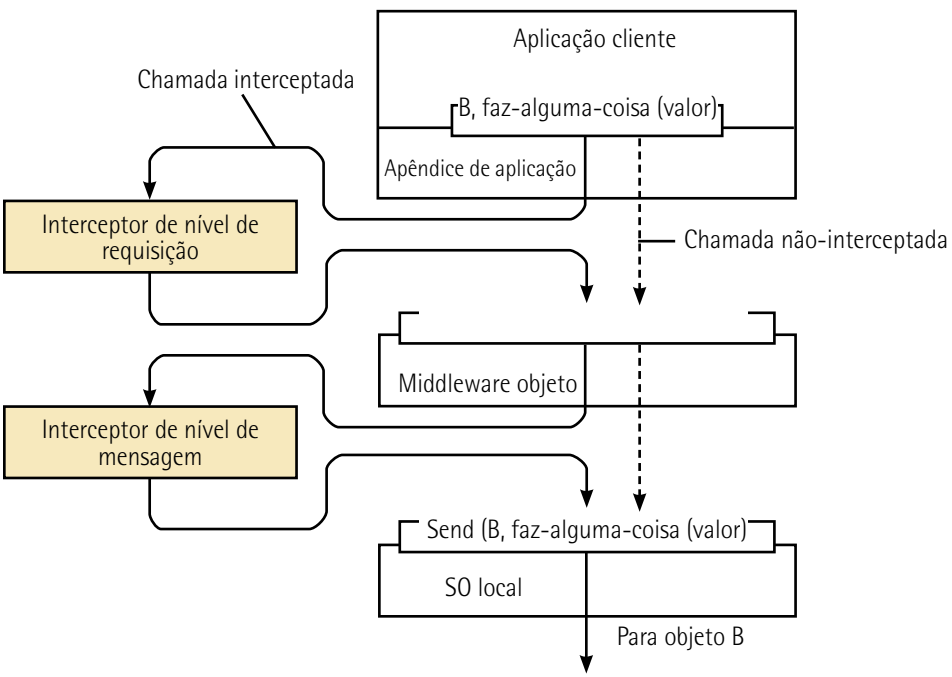

Figura 22 – Utilização de interceptadores

Fonte: Tanenbaum e Steen (2007, p. 34).

Adicionalmente, é possível inserir mecanismos de segurança em cada chamada para verificar se a operação solicitada pode ser executada.

Existem middlewares adaptativos, nos quais as instâncias dos componentes são inseridas e excluídas conforme são solicitadas. Desse modo, o sistema manterá na memória somente os módulos necessários para aquela aplicação, economizando memória do dispositivos, que é uma restrição importante para sistemas embarcados e ambientes IoT.

A aplicação será customizada de acordo com o contexto. Por exemplo, se está sendo utilizado o padrão Bluetooth para comunicação, somente os módulos de Bluetooth serão carregados.

3.3.1 Projeto de sistemas distribuídos

É formado por três modelos vitais: os físicos, que em geral descrevem o sistema visando aos tipos de equipamentos e interconexões existentes; os arquiteturais, que abrangem o projeto do sistema considerando tarefas computacionais e a respectiva comunicação inerente aos componentes; e, por fim, os fundamentais, que descrevem o sistema com foco nas propriedades presentes em todas as arquiteturas distribuídas.

Um desafio para o projeto de sistemas distribuídos é que há uma variação dos modos de uso ou de manipulação do sistema, que podem ter variação de componente ou de carga de trabalho. Por exemplo, dispositivos móveis podem entrar e sair do sistema com frequência.

Outra dificuldade é a variação de ambientes, já que sistemas distribuídos tendem a ser heterogêneos por natureza. Além disso, podem existir ameaças externas, que podem executar um ataque de negação de serviço ou denial of service (DoS), através da realização de muitas solicitações ao servidor, que não terá capacidade para responder a todas elas e, com isso, sofrerá uma indisponibilidade de serviço.

4 PROCESSOS DISTRIBUÍDOS

4.1 Processos e threads

Segundo Machado e Maia (2013), pode-se definir um processo como o conjunto necessário de informações para que o sistema operacional implemente a concorrência de programa, isto é, um programa em execução, abrangendo as informações que devem ser armazenadas para continuar o processamento em período posterior.

O processo mantém todas essas informações para execução de um programa, como o conteúdo de registradores e o espaço de memória. Processo também é o ambiente onde se executa um programa.

Nenhum programa é executado diretamente na memória principal, e sim dentro de um processo; nesse caso, o programa faria uso indiscriminado de qualquer área da memória principal, efetuando operações de E/S indevidas e comprometendo a integridade e a consistência dos dados.

Observação

Um programa por si só não é sinônimo de um processo. Ele é uma entidade passiva, assim como um arquivo contendo uma lista de instruções armazenadas no disco e normalmente é chamado de executável. Por sua vez, um processo é uma entidade ativa, com um contador de programa especificando a próxima instrução a ser executada e um conjunto de recursos associados. Um programa se torna um processo no instante em que seu arquivo executável é carregado na memória principal (Córdova Junior; Ledur; Moraes, 2019).

