1

Caracterização de Sistemas Distribuídos

Um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens. Essa definição leva às seguintes características especialmente importantes dos sistemas distribuídos: concorrência de componentes, falta de um relógio global e falhas de componentes independentes.

Examinaremos vários exemplos de aplicações distribuídas modernas, incluindo pesquisa na Web, jogos *online* para vários jogadores e sistemas de negócios financeiros. Veremos também as tendências básicas que estimulam o uso dos sistemas distribuídos atuais: a natureza pervasiva da interligação em rede moderna, o florescimento da computação móvel e ubíqua, a crescente importância dos sistemas multimídia distribuídos e a tendência no sentido de considerar os sistemas distribuídos como um serviço público. Em seguida, o capítulo destacará o compartilhamento de recursos como uma forte motivação para a construção de sistemas distribuídos. Os recursos podem ser gerenciados por servidores e acessados por clientes, ou podem ser encapsulados como objetos e acessados por outros objetos clientes.

Os desafios advindos da construção de sistemas distribuídos são a heterogeneidade dos componentes, ser um sistema aberto, o que permite que componentes sejam adicionados ou substituídos, a segurança, a escalabilidade – isto é, a capacidade de funcionar bem quando a carga ou o número de usuários aumenta –, o tratamento de falhas, a concorrência de componentes, a transparência e o fornecimento de um serviço de qualidade. Por fim, a Web será discutida como exemplo de sistema distribuído de grande escala e serão apresentados seus principais recursos.

1.1 Introdução

As redes de computadores estão por toda parte. A Internet é uma delas, assim como as muitas redes das quais ela é composta. Redes de telefones móveis, redes corporativas, redes de fábrica, redes em campus, redes domésticas, redes dentro de veículos, todas elas, tanto separadamente como em conjunto, compartilham as características básicas que as tornam assuntos relevantes para estudo sob o título *sistemas distribuídos*. Neste livro, queremos explicar as características dos computadores interligados em rede que afetam os projetistas e desenvolvedores de sistema e apresentar os principais conceitos e técnicas que foram criados para ajudar nas tarefas de projeto e implementação de sistemas que os têm por base.

Definimos um sistema distribuído como aquele no qual os componentes de *hardware* ou *software*, localizados em computadores interligados em rede, comunicam-se e coordenam suas ações apenas enviando mensagens entre si. Essa definição simples abrange toda a gama de sistemas nos quais computadores interligados em rede podem ser distribuídos de maneira útil.

Os computadores conectados por meio de uma rede podem estar separados por qualquer distância. Eles podem estar em continentes separados, no mesmo prédio ou na mesma sala. Nossa definição de sistemas distribuídos tem as seguintes consequências importantes:

Concorrência: em uma rede de computadores, a execução concorrente de programas é a norma. Posso fazer meu trabalho em meu computador, enquanto você faz o seu em sua máquina, compartilhando recursos como páginas Web ou arquivos, quando necessário. A capacidade do sistema de manipular recursos compartilhados pode ser ampliada pela adição de mais recursos (por exemplo, computadores) na rede. Vamos descrever como essa capacidade extra pode ser distribuída em muitos pontos de maneira útil. A coordenação de programas em execução concorrente e que compartilham recursos também é um assunto importante e recorrente.

Inexistência de relógio global: quando os programas precisam cooperar, eles coordenam suas ações trocando mensagens. A coordenação frequentemente depende de uma noção compartilhada do tempo em que as ações dos programas ocorrem. Entretanto, verifica-se que existem limites para a precisão com a qual os computadores podem sincronizar seus relógios em uma rede – não existe uma noção global única do tempo correto. Essa é uma consequência direta do fato de que a *única* comunicação se dá por meio do envio de mensagens em uma rede. Exemplos desses problemas de sincronização e suas soluções serão descritos no Capítulo 14.

Falhas independentes: todos os sistemas de computador podem falhar, e é responsabilidade dos projetistas de sistema pensar nas consequências das possíveis falhas. Nos sistemas distribuídos, as falhas são diferentes. Falhas na rede resultam no isolamento dos computadores que estão conectados a ela, mas isso não significa que eles param de funcionar. Na verdade, os programas neles existentes talvez não consigam detectar se a rede falhou ou se ficou demasiadamente lenta. Analogamente, a falha de um computador ou o término inesperado de um programa em algum lugar no sistema (um *colapso no sistema*) não é imediatamente percebida pelos outros componentes com os quais ele se comunica. Cada componente do sistema pode falhar independentemente, deixando os outros ainda em funcionamento. As consequências dessa característica dos sistemas distribuídos serão um tema recorrente em todo o livro.

A principal motivação para construir e usar sistemas distribuídos é proveniente do desejo de compartilhar recursos. O termo "recurso" é bastante abstrato, mas caracteriza

bem o conjunto de coisas que podem ser compartilhadas de maneira útil em um sistema de computadores interligados em rede. Ele abrange desde componentes de *hardware*, como discos e impressoras, até entidades definidas pelo *software*, como arquivos, bancos de dados e objetos de dados de todos os tipos. Isso inclui o fluxo de quadros de vídeo proveniente de uma câmera de vídeo digital ou a conexão de áudio que uma chamada de telefone móvel representa.

O objetivo deste capítulo é transmitir uma visão clara da natureza dos sistemas distribuídos e dos desafios que devem ser enfrentados para garantir que eles sejam bem-sucedidos. A Seção 1.2 fornece alguns exemplos ilustrativos de sistemas distribuídos, e a Seção 1.3 aborda as principais tendências subjacentes que estimulam os recentes desenvolvimentos. A Seção 1.4 enfoca o projeto de sistemas com compartilhamento de recursos, enquanto a Seção 1.5 descreve os principais desafios enfrentados pelos projetistas de sistemas distribuídos: heterogeneidade, sistemas abertos, segurança, escalabilidade, tratamento de falhas, concorrência, transparência e qualidade do serviço. A Seção 1.6 apresenta o estudo de caso detalhado de um sistema distribuído bastante conhecido, a World Wide Web, ilustrando como seu projeto suporta o compartilhamento de recursos.

1.2 Exemplos de sistemas distribuídos

O objetivo desta seção é dar exemplos motivacionais de sistemas distribuídos atuais, ilustrando seu papel predominante e a enorme diversidade de aplicações associadas a eles.

Conforme mencionado na Introdução, as redes estão por toda parte e servem de base para muitos serviços cotidianos que agora consideramos naturais; por exemplo, a Internet e a World Wide Web associada a ela, a pesquisa na Web, os jogos *online*, os *e-mails*, as redes sociais, o *e-Commerce* etc. Para ilustrar ainda mais esse ponto, considere a Figura 1.1, que descreve uma variedade de setores de aplicação comercial ou social importantes, destacando alguns usos associados estabelecidos ou emergentes da tecnologia de sistemas distribuídos.

Como se vê, os sistemas distribuídos abrangem muitos dos desenvolvimentos tecnológicos mais significativos atualmente e, portanto, um entendimento da tecnologia subjacente é absolutamente fundamental para o conhecimento da computação moderna. A figura também dá um vislumbre inicial da ampla variedade de aplicações em uso hoje, desde sistemas de localização relativa, conforme os encontrados, por exemplo, em um carro ou em um avião, até sistemas de escala global envolvendo milhões de nós; desde serviços voltados para dados até tarefas que exigem uso intenso do processador; desde sistemas construídos a partir de sensores muito pequenos e relativamente primitivos até aqueles que incorporam elementos computacionais poderosos; desde sistemas embarcados até os que suportam uma sofisticada experiência interativa do usuário e assim por diante.

Vamos ver agora exemplos de sistemas distribuídos mais específicos para ilustrar melhor a diversidade e a real complexidade da provisão de sistemas distribuídos atual.

1.2.1 Pesquisa na Web

A pesquisa na Web tem se destacado como um setor crescente na última década, com os valores recentes indicando que o número global de pesquisas subiu para mais de 10 bilhões por mês. A tarefa de um mecanismo de pesquisa na Web é indexar todo o conteúdo da World Wide Web, abrangendo uma grande variedade de estilos de informação, incluindo páginas Web, fontes de multimídia e livros (escaneados). Essa é uma tarefa muito complexa, pois as estimativas atuais mostram que a Web consiste em mais de 63 bilhões

Finanças e comércio	O crescimento do *e-Commerce*, exemplificado por empresas como Amazon e eBay, e as tecnologias de pagamento subjacentes, como PayPal; o surgimento associado de operações bancárias e negócios *online* e também os complexos sistemas de disseminação de informações para mercados financeiros.
A sociedade da informação	O crescimento da World Wide Web como repositório de informações e de conhecimento; o desenvolvimento de mecanismos de busca na Web, como Google e Yahoo, para pesquisar esse amplo repositório; o surgimento de bibliotecas digitais e a digitalização em larga escala de fontes de informação legadas, como livros (por exemplo, Google Books); a importância cada vez maior do conteúdo gerado pelos usuários por meio de *sites* como YouTube, Wikipedia e Flickr; o surgimento das redes sociais por meio de serviços como Facebook e MySpace.
Setores de criação e entretenimento	O surgimento de jogos *online* como uma forma nova e altamente interativa de entretenimento; a disponibilidade de músicas e filmes nos lares por meio de centros de mídia ligados em rede e mais amplamente na Internet, por intermédio de conteúdo que pode ser baixado ou em *streaming*; o papel do conteúdo gerado pelos usuários (conforme mencionado anteriormente) como uma nova forma de criatividade, por exemplo por meio de serviços como YouTube; a criação de novas formas de arte e entretenimento permitidas pelas tecnologias emergentes (inclusive em rede).
Assistência médica	O crescimento da informática médica como disciplina, com sua ênfase nos registros eletrônicos de pacientes *online* e nos problemas de privacidade relacionados; o papel crescente da telemedicina no apoio ao diagnóstico remoto ou a serviços mais avançados, como cirurgias remotas (incluindo o trabalho colaborativo entre equipes médicas); a aplicação cada vez maior de interligação em rede e tecnologia de sistemas incorporados em vida assistida; por exemplo, para monitorar idosos em suas próprias residências.
Educação	O surgimento do *e-learning* por meio, por exemplo, de ferramentas baseadas na Web, como os ambientes de ensino virtual; suporte associado ao aprendizado à distância; suporte ao aprendizado colaborativo ou em comunidade.
Transporte e logística	O uso de tecnologias de localização, como o GPS, em sistemas de descoberta de rotas e sistemas de gerenciamento de tráfego mais gerais; o próprio carro moderno como exemplo de sistema distribuído complexo (também se aplica a outras formas de transporte, como a aviação); o desenvolvimento de serviços de mapa baseados na Web, como MapQuest, Google Maps e Google Earth.
Ciências	O surgimento das grades computacionais (*grid*) como tecnologia fundamental para eScience, incluindo o uso de redes de computadores complexas para dar suporte ao armazenamento, à análise e ao processamento de dados científicos (frequentemente em volumes muito grandes); o uso associado das grades como tecnologia capacitadora para a colaboração entre grupos de cientistas do mundo inteiro.
Gerenciamento ambiental	O uso de tecnologia de sensores (interligados em rede) para monitorar e gerenciar o ambiente natural; por exemplo, para emitir alerta precoce de desastres naturais, como terremotos, enchentes ou tsunamis, e para coordenar a resposta de emergência; o cotejamento e a análise de parâmetros ambientais globais para entender melhor fenômenos naturais complexos, como a mudança climática.

Figura 1.1 Domínios de aplicação selecionados e aplicações de rede associadas.

de páginas e um trilhão de endereços Web únicos. Como a maioria dos mecanismos de busca analisa todo o conteúdo da Web e depois efetua um processamento sofisticado nesse banco de dados enorme, essa tarefa representa, por si só, um grande desafio para o projeto de sistemas distribuídos.

O Google, líder de mercado em tecnologia de pesquisa na Web, fez um trabalho significativo no projeto de uma sofisticada infraestrutura de sistema distribuído para dar suporte à pesquisa (e, na verdade, a outros aplicativos e serviços do Google, como o Google Earth). Isso representa uma das maiores e mais complexas instalações de sistemas distribuídos da história da computação e, assim, exige um exame minucioso. Os destaques dessa infraestrutura incluem:

- Uma infraestrutura física subjacente consistindo em grandes números de computadores interligados em rede, localizados em centros de dados por todo o mundo.
- Um sistema de arquivos distribuído projetado para suportar arquivos muito grandes e fortemente otimizado para o estilo de utilização exigido pela busca e outros aplicativos do Google (especialmente a leitura de arquivos em velocidades altas e constantes).
- Um sistema associado de armazenamento distribuído estruturado que oferece rápido acesso a conjuntos de dados muito grandes.
- Um serviço de bloqueio que oferece funções de sistema distribuído, como bloqueio e acordo distribuídos.
- Um modelo de programação que suporta o gerenciamento de cálculos paralelos e distribuídos muito grandes na infraestrutura física subjacente.

Mais detalhes sobre os serviços de sistemas distribuídos do Google e o suporte de comunicação subjacente podem ser encontrados no Capítulo 21, que apresenta um interessante estudo de caso de um sistema distribuído moderno em ação.

1.2.2 Massively multiplayer online games (MMOGs)

Os jogos *online* com vários jogadores, ou MMOGs (Massively Multiplayer Online Games), oferecem uma experiência imersiva com a qual um número muito grande de usuários interage com um mundo virtual persistente pela Internet. Os principais exemplos desses jogos incluem o EverQuest II, da Sony, e o EVE Online, da empresa finlandesa CCP Games. Esses mundos têm aumentado significativamente em termos de sofisticação e agora incluem, por exemplo, arenas de jogo complexas (o EVE Online, por exemplo, consiste em um universo com mais de 5.000 sistemas estelares) e variados sistemas sociais e econômicos. O número de jogadores também está aumentando, com os sistemas capazes de suportar mais de 50.000 usuários *online* simultâneos (e o número total de jogadores talvez seja dez vezes maior).

A engenharia dos MMOGs representa um grande desafio para as tecnologias de sistemas distribuídos, particularmente devido à necessidade de tempos de resposta rápidos para preservar a experiência dos usuários do jogo. Outros desafios incluem a propagação de eventos em tempo real para muitos jogadores e a manutenção de uma visão coerente do mundo compartilhado. Portanto, esse é um excelente exemplo dos desafios enfrentados pelos projetistas dos sistemas distribuídos modernos.

Foram propostas várias soluções para o projeto de MMOGs:

- Talvez surpreendentemente, o maior jogo *online*, o EVE Online, utiliza uma arquitetura *cliente-servidor* na qual uma única cópia do estado do mundo é mantida em um servidor centralizado e acessada por programas clientes em execução nos consoles

ou em outros equipamentos dos jogadores. Para suportar grandes números de clientes, por si só o servidor é uma entidade complexa, consistindo em uma arquitetura de agregrado (*cluster*) caracterizada por centenas de nós de computador (essa estratégia cliente-servidor está discutida com mais detalhes na Seção 1.4 e as estratégias de *cluster* estão discutidas na Seção 1.3.4). A arquitetura centralizada ajuda significativamente no gerenciamento do mundo virtual e a cópia única também diminui as preocupações com a coerência. Assim, o objetivo é garantir resposta rápida por meio da otimização de protocolos de rede e também para os eventos recebidos. Para suportar isso, a carga é particionada por meio da alocação de *sistemas estelares* individuais para computadores específicos dentro do *cluster*, com os sistemas estelares altamente carregados tendo seu próprio computador dedicado e outros compartilhando um computador. Os eventos recebidos são direcionados para os computadores certos dentro do *cluster* por intermédio do monitoramento da movimentação dos jogadores entre os sistemas estelares.

- Outros MMOGs adotam arquiteturas mais distribuídas, nas quais o universo é particionado por um número potencialmente muito grande de servidores, os quais também pode estar geograficamente distribuídos. Então, os usuários são alocados dinamicamente a um servidor em particular com base nos padrões de utilização momentâneos e também pelos atrasos de rede até o servidor (com base na proximidade geográfica, por exemplo). Esse estilo de arquitetura, que é adotado pelo EverQuest, é naturalmente extensível pela adição de novos servidores.
- A maioria dos sistemas comerciais adota um dos dois modelos apresentados anteriormente, mas agora os pesquisadores também estão examinando arquiteturas mais radicais, que não se baseiam nos princípios cliente-servidor, mas adotam estratégias completamente descentralizadas, baseadas em tecnologia *peer-to-peer*, em que cada participante contribui com recursos (armazenamento e processamento) para hospedar o jogo. Mais considerações sobre as soluções *peer-to-peer* estão nos Capítulos 2 e 10).

1.2.3 Negócios financeiros

Como um último exemplo, examinemos o suporte dos sistemas distribuídos para os mercados de negócios financeiros. Há muito tempo, o setor financeiro está na vanguarda da tecnologia de sistemas distribuídos, em particular por sua necessidade de acesso em tempo real a uma ampla variedade de fontes de informação (preços atuais e tendências de ações, mudanças econômicas e políticas, etc). O setor emprega monitoramento automatizado e aplicativos comerciais (veja a seguir).

Note que a ênfase de tais sistemas está na comunicação e no processamento de itens de interesse, conhecidos como *eventos* nos sistemas distribuídos, e também na necessidade de distribuir eventos de forma confiável e de maneira oportuna para uma quantidade de clientes que têm interesse declarado nesses itens de informação. Exemplos de tais eventos incluem queda no preço de uma ação, o comunicado dos últimos resultados do desemprego e assim por diante. Isso exige um estilo de arquitetura subjacente muito diferente dos estilos mencionados anteriormente (por exemplo, cliente-servidor), e esses sistemas normalmente empregam os que são conhecidos como *sistemas distribuídos baseados em eventos*. Apresentamos uma ilustração de uso típico de tais sistemas a seguir e voltaremos a esse importante tópico com mais profundidade no Capítulo 6.