Uma analogia feita por Deitel, Deitel e Chofines (2005) é que o programa é para o processo o mesmo que a partitura é para uma orquestra sinfônica.

De acordo com Maziero (2019), um processo pode ser entendido como uma unidade de contexto, isto é, um contêiner de recursos a serem usados por uma ou mais tarefas para sua execução. Esses recursos são as áreas de memória, como dados, código e pilha, informações de contextos e descritores de recursos do núcleo como conexões de rede, arquivos abertos, entre outros.

É frequente a necessidade de comunicação entre os processos que ocorre por meio da comunicação interprocessos ou IPCs.

4.1.1 Troca de contexto

Quando uma interrupção ocorre, é necessário que o sistema operacional salve o contexto corrente do processo em execução na CPU para retomar esse contexto de processo futuramente. De forma geral, armazena o estado corrente da unidade central de processamento (UCP), e então uma restauração do estado poderá retomar as operações.

A tarefa conhecida como **mudança de contexto** realiza a alocação da UCP a outro processo e requer a execução do armazenamento do estado do processo atual e a restauração do estado de um processo diferente. Durante o tempo gasto na mudança de contexto, o sistema não executa trabalho útil, o que representa uma perda no processamento. Uma velocidade típica é para a mudança de contexto, alguns milissegundos, ação que depende do suporte de hardware. A figura a seguir mostra as operações necessárias para a realização de uma troca de processos:

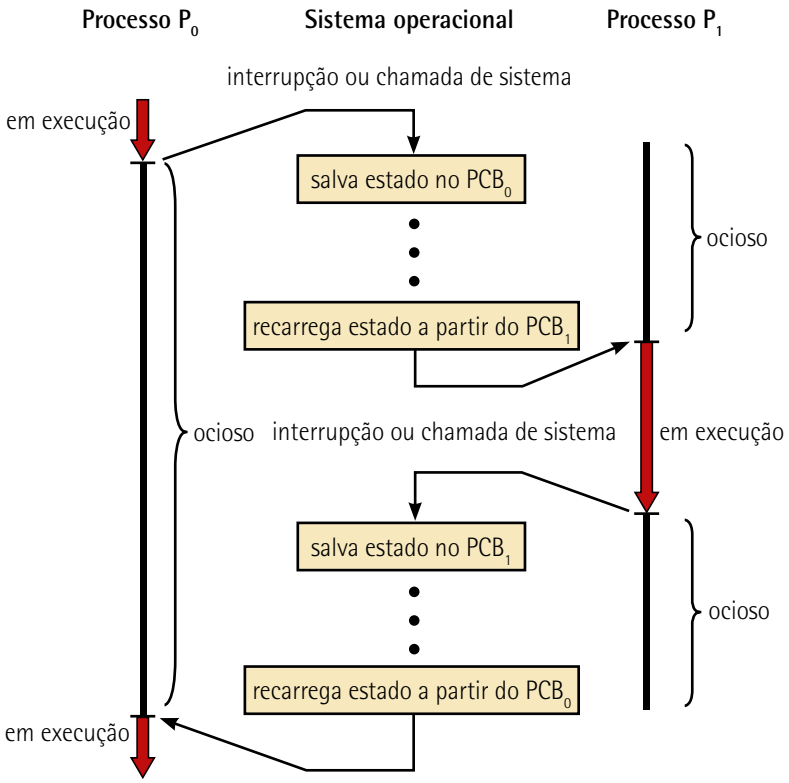

Figura 23 – Fluxo de atividades na UCP para a troca de processos

Fonte: Silberschatz, Galvin e Gagne (2015, p. 30).

A mudança de contexto leva a um overhead de tempo, sendo considerada uma tarefa cara. É preciso salvar as informações do processo cuja execução será interrompida na UCP, tais como os valores dos registradores e o status do processo em seu bloco de controle de processos ou process control block (PCB). Para o processo que será executado, deve-se carregar as informações relevantes.

Observação

O bloco de controle de processos inclui uma tabela que guarda informações de diversos contextos, como do gerenciamento de processos, representados por registros, contador de programa ou program counter (PC), ponteiro da pilha ou stack pointer (SP), identificador do processo (Pid) e prioridade.

4.2 Threads

Um subconjunto de um processo é denominado **thread** e representa os fluxos de execução dos processos. As threads compartilham entre si o mesmo dado de endereçamento e dados da tabela de processos PCB. Seu objetivo é contribuir com a execução de uma aplicação ao associar diversos fluxos a somente um processo.

Um dos benefícios da utilização de threads é o desempenho superior, porque existe a execução de tarefas simultânea do mesmo processo. Adicionalmente, as arquiteturas de processadores modernos, que têm múltiplos núcleos e permitem múltiplos processos simultâneos, combinam com o uso de threads, pois é possível aplicar um paralelismo real.