A Figura 1.2 ilustra um sistema de negócios financeiros típico. Ela mostra uma série de fontes de evento envolvidas em determinada instituição financeira. Essas fontes de

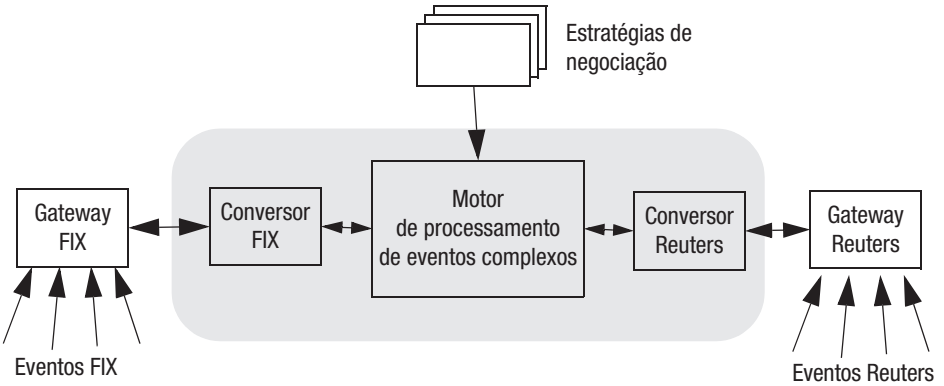

Figura 1.2 Um exemplo de sistema de negócios financeiros.

evento compartilham as seguintes características. Primeiramente, as fontes normalmente aparecem em formatos variados, como eventos de dados de mercado da Reuters e eventos FIX (que seguem o formato específico do protocolo Financial Information eXchange), e provêm de diferentes tecnologias de evento, ilustrando assim o problema da heterogeneidade encontrado na maioria dos sistemas distribuídos (consulte também a Seção 1.5.1). A figura mostra o uso de conversores que transformam formatos heterogêneos em um formato interno comum. Em segundo lugar, o sistema de negócios deve lidar com uma variedade de fluxos de evento, todos chegando em alta velocidade e frequentemente exigindo processamento em tempo real para se detectar padrões que indiquem oportunidades de negócios. Esse processo costumava ser manual, mas as pressões competitivas levaram a uma automação cada vez maior, nos termos do que é conhecido como CEP (Complex Event Processing), o qual oferece uma maneira de reunir ocorrências de evento em padrões lógicos, temporais ou espaciais.

Essa abordagem é usada principalmente no desenvolvimento de estratégias de negócios personalizadas, abrangendo tanto a compra como a venda de ações, em particular procurando padrões que indiquem uma oportunidade de negócio, respondendo automaticamente por fazer e gerenciar pedidos. Como exemplo, considere o seguinte *script*:

```
WHEN
    MSFT price moves outside 2% of MSFT Moving Average
FOLLOWED-BY (
    MyBasket moves up by 0.5%
    AND
        HPQ's price moves up by 5%
        OR
        MSFT's price moves down by 2%
    )
)
ALL WITHIN
    any 2 minute time period
THEN
    BUY MSFT
    SELL HPQ
```

Esse *script* é baseado na funcionalidade fornecida pelo Apama [www.progress.com], um produto comercial do mundo financeiro desenvolvido originalmente pela pesquisa feita na Universidade de Cambridge. Esse *script* detecta uma sequência temporal complexa, baseada nos preços de ações da Microsoft, da HP e uma série de outros preços de ação, resultando em decisões no sentido de comprar ou vender ações específicas.

Esse estilo de tecnologia está sendo cada vez mais usado em outras áreas dos sistemas financeiros, incluindo o monitoramento de atividade do negócio para o gerenciamento de risco (em particular, o controle de exposição), a garantia do cumprimento de normas e o monitoramento de padrões de atividade que possam indicar transações fraudulentas. Nesses sistemas, os eventos normalmente são interceptados e submetidos ao que é equivalente a um *firewall* de cumprimento e risco, antes de serem processados (consulte também a discussão sobre *firewalls* na Seção 1.3.1, a seguir).

1.3 Tendências em sistemas distribuídos

Os sistemas distribuídos estão passando por um período de mudança significativa e isso pode ser consequência de diversas tendências influentes:

- O surgimento da tecnologia de redes pervasivas.
- O surgimento da computação ubíqua, combinado ao desejo de suportar mobilidade do usuário em sistemas distribuídos.
- A crescente demanda por serviços multimídia.
- A visão dos sistemas distribuídos como um serviço público.

1.3.1 Interligação em rede pervasiva e a Internet moderna

A Internet moderna é um conjunto de redes de computadores interligadas, com uma variedade de tipos que aumenta cada vez mais e que agora inclui, por exemplo, uma grande diversidade de tecnologias de comunicação sem fio, como WiFi, WiMAX, Bluetooth (consulte o Capítulo 3) e redes de telefonia móvel de 3ª e 4ª geração. O resultado é que a interligação em rede se tornou um recurso pervasivo, e os dispositivos podem ser conectados (se assim for desejado) a qualquer momento e em qualquer lugar. A Figura 1.3 ilustra uma parte típica da Internet. Os programas que estão em execução nos computadores conectados a ela interagem enviando mensagens através de um meio de comunicação comum. O projeto e a construção dos mecanismos de comunicação da Internet (os protocolos Internet) são uma realização técnica importante, permitindo que um programa em execução em qualquer lugar envie mensagens para programas em qualquer outro lugar e abstraindo a grande variedade de tecnologias mencionadas anteriormente.

A Internet é um sistema distribuído muito grande. Ela permite que os usuários, onde quer que estejam, façam uso de serviços como a World Wide Web, *e-mail* e transferência de arquivos. Na verdade, às vezes, a Web é confundida como sendo a Internet. O conjunto de serviços da Internet é aberto – ele pode ser ampliado pela adição de computadores servidores e de novos tipos de serviços. A Figura 1.3 mostra um conjunto de intranets – sub-redes operadas por empresas e outras organizações, normalmente protegidas por *firewalls*. A função de um *firewall* é proteger uma intranet, impedindo a entrada

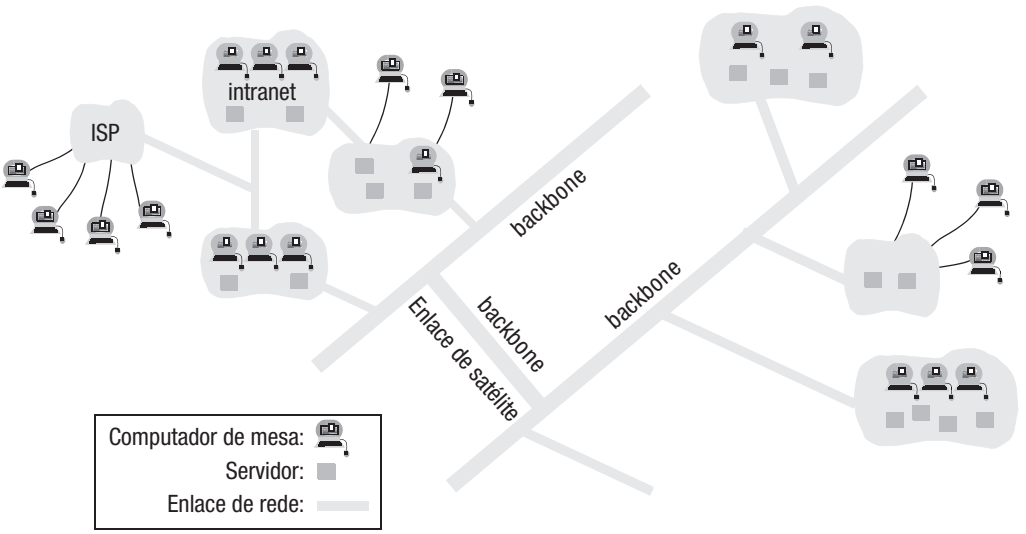

Figura 1.3 Uma parte típica da Internet.

ou a saída de mensagens não autorizadas. Um *firewall* é implementado pela filtragem de mensagens recebidas e enviadas, por exemplo, de acordo com sua origem ou destino. Um *firewall* poderia, por exemplo, permitir que somente mensagens relacionadas com *e-mail* e acesso a Web entrem ou saiam da intranet que ele está protegendo. Os provedores de serviços de Internet (ISPs, Internet Service Providers) são empresas que fornecem, através de *links* de banda larga e de outros tipos de conexões, o acesso a usuários individuais, ou organizações, à Internet. Esse acesso permite que os usuários utilizem os diferentes serviços disponibilizados pela Internet. Os provedores fornecem ainda alguns serviços locais, como *e-mail* e hospedagem de páginas Web. As intranets são interligadas por meio de *backbones*. Um *backbone* é um enlace de rede com uma alta capacidade de transmissão, empregando conexões via satélite, cabos de fibra óptica ou outros meios físicos de transmissão que possuam uma grande largura de banda.

Note que algumas empresas talvez não queiram conectar suas redes internas na Internet. Por exemplo, a polícia e outros órgãos de segurança e ordem pública provavelmente têm pelo menos algumas intranets internas que estão isoladas do mundo exterior (o *firewall* mais eficiente possível – a ausência de quaisquer conexões físicas com a Internet). Os *firewalls* também podem ser problemáticos em sistemas distribuídos por impedir o acesso legítimo a serviços, quando é necessário o compartilhamento de recursos entre usuários internos e externos. Assim, os *firewalls* frequentemente devem ser complementados por mecanismos e políticas mais refinados, conforme discutido no Capítulo 11.

A implementação da Internet e dos serviços que ela suporta tem acarretado o desenvolvimento de soluções práticas para muitos problemas dos sistemas distribuídos (incluindo a maior parte daqueles definidos na Seção 1.5). Vamos destacar essas soluções ao longo de todo o livro, apontando sua abrangência e suas limitações quando for apropriado.

1.3.2 Computação móvel e ubíqua

Os avanços tecnológicos na miniaturização de dispositivos e interligação em rede sem fio têm levado cada vez mais à integração de equipamentos de computação pequenos e portáteis com sistemas distribuídos. Esses equipamentos incluem:

- Computadores *notebook*.
- Aparelhos portáteis, incluindo telefones móveis, *smartphones*, equipamentos com GPS, *pagers*, assistentes digitais pessoais (PDAs), câmeras de vídeo e digitais.
- Aparelhos acoplados ao corpo, como relógios de pulso inteligentes com funcionalidade semelhante ao de um PDA.
- Dispositivos incorporados em aparelhos, como máquinas de lavar, aparelhos de som de alta fidelidade, carros, geladeiras, etc.

A portabilidade de muitos desses dispositivos, junto a sua capacidade de se conectar convenientemente com redes em diferentes lugares, tornam a *computação móvel* possível. Computação móvel é a execução de tarefas de computação enquanto o usuário está se deslocando de um lugar a outro ou visitando lugares diferentes de seu ambiente usual. Na computação móvel, os usuários que estão longe de suas intranets "de base" (a intranet do trabalho ou de sua residência) podem acessar recursos por intermédio dos equipamentos que carregam consigo. Eles podem continuar a acessar a Internet, os recursos em sua intranet de base e, cada vez mais, existem condições para que os usuários utilizem recursos como impressoras ou mesmo pontos de venda, que estão convenientemente próximos, enquanto transitam. Esta última possibilidade também é conhecida como *computação com reconhecimento de localização* ou *com reconhecimento de contexto*. A mobilidade introduz vários desafios para os sistemas distribuídos, incluindo a necessidade de lidar com a conectividade variável e mesmo com a desconexão, e a necessidade de manter o funcionamento em face da mobilidade do aparelho (consulte a discussão sobre transparência da mobilidade, na Seção 1.5.7).

A *computação ubíqua*, também denominada *computação pervasiva*, é a utilização de vários dispositivos computacionais pequenos e baratos, que estão presentes nos ambientes físicos dos usuários, incluindo suas casas, escritórios e até na rua. O termo "pervasivo" se destina a sugerir que pequenos equipamentos de computação finalmente se tornarão tão entranhados nos objetos diários que mal serão notados. Isto é, seu comportamento computacional será transparente e intimamente vinculado à sua função física. Por sua vez, o termo "ubíquo" dá a noção de que o acesso a serviços de computação está onipresente, isto é, disponível em qualquer lugar.

A presença de computadores em toda parte só se torna útil quando eles podem se comunicar uns com os outros. Por exemplo, pode ser conveniente para um usuário controlar sua máquina de lavar e seu aparelho de som de alta fidelidade a partir de um único dispositivo universal de controle remoto em sua casa. Da mesma forma, seria cômodo a máquina de lavar enviar uma mensagem quando a lavagem tiver terminado.

A computação ubíqua e a computação móvel se sobrepõem, pois, em princípio, o usuário móvel pode usar computadores que estejam em qualquer lugar. Porém, elas são distintas, de modo geral. A computação ubíqua pode ajudar os usuários enquanto estão em um único local, como em casa ou em um hospital. Do mesmo modo, a computação móvel tem vantagens, mesmo envolvendo apenas computadores e equipamentos convencionais isolados, como *notebooks* e impressoras.

A Figura 1.4 mostra um usuário, visitante em uma organização anfitriã, que pode acessar serviços locais a intranet dessa organização, a Internet ou a sua intranet doméstica.

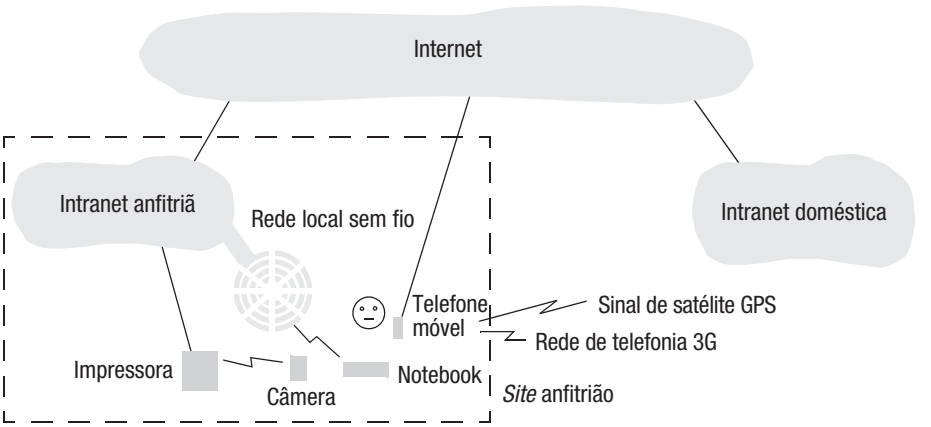

Figura 1.4 Equipamentos portáteis em um sistema distribuído.

O usuário tem acesso a três formas diferentes de conexão sem fio. Seu *notebook* tem uma maneira de se conectar com a rede local sem fio da organização anfitriã. Essa rede fornece cobertura de algumas centenas de metros (um andar de um edifício, digamos) e se conecta ao restante da intranet anfitriã por meio de um *gateway* ou ponto de acesso. O usuário também tem um telefone móvel (celular) que tem acesso à Internet, permitindo acesso à Web e a outros serviços de Internet, restrito apenas pelo que pode ser apresentado em sua pequena tela. O telefone também fornece informações locais por meio da funcionalidade de GPS incorporada. Por fim, o usuário está portando uma câmera digital, que se comunica por intermédio de uma rede sem fio pessoal (com alcance de cerca de 10 m) com outro equipamento, como, por exemplo, uma impressora.

Com uma infraestrutura conveniente, o usuário pode executar algumas tarefas no *site* anfitrião, usando os equipamentos que carrega consigo. Enquanto está no *site* anfitrião, o usuário pode buscar os preços de ações mais recentes a partir de um servidor Web, utilizando seu celular, e também pode usar o GPS incorporado e o *software* de localização de rota para obter instruções para a localização do *site*. Durante uma reunião com seus anfitriões, o usuário pode lhes mostrar uma fotografia, enviando-a a partir da câmera digital, diretamente para uma impressora ou projetor (local) adequadamente ativado na sala de reuniões (descoberto por meio de um serviço de localização). Isso exige apenas a comunicação sem fio entre a câmera e a impressora. Em princípio, eles podem enviar um documento de seus *notebooks* para a mesma impressora ou projetor, utilizando a rede local sem fio e as conexões Ethernet cabeadas com a impressora.

Esse cenário demonstra a necessidade de suportar *operação conjunta espontânea*, por meio da qual associações entre dispositivos são criadas e destruídas rotineiramente, por exemplo, localizando e utilizando os equipamentos da anfitriã (como as impressoras). O principal desafio que se aplica a tais situações é tornar a operação conjunta rápida e conveniente (isto é, espontânea), mesmo que o usuário esteja em um ambiente onde nunca esteve. Isso significa permitir que o equipamento do visitante se comunique com a rede da anfitriã e associá-lo a serviços locais convenientes – um processo chamado de *descoberta de serviço*.

A computação móvel e a computação ubíqua representam áreas de pesquisa ativa, e as várias dimensões anteriores serão discutidas em profundidade no Capítulo 19.

1.3.3 Sistemas multimídia distribuídos

Outra tendência importante é o requisito de suportar serviços multimídia em sistemas distribuídos. Uma definição útil de multimídia é a capacidade de suportar diversos tipos de mídia de maneira integrada. É de se esperar que um sistema distribuído suporte o armazenamento, a transmissão e a apresentação do que frequentemente são referidos como tipos de mídia distintos, como imagens ou mensagens de texto. Um sistema multimídia distribuído deve ser capaz de executar as mesmas funções para tipos de mídia contínuos, como áudio e vídeo, assim como armazenar e localizar arquivos de áudio ou vídeo, transmiti-los pela rede (possivelmente em tempo real, à medida que os fluxos saem de uma câmera de vídeo), suportar a apresentação dos tipos de mídia para o usuário e, opcionalmente, também compartilhar os tipos de mídia por um grupo de usuários.

A característica fundamental dos tipos de mídia contínuos é que eles incluem uma dimensão temporal e, de fato, a integridade do tipo de mídia depende fundamentalmente da preservação das relações em tempo real entre seus elementos. Por exemplo, em uma apresentação de vídeo, é necessário preservar determinado ritmo de transferência em termos de quadros por segundo e, para fluxos em tempo real, determinado atraso ou latência máxima para o envio dos quadros (esse é um exemplo da qualidade do serviço, discutida com mais detalhes na Seção 1.5.8, a seguir).