Um servidor de arquivos com um único fluxo realiza uma requisição e espera o resultado. O mesmo servidor de arquivos com múltiplos fluxos pode atender a solicitações de outros usuários, aumentando o desempenho e a saída de dados (throughput).

O chaveamento de processos é computacionalmente caro, pois envolve salvar e carregar os dados de contexto de processos. Outra consideração relevante sobre threads é que, na maioria dos sistemas, o tempo para criação e término de uma thread pode ser até 100 vezes mais rápido quando comparado ao de um processo (Tanenbaum; Steen, 2007).

Em sistemas distribuídos, threads do kernel permitem chamadas bloqueantes sem obstruir todo o processo. As vantagens das threads para sistemas distribuídos é que elas permitem a criação de múltiplas conexões, cada um sendo implementado por uma thread. Com isso, é possível ocultar a latência da comunicação na rede e gerar transparência para os usuários.

Um exemplo de clientes multithread são os navegadores ou browsers web. As requisições são feitas sem que todos os objetos tenham chegado à máquina do cliente. Assim, o cliente pode manipular fluxos em paralelo usando as threads – por exemplo, uma thread pode carregar o texto da página, outra o som e outra as imagens atuando em paralelo. A vantagem ao usuário é que não é necessário esperar até que todos os componentes das páginas cheguem.

4.2.1 Threads em sistemas distribuídos

Uma característica útil das threads para a comunicação em sistemas distribuídos é que elas permitem o bloqueio de chamadas de sistemas sem precisar do processo como um todo. Isso ocorre porque as threads viabilizam um meio de manter múltiplas conexões lógicas no momento.

Enquanto processos diferentes utilizam espaços de armazenamento distintos, as threads compartilham o mesmo espaço de armazenamento. Com isso, o desenvolvimento de aplicações multithreads nativas torna-se mais complexo, dado que o dilema da concorrência se torna mais claro e necessita de mecanismos adicionais, como programadores com maior expertise para prover o controle dos recursos compartilhados e da consistência dos dados resultantes das aplicações.

Existem vantagens do uso de threads em sistemas distribuídos: elas possibilitam a criação de diversas conexões e cada uma é implementada por um thread. Dessa forma, gera-se transparência e oculta-se a latência de comunicação da rede. É possível iniciar com a comunicação e logo depois realizar outra tarefa.

Para diminuir o impacto da troca de contexto, criam-se as threads híbridas ou lightweith process (LWP), e então cada conjunto de threads é mapeado em um número n de threads do kernel.

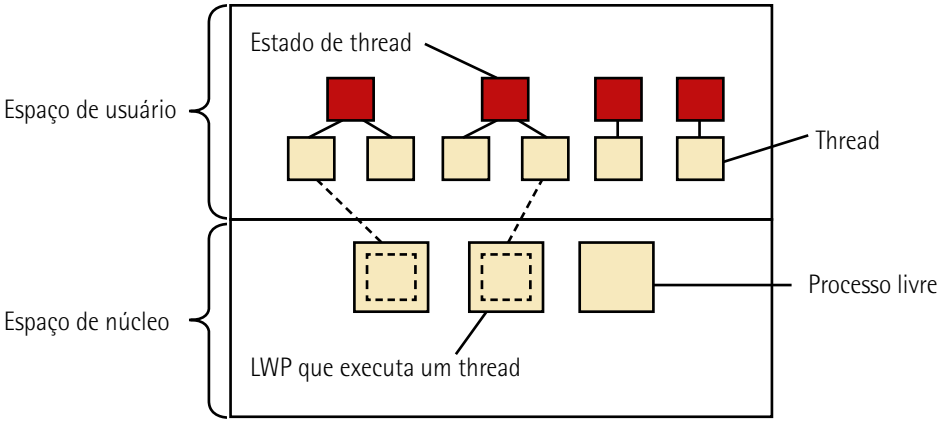

Figura 24 – Combinação de processos leves

Fonte: Tanenbaum e Steen (2007, p. 45).

É possível entender threads e processos como algo que permite mais de uma instrução ser executada simultaneamente. Um computador com um único processador fornece essa ilusão. Como existe uma única UCP, somente uma instrução de um único thread ou processo será executada a cada instante do processador. Esse aparente paralelismo é criado pelo chaveamento rápido ou troca de contextos entre threads e processos.

4.2.2 Clientes multithread

Uma das razões para usar clientes multithreads em sistemas distribuídos é ocultar tempos longos de propagação de mensagens entre processos em redes de alta e ampla disponibilidade. Assim, pode-se oferecer um alto nível de transparência de distribuição. Além disso, deve-se considerar que o tempo para concluir cada operação muitas vezes é relevante, podendo se tornar um problema.

Um exemplo recorrente de clientes multithreads são os navegadores web, que normalmente começam buscando uma página hypertext markup language (HTML) e as apresentam depois. Um documento web é composto de um arquivo HTML e, para realizar a busca de cada elemento, precisa de uma conexão transmission control protocol (TCP)/internet protocol (IP), da leitura da entrada de dados e do envio de dados para um componente de exibição.

Com o objetivo de reduzir a latência de comunicação, navegadores web mostram os dados conforme eles chegam. Depois de disponibilizar o texto ao usuário, o navegador realiza buscas adicionais dos outros arquivos que constituem a página web relacionada. Com isso, o usuário obtém resultados parciais enquanto espera que todos os componentes que compõem a página sejam pesquisados.

Com certeza você já se deparou com a seguinte situação: ao visitar um site, inicialmente apenas o texto é apresentado. Com o tempo, as imagens são carregadas; depois, banners, vídeos e áudios são disponibilizados.

Em geral, os navegadores web são executados como clientes multithread e viabilizam a abertura de várias conexões simultaneamente, inclusive abrindo diversas guias de navegação.

4.2.3 Servidores multithread

Além da implementação de clientes multithread, pode-se aplicar um conceito análogo nos servidores, gerando servidores multithread. A execução de mecanismos mutltithreads auxilia a desenvolver o servidor, provendo uma simplificação considerável no código e viabilizando meios que facilitam a exploração do paralelismo, com o intuito de obter o melhor desempenho possível.

Pense em um servidor de arquivos que, eventualmente, apresentará pedidos de bloqueio enquanto aguarda alguma operação no disco. O sistema de arquivos irá aguardar uma solicitação de operação em um arquivo, tanto de leitura como de escrita; depois, irá executar a solicitação e enviar os resultados desejados.

Uma das formas de organizar o servidor multithread é o modelo despachante/operário. Nele, a thread despachante ou dispatcher thread realiza as leituras das requisições originadas pelos clientes através da rede referentes a potenciais operações em um arquivo. Após a análise dessas solicitações, a thread despachante seleciona uma thread operária ou worker thread para tratar cada uma das requisições. Ao término, os resultados são transmitidos. Caso todas as threads operárias já estejam ocupadas, o thread despachante poderá ser escolhido para fazer a execução.

A figura a seguir apresenta o diagrama desse tipo de servidor multithread.

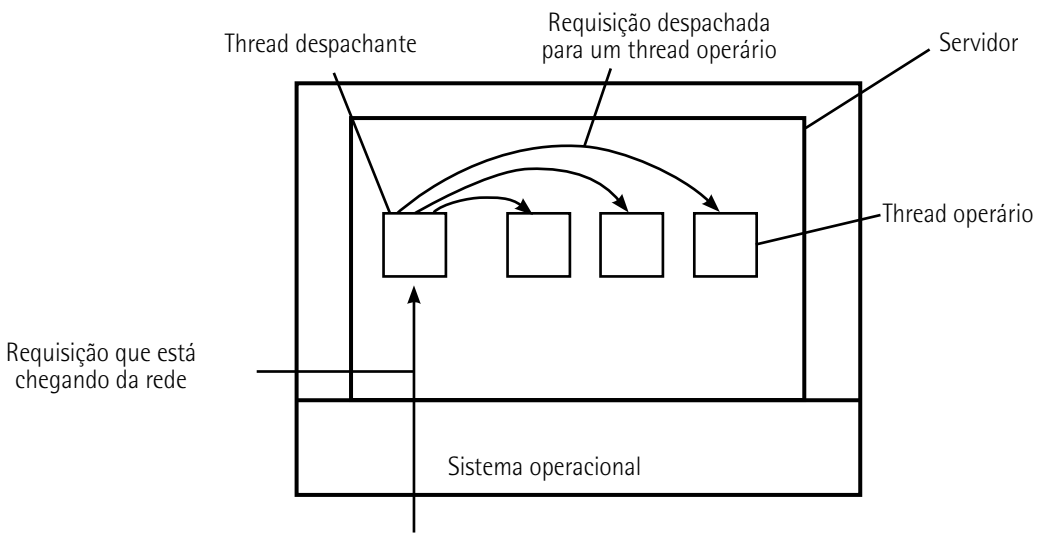

Figura 25 – Servidor multithread organizado segundo o modelo despachante/operário

Fonte: Tanenbaum e Steen (2007, p. 46).

Uma alternativa de implementação é o servidor monothread, que irá operar somente com uma thread de cada vez. No caso de um servidor de arquivos, este recebe a requisição de um cliente, depois

realiza a análise e então executa a solicitação até a sua conclusão. Enquanto espera pelo disco rígido, o servidor permanece ocioso, sem realizar nenhum processamento e sem receber nenhuma solicitação.

Adicionalmente, se o servidor de arquivos estiver rodando em uma máquina dedicada, o processador estará ocioso durante a espera da resposta do disco. Com isso, o número de solicitações tratadas será menor que a implementação multithread, tornando evidente o ganho de desempenho com o paralelismo.