As vantagens da computação multimídia distribuída são consideráveis, pois uma ampla variedade de novos serviços (multimídia) e aplicativos pode ser fornecida na área de trabalho, incluindo o acesso a transmissões de televisão ao vivo ou gravadas, o acesso a bibliotecas de filmes que oferecem serviços de vídeo sob demanda, o acesso a bibliotecas de músicas, o fornecimento de recursos de áudio e teleconferência e os recursos de telefonia integrados, incluindo telefonia IP ou tecnologias relacionadas, como o Skype – uma alternativa *peer-to-peer* à telefonia IP (a infraestrutura de sistema distribuído que serve de base para o Skype está discutida na Seção 4.5.2). Note que essa tecnologia é revolucionária, desafiando os fabricantes a repensar muitos aparelhos. Por exemplo, qual é o equipamento de entretenimento doméstico básico do futuro – o computador, a televisão ou os console de games?

Webcasting é uma aplicação da tecnologia de multimídia distribuída. *Webcasting* é a capacidade de transmitir mídia contínua (normalmente áudio ou vídeo) pela Internet. Atualmente, é normal eventos esportivos ou musicais importantes serem transmitidos dessa maneira, frequentemente atraindo um grande número de espectadores (por exemplo, o concerto Live8 de 2005 atraiu cerca de 170.000 usuário simultâneos em seu momento de pico).

Sistemas multimídia distribuídos como o *Webcasting* impõem exigências consideráveis na infraestrutura distribuída subjacente, em termos de:

- Dar suporte a uma variedade (extensível) de formatos de codificação e criptografia, como a série MPEG de padrões (incluindo, por exemplo, o popular padrão MP3, conhecido como MPEG-1, Audio Layer 3) e HDTV.
- Fornecer diversos mecanismos para garantir que a qualidade de serviço desejada possa ser obtida.
- Fornecer estratégias de gerenciamento de recursos associadas, incluindo políticas de programação apropriadas para dar suporte à qualidade de serviço desejada.
- Fornecer estratégias de adaptação para lidar com a inevitável situação, em sistemas abertos, em que a qualidade do serviço não pode ser obtida ou mantida.

Mais discussões sobre tais mecanismos podem ser encontradas no Capítulo 20.

1.3.4 Computação distribuída como um serviço público

Com a crescente maturidade da infraestrutura dos sistemas distribuídos, diversas empresas estão promovendo o conceito dos recursos distribuídos como uma *commodity* ou um serviço público, fazendo a analogia entre recursos distribuídos e outros serviços públicos, como água ou eletricidade. Nesse modelo, os recursos são supridos por fornecedores de serviço apropriados e efetivamente alugados, em vez de pertencerem ao usuário final. Esse modelo se aplica tanto a recursos físicos como a serviços lógicos:

- Recursos físicos, como armazenamento e processamento, podem se tornar disponíveis para computadores ligados em rede, eliminando a necessidade de possuírem, eles próprios, esses recursos. Em uma extremidade do espectro, um usuário pode optar por um recurso de armazenamento remoto para requisitos de armazenamento de arquivos (por exemplo, para dados multimídia, como fotografias, música ou vídeo) e/ou para *backups*. Analogamente, essa estratégia permitiria a um usuário alugar um ou mais nós computacionais, para atender a suas necessidades de computação básicas ou para fazer computação distribuída. Na outra extremidade do espectro, os usuários podem acessar sofisticados *data centers* (recursos interligados em rede que dão acesso a repositórios de volumes de dados frequentemente grandes para usuários ou organizações) ou mesmo infraestrutura computacional, usando os serviços agora fornecidos por empresas como Amazon e Google. A virtualização do sistema operacional é uma tecnologia importante para essa estratégia, significando que os usuários podem ser atendidos por serviços em um nó virtual, em vez de físico, oferecendo maior flexibilidade ao fornecedor de serviço em termos de gerenciamento de recursos (a virtualização do sistema operacional é discutida com mais detalhes no Capítulo 7).
- Serviços de *software* (conforme definidos na Seção 1.4) também podem se tornar disponíveis pela Internet global por meio dessa estratégia. De fato, diversas empresas agora oferecem uma ampla variedade deles para aluguel, incluindo serviços como *e-mail* e calendários distribuídos. O Google, por exemplo, empacota diversos serviços empresariais sob o banner Google Apps [www.google.com I]. Esse avanço é possível graças a padrões combinados para serviços de *software*, como, por exemplo, os fornecidos pelos serviços Web (consulte o Capítulo 9).

O termo *computação em nuvem* é usado para capturar essa visão da computação como um serviço público. Uma nuvem é definida 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 prescindam, em grande medida ou totalmente, do *software* local de armazenamento de dados ou de aplicativo (consulte a Figura 1.5). O termo também promove a visão de tudo como um serviço de infraestrutura física ou virtual por meio de *software*, frequentemente pago com base na utilização, em vez de na aquisição. Note que a computação em nuvem reduz os requisitos dos equipamentos dos usuários, permitindo que aparelhos de mesa ou portáteis muito simples acessem uma variedade potencialmente ampla de recursos e serviços.

Geralmente, as nuvens são implementadas em *cluster* de computadores para fornecer a escala e o desempenho necessários exigidos por tais serviços. Um *cluster de computadores* é um conjunto de computadores interligados que cooperam estreitamente para fornecer um único recurso de computação integrado de alto desempenho. Ampliando projetos como o NOW (Network of Workstations) Project da Berkeley [Anderson *et al*. 1995, now.cs.berkeley.edu] e o Beowulf da NASA [www.beowulf.org], a tendência é no sentido de utilizar *hardware* de prateleira tanto para os computadores como para a interligação

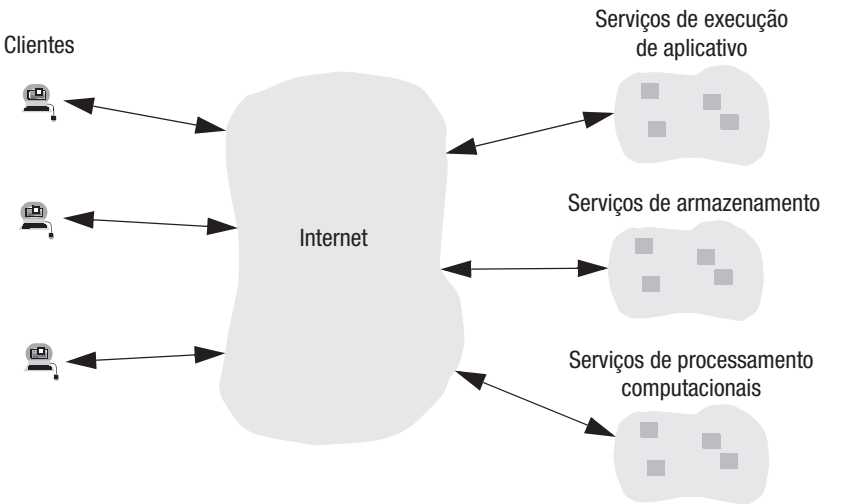

Figura 1.5 Computação em nuvem.

de redes. A maioria dos grupos consiste em PCs convencionais executando uma versão padrão (às vezes reduzida) de um sistema operacional, como o GNU/Linux, interligados por uma rede local. Empresas como HP, Sun e IBM oferecem soluções *blade*. Os *servidores blade* são elementos computacionais mínimos, contendo, por exemplo, recursos de processamento e armazenamento (memória principal). Um sistema *blade* consiste em um número potencialmente grande de servidores blade contidos em gabinetes e armários específicos, também denominados *rack*, mantidos em uma mesma instalação física. Outros elementos, como energia, refrigeração, armazenamento persistente (discos), ligação em rede e telas, são fornecidos pela instalação fisica ou por meio de soluções virtualizadas (discutidas no Capítulo 7). Com essa solução, os servidores *blade* individuais podem ser muito menores e também mais baratos de produzir do que os PCs convencionais.

O objetivo geral dos grupos de computadores é fornecer serviços na nuvem, incluindo recursos de computação de alto desempenho, mas também diversos outros serviços, englobando armazenamento de massa (por exemplo, por intermédio de centros de dados) ou serviços de aplicativo mais elaborados, como pesquisa na Web (o Google, por exemplo, conta com uma arquitetura de grupo de computadores volumosa para implementar seu mecanismo de busca e outros serviços, conforme discutido no Capítulo 21).

A *computação em grade* (discutida no Capítulo 9, Seção 9.7.2) também pode ser vista como uma forma de computação em nuvem. Em grande medida, os termos são sinônimos e, às vezes, mal definidos, mas a computação em grade geralmente pode ser vista como precursora do paradigma mais geral da computação em nuvem, com tendência para o suporte para aplicativos científicos.

1.4 Enfoque no compartilhamento de recursos

Os usuários estão tão acostumados às vantagens do compartilhamento de recursos que podem facilmente ignorar seu significado. Rotineiramente, compartilhamos recursos de *hardware* (como impressoras), recursos de dados (como arquivos) e recursos com funcionalidade mais específica (como os mecanismos de busca).

Do ponto de vista do *hardware*, compartilhamos equipamentos como impressoras e discos para reduzir os custos. Contudo, a maior importância para os usuários é o compartilhamento dos recursos em um nível de abstração mais alto, como informações necessárias às suas aplicações, ao seu trabalho diário e em suas atividades sociais. Por exemplo, os usuários se preocupam em compartilhar informações através de um banco de dados ou de um conjunto de páginas Web – e não com discos ou processadores em que eles estão armazenados. Analogamente, os usuários pensam em termos de recursos compartilhados, como um mecanismo de busca ou um conversor de moeda corrente, sem considerar o servidor (ou servidores) que os fornecem.