4.3 Virtualização

Segundo Laureano (2006), o conceito de máquina virtual surgiu ainda no início da história dos computadores, entre o fim dos anos 1950 e início da década de 1960. Originalmente, as máquinas virtuais foram desenvolvidas para centralizar os sistemas computacionais usados no ambiente VM/370 da IBM e tiveram êxito. Para esse sistema, cada máquina virtual simula uma réplica física da máquina real e os usuários têm a impressão de que têm o sistema de forma exclusiva.

Como existe um desenvolvimento independente entre projetistas de hardware, sistemas operacionais e aplicações, que podem ocorrer em momentos e em empresas diferentes, esses trabalhos independentes resultam em diversas plataformas operacionais diferentes e que podem não ser compatíveis entre si. Uma plataforma operacional representa a união entre hardware, sistema operacional e aplicações. Assim, aplicações escritas para uma plataforma operacional não funcionam em outras plataformas.

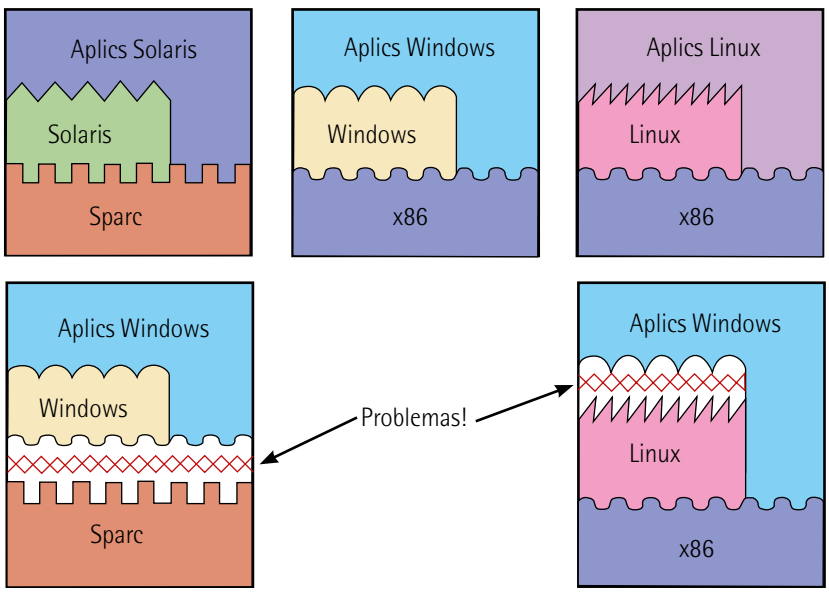

Figura 26 – Incompatibilidade entre plataformas operacionais

Adaptada de: Smith e Nair (2004).

Com a introdução de máquina virtual, o problema de incompatibilidade é resolvido, pois ela introduz uma camada intermediária entre o hardware e o sistema operacional para compatibilizar diferentes plataformas. O processo para criar essa camada adicional é denominado **virtualização**. Os sistemas

de virtualização substituem ou estendem uma interface existente para simular o comportamento de outro sistema.

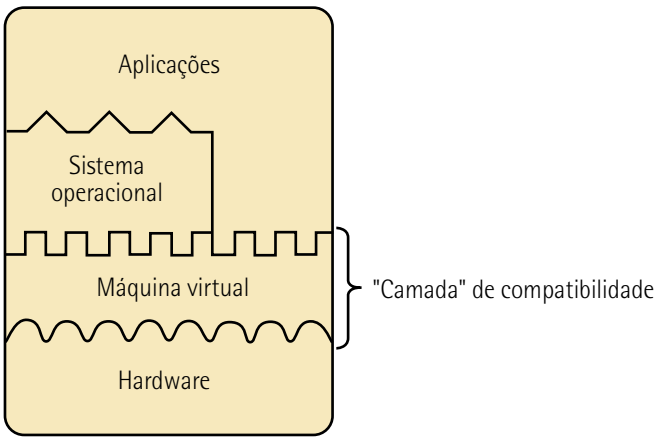

Figura 27 – Introdução da camada de virtualização

Fonte: Laureano (2006, p. 17).

Em servidores monothread, a virtualização de recursos permite que um único processador seja capaz de simular vários núcleos de execução, os quais podem se expandir e atender a outros recursos. Portanto, a ideia básica da virtualização de recursos é considerar que, hipoteticamente, há um recurso replicado no sistema.

De acordo com Tanenbaum e Bos (2016), um benefício dessa implementação é que o código da JVM poderá ser executado por qualquer computador conectado à internet que tenha um JVM. Caso o compilador fosse produzido por programas binários em SPARC ou Pentium, por exemplo, eles não poderiam ser enviados e executados em qualquer lugar de forma fácil. Além disso, a segurança dos programas JVM recebidos pelo usuário poderá ser verificada e o software poderá ser executado em um ambiente seguro, evitando roubo de dados ou danos no computador do usuário.