Na prática, os padrões de compartilhamento de recursos variam amplamente na abrangência e no quanto os usuários trabalham em conjunto. Em um extremo, temos um mecanismo de busca na Web que fornece um recurso para usuários de todo o mundo, usuários estes que nunca entram em contato diretamente. No outro extremo, no *trabalho cooperativo apoiado por computador*, um grupo de usuários colabora diretamente compartilhando recursos, como documentos, em um pequeno grupo fechado. O padrão de compartilhamento e a distribuição geográfica dos usuários determinam quais mecanismos o sistema deve fornecer para coordenar as ações dos usuários.

O termo *serviço* é usado para designar uma parte distinta de um sistema computacional que gerencia um conjunto de recursos relacionados e apresenta sua funcionalidade para usuários e aplicativos. Por exemplo, acessamos arquivos compartilhados por intermédio de um serviço de sistema de arquivos; enviamos documentos para impressoras por meio de um serviço de impressão; adquirimos bens por meio de um serviço de pagamento eletrônico. O único acesso que temos ao serviço é por intermédio do conjunto de operações que ele exporta. Por exemplo, um serviço de sistema de arquivos fornece operações de *leitura*, *escrita* e *exclusão* dos arquivos.

Em parte, o fato de os serviços restringirem o acesso ao recurso a um conjunto de operações bem definidas é uma prática padrão na engenharia de *software*, mas isso também reflete a organização física dos sistemas distribuídos. Em um sistema distribuído, os recursos são fisicamente encapsulados dentro dos computadores e só podem ser acessados a partir de outros computadores por intermédio de mecanismos de comunicação. Para se obter um compartilhamento eficiente, cada recurso deve ser gerenciado por um programa que ofereça uma interface de comunicação, permitindo ao recurso ser acessado e atualizado de forma confiável e consistente.

O termo *servidor* provavelmente é conhecido da maioria dos leitores. Ele se refere a um programa em execução (um *processo*) em um computador interligado em rede, que aceita pedidos de programas em execução em outros computadores para efetuar um serviço e responder apropriadamente. Os processos que realizam os pedidos são referidos como *clientes* e a estratégia geral é conhecida como *computação cliente-servidor*. Nessa estratégia, os pedidos são enviados em mensagens dos clientes para um servidor, e as respostas são enviadas do servidor para os clientes. Quando o cliente envia um pedido para que uma operação seja efetuada, dizemos que o cliente *requisita uma operação* no servidor. Uma interação completa entre um cliente e um servidor, desde quando o cliente envia seu pedido até o momento em que recebe a resposta do servidor, é chamada de *requisição remota*.

O mesmo processo pode ser tanto cliente como servidor, pois, às vezes, os servidores solicitam operações em outros servidores. Os termos cliente e servidor só se aplicam às funções desempenhadas em um único pedido. Os clientes são ativos (fazendo pedidos) e os servidores são passivos (sendo ativados somente ao receberem pedidos); os servidores funcionam continuamente, enquanto os clientes duram apenas enquanto os aplicativos dos quais fazem parte estiverem ativos.

Note que, por padrão, os termos *cliente* e *servidor* se referem a *processos* e não aos computadores em que são executados, embora no jargão comum esses termos também se refiram aos computadores em si. Outra distinção, que discutiremos no Capítulo 5, é que, em um sistema distribuído escrito em uma linguagem orientada a objetos, os recursos podem ser encapsulados como objetos e acessados por objetos clientes; nesse caso, falamos em um *objeto cliente* invocando um método em um *objeto servidor*.

Muitos sistemas distribuídos (mas certamente não todos) podem ser totalmente construídos na forma de clientes e servidores. A World Wide Web, o *e-mail* e as impressoras interligadas em rede são exemplos que se encaixam nesse modelo. Discutiremos alternativas para os sistemas cliente-servidor no Capítulo 2.

Um navegador Web em execução é um exemplo de cliente. O navegador Web se comunica com um servidor Web para solicitar páginas Web. Examinaremos a Web com mais detalhes na Seção 1.6.

1.5 Desafios

Os exemplos da Seção 1.2 ilustram a abrangência dos sistemas distribuídos e os problemas que surgem em seu projeto. Em muitos deles, desafios significativos foram encontrados e superados. Como a abrangência e a escala dos sistemas distribuídos e dos aplicativos são maiores, é provável que eles apresentem os mesmos desafios e ainda outros. Nesta seção, descrevemos os principais desafios dos sistemas distribuídos.

1.5.1 Heterogeneidade

A Internet permite aos usuários acessarem serviços e executarem aplicativos por meio de um conjunto heterogêneo de computadores e redes. A heterogeneidade (isto é, variedade e diferença) se aplica aos seguintes aspectos:

- redes;
- *hardware* de computador;
- sistemas operacionais;
- linguagens de programação;
- implementações de diferentes desenvolvedores.

Embora a Internet seja composta de muitos tipos de redes (como ilustrado na Figura 1.3), suas diferenças são mascaradas pelo fato de que todos os computadores ligados a elas utilizam protocolos Internet para se comunicar. Por exemplo, um computador que possui uma placa Ethernet tem uma implementação dos protocolos Internet, enquanto um computador em um tipo diferente de rede tem uma implementação dos protocolos Internet para essa rede. O Capítulo 3 explica como os protocolos Internet são implementados em uma variedade de redes diferentes.

Os tipos de dados, como os inteiros, podem ser representados de diversas maneiras em diferentes tipos de *hardware*; por exemplo, existem duas alternativas para a ordem em que os bytes de valores inteiros são armazenados: uma iniciando a partir do byte mais significativo e outra, a partir do byte menos significativo. Essas diferenças na representação devem ser consideradas, caso mensagens devam ser trocadas entre programas sendo executados em diferentes *hardware*s.

Embora os sistemas operacionais de todos os computadores na Internet precisem incluir uma implementação dos protocolos Internet, nem todos fornecem, necessariamente,

a mesma interface de programação de aplicativos para esses protocolos. Por exemplo, as chamadas para troca de mensagens no UNIX são diferentes das chamadas no Windows.

Diferentes linguagens de programação usam diferentes representações para caracteres e estruturas de dados, como vetores e registros. Essas diferenças devem ser consideradas, caso programas escritos em diferentes linguagens precisem se comunicar.

Os programas escritos por diferentes desenvolvedores não podem se comunicar, a menos que utilizem padrões comuns; por exemplo, para realizar a comunicação via rede e usar uma mesma representação de tipos de dados primitivos e estruturas de dados nas mensagens. Para que isso aconteça, padrões precisam ser estabelecidos e adotados. Assim é o caso dos protocolos Internet.

Middleware • O termo *middleware* se aplica a uma camada de *software* que fornece uma abstração de programação, assim como o mascaramento da heterogeneidade das redes, do *hardware*, dos sistemas operacionais e das linguagens de programação subjacentes. O CORBA (Common Object Request Broker), que será descrito nos Capítulos 4, 5 e 8, é um exemplo. Alguns *middlewares*, como o Java RMI (Remote Method Invocation) (veja o Capítulo 5), suportam apenas uma linguagem de programação. A maioria é implementada sobre os protocolos Internet, os quais escondem a diferença existente entre redes subjacentes. Todo *middleware*, em si, trata das diferenças em nível dos sistemas operacionais e do *hardware* – o modo como isso é feito será o tópico principal do Capítulo 4.

Além de resolver os problemas de heterogeneidade, o *middleware* fornece um modelo computacional uniforme para ser usado pelos programadores de serviços e de aplicativos distribuídos. Os modelos possíveis incluem a invocação remota de objetos, a notificação remota de eventos, o acesso remoto a banco de dados e o processamento de transação distribuído. Por exemplo, o CORBA fornece invocação remota de objetos, a qual permite que um objeto, em um programa sendo executado em um computador, invoque um método de um objeto em um programa executado em outro computador. Sua implementação oculta o fato de que as mensagens passam por uma rede para enviar o pedido de invocação e sua resposta.

Heterogeneidade e migração de código • O termo *migração de código*, ou ainda, *código móvel,* é usado para se referir ao código de programa que pode ser transferido de um computador para outro e ser executado no destino – os *applets* Java são um exemplo. Um código destinado à execução em um computador não é necessariamente adequado para outro computador, pois, normalmente, os programas executáveis são específicos a um conjunto de instruções e a um sistema operacional.

A estratégia de *máquina virtual* oferece uma maneira de tornar um código executável em uma variedade de computadores hospedeiros: o compilador de uma linguagem em particular gera código para uma máquina virtual, em vez de código para um processador e sistema operacional específicos. Por exemplo, o compilador Java produz código para uma máquina virtual Java (JVM, Java Virtual Machine), a qual o executa por meio de interpretação. A máquina virtual Java precisa ser implementada uma vez para cada tipo de computador a fim de que os programas Java sejam executados.

Atualmente, a forma mais usada de código móvel é a inclusão de programas Java*script* em algumas páginas Web carregadas nos navegadores clientes. Essa extensão da tecnologia da Web será discutida com mais detalhes na Seção 1.6, a seguir.

1.5.2 Sistemas abertos

Diz-se que um sistema computacional é aberto quando ele pode ser estendido e reimplementado de várias maneiras. O fato de um sistema distribuído ser ou não um sistema

aberto é determinado principalmente pelo grau com que novos serviços de compartilhamento de recursos podem ser adicionados e disponibilizados para uso por uma variedade de programas clientes.

A característica de sistema aberto é obtida a partir do momento em que a especificação e a documentação das principais interfaces de *software* dos componentes de um sistema estão disponíveis para os desenvolvedores de *software*. Em uma palavra, as principais interfaces são *publicadas*. Esse processo é similar àquele realizado por organizações de padronização, porém, frequentemente, ignora os procedimentos oficiais, os quais normalmente são pesados e lentos.

Entretanto, a publicação de interfaces é apenas o ponto de partida para adicionar e estender serviços em um sistema distribuído. O maior desafio para os projetistas é encarar a complexidade de sistemas distribuídos compostos por muitos componentes e elaborados por diferentes pessoas.

Os projetistas dos protocolos Internet elaboraram uma série de documentos, chamados *Requests For Comments,* ou RFCs, cada um identificado por um número. No início dos anos 80, as especificações dos protocolos de comunicação Internet foram publicadas nessa série, acompanhadas de especificações de aplicativos executados com eles, como transferência de arquivos, *e-mail* e telnet. Essa prática continua e forma a base da documentação técnica da Internet. Essa série inclui discussões, assim como as especificações dos protocolos (pode-se obter cópias das RFCs no endereço [www.ietf.org]). Dessa forma, com a publicação dos protocolos de comunicação Internet, permitiu-se a construção de uma variedade de sistemas e aplicativos para a Internet, incluindo a Web. As RFCs não são os únicos meios de publicação. Por exemplo, o W3C (World Wide Web Consortium) desenvolve e publica padrões relacionados ao funcionamento da Web (veja [www.w3.org]).

Os sistemas projetados a partir de padrões públicos são chamados de *sistemas distribuídos abertos*, para reforçar o fato de que eles são extensíveis. Eles podem ser ampliados em nível de *hardware*, pela adição de computadores em uma rede, e em nível de *software*, pela introdução de novos serviços ou pela reimplementação dos antigos, permitindo aos programas aplicativos compartilharem recursos. Uma vantagem adicional, frequentemente mencionada para sistemas abertos, é sua independência de fornecedores individuais.

Resumindo:

- Os sistemas abertos são caracterizados pelo fato de suas principais interfaces serem publicadas.
- Os sistemas distribuídos abertos são baseados na estipulação de um mecanismo de comunicação uniforme e em interfaces publicadas para acesso aos recursos compartilhados.
- Os sistemas distribuídos abertos podem ser construídos a partir de *hardware* e *software* heterogêneo, possivelmente de diferentes fornecedores. Para que um sistema funcione corretamente, a compatibilidade de cada componente com o padrão publicado deve ser cuidadosamente testada e verificada.

1.5.3 Segurança

Muitos recursos de informação que se tornam disponíveis e são mantidos em sistemas distribuídos têm um alto valor intrínseco para seus usuários. Portanto, sua segurança é de considerável importância. A segurança de recursos de informação tem três componentes: confidencialidade (proteção contra exposição para pessoas não autorizadas), integridade (proteção contra alteração ou dano) e disponibilidade (proteção contra interferência com os meios de acesso aos recursos).

A Seção 1.1 mostrou que, embora a Internet permita que um programa em um computador se comunique com um programa em outro computador, independentemente de sua localização, existem riscos de segurança associados ao livre acesso a todos os recursos em uma intranet. Embora um *firewall* possa ser usado para formar uma barreira em torno de uma intranet, restringindo o tráfego que pode entrar ou sair, isso não garante o uso apropriado dos recursos pelos usuários de dentro da intranet, nem o uso apropriado dos recursos na Internet, que não são protegidos por *firewalls*.

Em um sistema distribuído, os clientes enviam pedidos para acessar dados gerenciados por servidores, o que envolve o envio de informações em mensagens por uma rede. Por exemplo:

1. Um médico poderia solicitar acesso aos dados dos pacientes de um hospital ou enviar mais informações sobre esses pacientes.
2. No comércio eletrônico e nos serviços bancários, os usuários enviam seus números de cartão de crédito pela Internet.

Nesses dois exemplos, o desafio é enviar informações sigilosas em uma ou mais mensagens, por uma rede, de maneira segura. No entanto, a segurança não é apenas uma questão de ocultar o conteúdo das mensagens – ela também envolve saber com certeza a identidade do usuário, ou outro agente, em nome de quem uma mensagem foi enviada. No primeiro exemplo, o servidor precisa saber se o usuário é realmente um médico e, no segundo exemplo, o usuário precisa ter certeza da identidade da loja ou do banco com o qual está tratando. O segundo desafio, neste caso, é identificar corretamente um usuário ou outro agente remoto. Esses dois desafios podem ser resolvidos com o uso de técnicas de criptografia desenvolvidas para esse propósito. Elas são amplamente utilizadas na Internet e serão discutidas no Capítulo 11.

Entretanto, dois desafios de segurança, descritos a seguir, ainda não foram totalmente resolvidos:

Ataque de negação de serviço (Denial of Service): ocorre quando um usuário interrompe um serviço por algum motivo. Isso pode ser conseguido bombardeando o serviço com um número tão grande de pedidos sem sentido, que os usuários sérios não são capazes de utilizá-lo. Isso é chamado de *ataque de negação de serviço*. De tempos em tempos, surgem ataques de negação de serviços contra alguns serviços e servidores Web bastante conhecidos. Atualmente, tais ataques são controlados buscando-se capturar e punir os responsáveis após o evento, mas essa não é uma solução geral para o problema. Medidas para se opor a isso, baseadas em melhorias no gerenciamento das redes, estão sendo desenvolvidas e serão assunto do Capítulo 3.

Segurança de código móvel: um código móvel precisa ser manipulado com cuidado. Considere alguém que receba um programa executável como um anexo de correio eletrônico: os possíveis efeitos da execução do programa são imprevisíveis; por exemplo, pode parecer que ele está apenas exibindo uma figura interessante, mas, na realidade, está acessando recursos locais ou, talvez, fazendo parte de um ataque de negação de serviço. Algumas medidas para tornar seguro um código móvel serão esboçadas no Capítulo 11.

1.5.4 Escalabilidade

Os sistemas distribuídos funcionam de forma efetiva e eficaz em muitas escalas diferentes, variando desde uma pequena intranet até a Internet. Um sistema é descrito como *escalável* se permanece eficiente quando há um aumento significativo no número de re-

cursos e no número de usuários. O número de computadores e de serviços na Internet aumentou substancialmente. A Figura 1.6 mostra o aumento no número de computadores e servidores Web durante os 12 anos de história da Web até 2005 (veja [zakon.org]). É interessante notar o aumento significativo tanto no número de computadores como no serviços Web nesse período, mas também a porcentagem relativa, que cresce rapidamente, uma tendência explicada pelo crescimento da computação pessoal fixa e móvel. Um único servidor Web também pode, cada vez mais, ser hospedado em vários computadores.

O projeto de sistemas distribuídos escaláveis apresenta os seguintes desafios:

Controlar o custo dos recursos físicos: à medida que a demanda por um recurso aumenta, deve ser possível, a um custo razoável, ampliar o sistema para atendê-la. Por exemplo, a frequência com que os arquivos são acessados em uma intranet provavelmente vai crescer à medida que o número de usuários e de computadores aumentar. Deve ser possível adicionar servidores de arquivos de forma a evitar o gargalo de desempenho que haveria caso um único servidor de arquivos tivesse de tratar todos os pedidos de acesso a arquivos. Em geral, para que um sistema com n usuários seja escalável, a quantidade de recursos físicos exigida para suportá-los deve ser, no máximo, $O(n)$ – isto é, proporcional a n. Por exemplo, se um único servidor de arquivos pode suportar 20 usuários, então dois servidores deverão suportar 40 usuários. Embora isso pareça um objetivo óbvio, não é necessariamente fácil atingi-lo, como mostraremos no Capítulo 12.

Controlar a perda de desempenho: considere o gerenciamento de um conjunto de dados, cujo tamanho é proporcional ao número de usuários ou recursos presentes no sistema; por exemplo, a tabela de correspondência entre os nomes de domínio dos computadores e seus endereços IP mantidos pelo Domain Name System (DNS), que é usado principalmente para pesquisar nomes, como www.amazon.com. Os algoritmos que utilizam estruturas hierárquicas têm melhor escalabilidade do que os que usam estruturas lineares. Porém, mesmo com as estruturas hierárquicas, um aumento no tamanho resulta em alguma perda de desempenho: o tempo que leva para acessar dados hierarquicamente estruturados é $O(\log n)$, onde n é o tamanho do conjunto de dados. Para que um sistema seja escalável, a perda de desempenho máxima não deve ser maior do que isso.