O software de nível mais alto, isto é, mais próximo ao usuário final, é mais estável e não é alterado menos frequentemente que hardwares e sistemas de software de baixo nível. Com a virtualização, o middleware pode contribuir com o transporte das interfaces de software para novas plataformas, as quais terão capacidade de executar os softwares existentes anteriormente.

Com a virtualização de recursos, é possível criar a impressão que determinado recurso está replicado no sistema. Para imitar o comportamento das interfaces são utilizadas instruções de máquinas, que são denominadas chamadas de sistema ou system calls.

Sistemas computacionais disponibilizam quatro tipos de interface em quatro níveis diferentes. No primeiro nível, existe uma interface entre o hardware e o software, composto de instruções de máquina. Tais comandos podem ser invocados por qualquer programa. Há uma segunda interface entre hardware e software, na qual existe um grupo mais restrito de instruções de máquinas que somente pode ser

invocado por programas privilegiados, como o sistema operacional. Instruções privilegiadas têm acesso ao hardware do sistema computacional e não são acessíveis diretamente pelas aplicações.

Outra interface existente entre o sistema operacional e a biblioteca são as system calls, que são oferecidas pelo sistema operacional. A interface entre a biblioteca e a aplicação é formada por chamadas de biblioteca ou funções de biblioteca, que são conhecidas como interface de aplicação de programação ou application programming interface (API), em inglês.

O isolamento de cada aplicação em sua área de memória designada pela unidade de gerenciamento de memória (memory management unit – MMU) proporciona robustez e confiabilidade ao sistema, pois garante que uma aplicação não poderá interferir nas áreas de memória. Entretanto, essa proteção introduz um dilema (Maziero, 2019): como a aplicação atuará para invocar as rotinas disponíveis pelo núcleo para o acesso ao hardware e demais serviços do SO?

O código do núcleo reside em uma área de memória inacessível à aplicação, e a ativação de uma rotina do núcleo utilizando esse mecanismo é conhecida como system call, abreviada como syscall.

Os sistemas operacionais definem chamadas de sistema para todas as operações envolvendo o acesso a recursos de baixo nível (periféricos, arquivos, alocação de memória etc.) ou abstrações lógicas (criação e encerramento de tarefas, operadores de sincronização). Geralmente, as chamadas de sistema são oferecidas para as aplicações em modo usuário por meio de uma system library, que prepara os parâmetros, invoca a chamada e, no retorno desta, devolve à aplicação os resultados obtidos.

A figura a seguir ilustra essas interfaces e os níveis do sistema computacional:

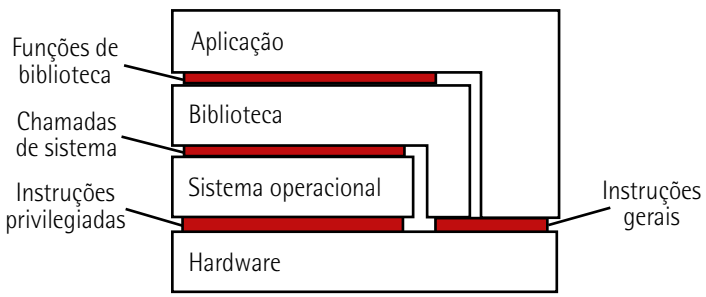

Figura 28 – As diversas interfaces de sistemas computacionais

Fonte: Tanenbaum e Steen (2007, p. 48).

O fundamento da virtualização é a imitação do comportamento dessas interfaces. A figura a seguir acentua a organização do hardware, da interface e do sistema e a ordenação simplificada da virtualização do sistema A sobre o sistema B:

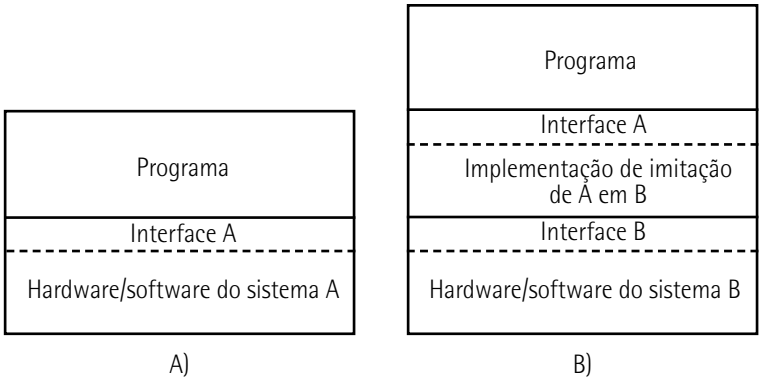

Figura 29 – A) Organização geral entre programa, interface e sistema.
B) Organização geral da virtualização do sistema A sobre o sistema B

Fonte: Tanenbaum e Steen (2007, p. 48).

Podemos classificar as máquinas virtuais em máquina virtual de processo ou monitor de máquina virtual. Na primeira, as aplicações que foram desenvolvidas para um sistema operacional são executadas em um sistema operacional diferente. A virtualização ocorre somente em um único processo. Por exemplo, é possível executar aplicações Windows na distribuição Linux Ubuntu.

Outra aplicação de **máquinas virtuais de processos** é na execução de programas Java. A linguagem Java foi criada pela Sun Microsystems, que também elaborou a máquina virtual Java ou Java virtual machine (JVM). Quando o compilador Java deseja rodar um código para a JVM, ele é executado por um interpretador JVM em software.

No **monitor de máquina virtual** ou virtual machine monitor (VMM), o hardware terá à disposição um conjunto de instruções completo. Diversos sistemas operacionais podem ser executados independente e concorrentemente na mesma plataforma.

O software VMWare, XenProject.org e o VirtualBox são exemplos de monitores de máquina virtual.

Saiba mais

Um software usado para virtualização e que pode executar diferentes máquinas virtuais é o Oracle VM VirtualBox. O usuário pode instalar e rodar quantas máquinas virtuais desejar. Contudo, existe a limitação do espaço em disco rígido e exige-se memória para a virtualização.

Ele pode ser baixado no site a seguir:

Disponível em: https://www.virtualbox.org. Acesso em: 15 abr. 2024.

Considerando a necessidade de segurança e confiabilidade de sistemas distribuídos, essa implementação ganhou ainda mais importância. Dado o isolamento entre uma aplicação completa e o seu ambiente, um ataque à segurança na máquina virtual não irá causar efeito na máquina física. Uma falha causada por um erro na máquina virtual não irá afetar a máquina física. Adicionalmente, a portabilidade é ainda maior nos monitores de máquina virtual, pois o hardware e o software estão mais desacoplados, permitindo que um ambiente completo seja transferido de uma máquina para outra.

A figura a seguir apresenta as interfaces nos diferentes tipos de máquina virtual. Em (a), é mostrada a máquina virtual de processo, que fornece um conjunto de instruções abstratas e que podem ser invocadas na execução das aplicações. Em (b), o monitor de máquina virtual nativa é uma implementação realizada diretamente na camada acima do hardware subjacente, isto é, a máquina física. Em (c), o monitor de máquina virtual hospedeira é executado sobre o sistema operacional hospedeiro, ou seja, é uma camada que é executada sobre o sistema operacional instalado da máquina física.

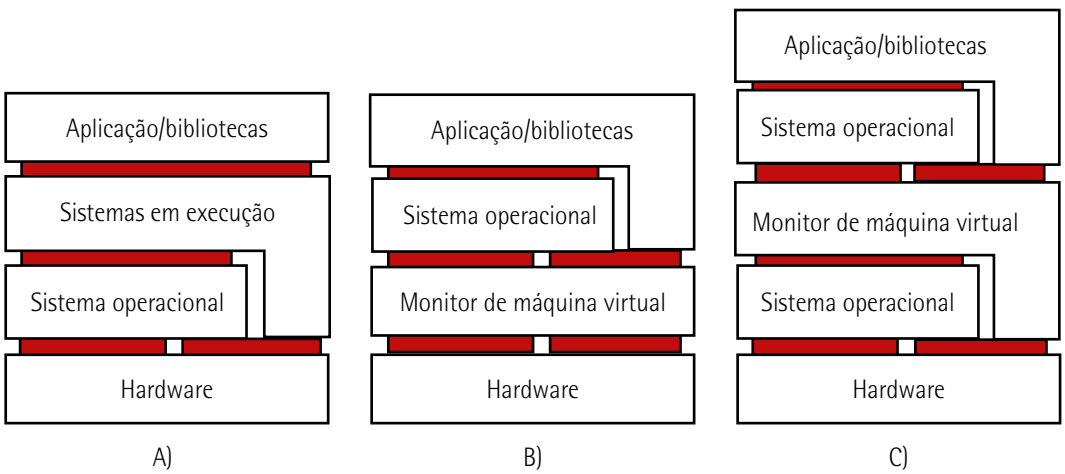

Figura 30 – Interfaces nos diferentes tipos de máquinas virtuais

Fonte: Monteiro *et al.* (2020, p. 93).

Resumo

Nesta unidade, foram apresentados os sistemas distribuídos. Seus componentes de hardware ou software estão situados em máquinas distintas e interconectadas em rede. Esses computadores se comunicam e fazem a coordenação de suas atividades com o envio de mensagem entre eles. Diversas aplicações, como jogos online, bancos de dados distribuídos, pesquisas na web e serviços de comunicação instantânea como WhatsApp utilizam os conceitos de sistemas distribuídos.

É possível dividir os tipos de sistemas distribuídos em sistemas distribuídos de alto desempenho, como clusters e grids, computação em nuvem, sistemas de informação distribuídos e sistemas distribuídos pervasivos.

Vimos que a escolha da arquitetura de sistemas distribuídos contribui para determinar quais são as entidades que irão se comunicar, o paradigma de comunicação aplicado ou como essas entidades se comunicam, as funções e responsabilidades referentes aos componentes do sistema e a localização de cada um. A comunicação é um fator essencial nesse contexto, pois os diferentes componentes devem se comunicar para que a computação seja realizada conjuntamente.

As arquiteturas de sistemas distribuídos podem ser centralizadas, descentralizadas e híbridas. A primeira usa a configuração cliente-servidor, na qual cliente e servidor rodam processos diferentes e em máquinas distintas. O padrão de comunicação é requisição-resposta, isto é, o cliente faz uma requisição ao servidor, que irá processar e comunicar a resposta para esse cliente. A maior parte de serviços web utilizam arquiteturas centralizadas, como serviços de e-mail, HTTP e serviços de nomes de domínios (DNS).

Já as arquiteturas descentralizadas são muito usadas como redes de distribuição de conteúdo, nas quais cada nó da rede será denominado par ou peer e será cliente ou servidor em momentos diferentes. Elas podem ser classificadas em estruturadas, não estruturadas e superpares.

Por sua vez, arquiteturas híbridas combinam características das centralizadas e descentralizadas e são usadas em aplicações como BitTorrent, Skype e WhatsApp.

Em sistemas distribuídos, haverá o processamento de diferentes processos em máquinas distintas. Os processos podem ser organizados com fluxos de execução denominados threads. A utilização de threads

contribui para a execução paralela dos processos e diversas aplicações são capazes de terem o processamento de múltiplas threads simultâneas, pois existem aplicações multithread nativas, clientes multithread e servidores multithread.

As máquinas virtuais ou virtual machines (VMs) permitem a execução de processos para plataformas diferentes das quais foram projetados e inserem uma camada de compatibilização entre o hardware e o sistema operacional. Podemos separar as VMs em máquinas virtuais em máquinas virtuais de processo ou monitor de máquina virtual.

Exercícios

Questão 1. (Ibade/2020, adaptada) O conceito de transparência representa uma das características dos sistemas distribuídos. A transparência consiste em esconder do usuário final e das aplicações o fato de que os processos e os recursos estão fisicamente distribuídos por múltiplos computadores.

A transparência pode ser dividida em tipos. O tipo de transparência que esconde do usuário as diferenças da representação de dados, e no modo como um recurso é acessado é chamado de:

A) Transparência de localização.

B) Transparência de escalabilidade.

C) Transparência de falhas.

D) Transparência de acesso.

E) Transparência de concorrência.

Resposta correta: alternativa D.

Análise das alternativas

A) Alternativa incorreta.

Justificativa: a transparência de localização dita que, em um sistema distribuído, recursos sejam acessados sem o conhecimento da sua localização física.

B) Alternativa incorreta.

Justificativa: a transparência de escalabilidade permite que o sistema se expanda sem alterar sua estrutura ou os algoritmos de aplicação.

C) Alternativa incorreta.

Justificativa: a transparência de falhas busca ocultar as falhas e as recuperações de partes do sistema, para permitir a continuidade do serviço mesmo na ocorrência de problemas.

D) Alternativa correta.

Justificativa: quando dados são transmitidos entre máquinas com arquiteturas ou com sistemas operacionais distintos, a transparência de acesso oculta as diferenças na representação dos dados e no acesso aos recursos. Logo, o trecho apresentado no enunciado diz respeito à transparência de acesso.

E) Alternativa incorreta.

Justificativa: a transparência de concorrência possibilita que diversos processos operem concorrentemente, utilizando recursos compartilhados e sem interferência entre eles.

Questão 2. (Iades/2019, adaptada) A computação em nuvem é uma área recente da computação que trabalha com a transferência de informação e o acesso de arquivos a distância. A respeito da computação em nuvem, avalie as afirmativas:

I – Existem empresas especializadas em disponibilizar serviços em nuvem.

II – A computação em nuvem permite o armazenamento de dados com capacidade infinita, sem alterar o valor cobrado pelo serviço.

III – Não existem softwares para serem instalados localmente, tendo em vista que a computação em nuvem visa à descentralização geográfica.

É correto o que se afirma em:

A) I, apenas.

B) III, apenas.

C) I e III, apenas.

D) II e III, apenas.

E) I, II e III.

Resposta correta: alternativa A.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: há diversas empresas especializadas em fornecer serviços de computação em nuvem. A Amazon oferece o Amazon Web Services (AWS), a Google oferece o Google Cloud e a Microsoft oferece o Azure, por exemplo.

II – Afirmativa incorreta.

Justificativa: a computação em nuvem entrega recursos de TI sob demanda por meio da internet, com preço definido pelo uso. Assim, o valor cobrado pelo serviço é uma função da utilização demandada. Além disso, embora a computação em nuvem permita grande capacidade de armazenamento de dados, essa capacidade não é infinita.

III – Afirmativa incorreta.

Justificativa: embora a computação em nuvem permita o acesso a aplicativos e a dados de qualquer lugar, alguns softwares podem precisar ser instalados localmente para acessar certos serviços em nuvem.