Impedir que os recursos de software se esgotem: um exemplo de falta de escalabilidade é mostrado pelos números usados como endereços IP (endereços de computador na Internet). No final dos anos 70, decidiu-se usar 32 bits para esse propósito,

Data	*Computadores*	*Servidores Web*	*Percentual*
Julho de 1993	1.776.000	130	0,008
Julho de 1995	6.642.000	23.500	0,4
Julho de 1997	19.540.000	1.203.096	6
Julho de 1999	56.218.000	6.598.697	12
Julho de 2001	125.888.197	31.299.592	25
Julho de 2003	~200.000.000	42.298.371	21
Julho de 2005	353.284.187	67.571.581	19

Figura 1.6 Crescimento da Internet (computadores e servidores Web).

mas, conforme será explicado no Capítulo 3, a quantidade de endereços IP disponíveis está se esgotando. Por isso, uma nova versão do protocolo, com endereços IP em 128 bits, está sendo adotada e isso exige modificações em muitos componentes de *software*. Para sermos justos com os primeiros projetistas da Internet, não há uma solução correta para esse problema. É difícil prever, com anos de antecedência, a demanda que será imposta sobre um sistema. Além disso, superestimar o crescimento futuro pode ser pior do que se adaptar para uma mudança quando formos obrigados a isso – por exemplo, endereços IP maiores ocupam espaço extra no armazenamento de mensagens e no computador.

Evitar gargalos de desempenho: em geral, os algoritmos devem ser descentralizados para evitar a existência de gargalos de desempenho. Ilustramos esse ponto com referência ao predecessor do Domain Name System, no qual a tabela de correspondência entre endereços IP e nomes era mantida em um único arquivo central, cujo *download* podia ser feito em qualquer computador que precisasse dela. Isso funcionava bem quando havia apenas algumas centenas de computadores na Internet, mas logo se tornou um sério gargalo de desempenho e de administração. Com milhares de computadores na Internet, imagine o tamanho e o acesso a esse arquivo central. O Domain Name System eliminou esse gargalo, particionando a tabela de correspondência de nomes entre diversos servidores localizados em toda a Internet e administrados de forma local – veja os Capítulos 3 e 13. Alguns recursos compartilhados são acessados com muita frequência; por exemplo, muitos usuários podem acessar uma mesma página Web, causando uma queda no desempenho. No Capítulo 2, veremos que o uso de cache e de replicação melhora o desempenho de recursos que são pesadamente utilizados.

De preferência, o *software* de sistema e de aplicativo não deve mudar quando a escala do sistema aumentar, mas isso é difícil de conseguir. O problema da escala é um tema central no desenvolvimento de sistemas distribuídos. As técnicas que têm obtido sucesso serão discutidas extensivamente neste livro. Elas incluem o uso de dados replicados (Capítulo 18), a técnica associada de uso de cache (Capítulos 2 e 12) e a distribuição de vários servidores para manipular as tarefas comumente executadas, permitindo que várias tarefas semelhantes sejam executadas concorrentemente.

1.5.5 Tratamento de falhas

Às vezes, os sistemas de computador falham. Quando ocorrem falhas no *hardware* ou no *software*, os programas podem produzir resultados incorretos ou podem parar antes de terem concluído a computação pretendida. No Capítulo 2, discutiremos e classificaremos uma variedade de tipos de falhas possíveis, que podem ocorrer nos processos e nas redes que compõem um sistema distribuído.

As falhas em um sistema distribuído são parciais – isto é, alguns componentes falham, enquanto outros continuam funcionando. Portanto, o tratamento de falhas é particularmente difícil. As técnicas para tratamento de falhas a seguir serão discutidas ao longo de todo o livro:

Detecção de falhas: algumas falhas podem ser detectadas. Por exemplo, somas de verificação podem ser usadas para detectar dados corrompidos em uma mensagem ou em um arquivo. O Capítulo 2 mostra que é difícil, ou mesmo impossível, detectar algumas outras falhas, como um servidor remoto danificado na Internet. O desafio é gerenciar a ocorrência de falhas que não podem ser detectadas, mas que podem ser suspeitas.

Mascaramento de falhas: algumas falhas detectadas podem ser ocultas ou se tornar menos sérias. Dois exemplos de ocultação de falhas:

1. Mensagens podem ser retransmitidas quando não chegam.
2. Dados de arquivos podem ser gravados em dois discos, para que, se um estiver danificado, o outro ainda possa estar correto.

Simplesmente eliminar uma mensagem corrompida é um exemplo de como tornar uma falha menos grave – ela pode ser retransmitida. O leitor provavelmente perceberá que as técnicas descritas para o mascaramento de falhas podem não funcionar nos piores casos; por exemplo, os dados no segundo disco também podem estar danificados ou a mensagem pode não chegar em um tempo razoável e, contudo, ser retransmitida frequentemente.

Tolerância a falhas: a maioria dos serviços na Internet apresenta falhas – não seria prático para eles tentar detectar e mascarar tudo que possa ocorrer em uma rede grande assim, com tantos componentes. Seus clientes podem ser projetados de forma a tolerar falhas, o que geralmente envolve a tolerância também por parte dos usuários. Por exemplo, quando um navegador não consegue contatar um servidor Web, ele não faz o usuário esperar indefinidamente, enquanto continua tentando – ele informa o usuário sobre o problema, deixando-o livre para tentar novamente. Os serviços que toleram falhas serão discutidos no item sobre redundância, logo a seguir.

Recuperação de falhas: a recuperação envolve projetar *software* de modo que o estado dos dados permanentes possa ser recuperado ou "retrocedido" após a falha de um servidor. Em geral, as computações realizadas por alguns programas ficarão incompletas quando ocorrer uma falha, e os dados permanentes que eles atualizam (arquivos em disco) podem não estar em um estado consistente. A recuperação de falhas será estudada no Capítulo 17.

Redundância: os serviços podem se tornar tolerantes a falhas com o uso de componentes redundantes. Considere os exemplos a seguir:

1. Sempre deve haver pelo menos duas rotas diferentes entre dois roteadores quaisquer na Internet.
2. No Domain Name System, toda tabela de correspondência de nomes é replicada em pelo menos dois servidores diferentes.
3. Um banco de dados pode ser replicado em vários servidores, para garantir que os dados permaneçam acessíveis após a falha de qualquer servidor. Os servidores podem ser projetados de forma a detectar falhas em seus pares; quando uma falha é detectada em um servidor, os clientes são redirecionados para os servidores restantes.

O projeto de técnicas eficazes para manter réplicas atualizadas de dados que mudam rapidamente, sem perda de desempenho excessiva, é um desafio. Várias estratégias para isso serão discutidas no Capítulo 18.

Os sistemas distribuídos fornecem um alto grau de disponibilidade perante falhas de *hardware*. A *disponibilidade* de um sistema é a medida da proporção de tempo em que ele está pronto para uso. Quando um dos componentes de um sistema distribuído falha, apenas o trabalho que estava usando o componente defeituoso é afetado. Um usuário pode passar para outro computador, caso aquele que estava sendo utilizado falhe; um processo servidor pode ser iniciado em outro computador.

1.5.6 Concorrência

Tanto os serviços como os aplicativos fornecem recursos que podem ser compartilhados pelos clientes em um sistema distribuído. Portanto, existe a possibilidade de que vários clientes tentem acessar um recurso compartilhado ao mesmo tempo. Por exemplo, uma estrutura de dados que registra lances de um leilão pode ser acessada com muita frequência, quando o prazo final se aproximar.

O processo que gerencia um recurso compartilhado poderia aceitar e tratar um pedido de cliente por vez. Contudo, essa estratégia limita o desempenho do tratamento de pedidos. Portanto, os serviços e aplicativos geralmente permitem que vários pedidos de cliente sejam processados concorrentemente. Para tornar isso mais concreto, suponha que cada recurso seja encapsulado como um objeto e que as chamadas sejam executadas em diferentes fluxos de execução, processos ou *threads*, concorrentes. Nesta situação, é possível que vários fluxos de execução estejam simultaneamente dentro de um objeto e, eventualmente, suas operações podem entrar em conflito e produzir resultados inconsistentes. Por exemplo, se dois lances em um leilão forem Smith: $122 e Jones: $111 e as operações correspondentes fossem entrelaçadas sem nenhum controle, eles poderiam ser armazenados como Smith: $111 e Jones: $122.

A moral da história é que qualquer objeto que represente um recurso compartilhado em um sistema distribuído deve ser responsável por garantir que ele opere corretamente em um ambiente concorrente. Isso se aplica não apenas aos servidores, mas também aos objetos nos aplicativos. Portanto, qualquer programador que implemente um objeto que não foi destinado para uso em um sistema distribuído deve fazer o que for necessário para garantir que, em um ambiente concorrente, ele não assuma resultados inconsistentes.

Para que um objeto mantenha coerência em um ambiente concorrente, suas operações devem ser sincronizadas de tal maneira que seus dados permaneçam consistentes. Isso pode ser obtido por meio de técnicas padrão, como semáforos, que são disponíveis na maioria dos sistemas operacionais. Esse assunto, e sua extensão para coleções de objetos compartilhados distribuídos, serão discutidos nos Capítulos 7 e 17.

1.5.7 Transparência

A transparência é definida como a ocultação, para um usuário final ou para um programador de aplicativos, da separação dos componentes em um sistema distribuído, de modo que o sistema seja percebido como um todo, em vez de como uma coleção de componentes independentes. As implicações da transparência têm grande influência sobre o projeto do *software* de sistema.

O ANSA *Reference Manual* [ANSA 1989] e o RM-ODP (Reference Model for Open Distributed Processing) da *International Organization for Standardization* [ISO 1992] identificam oito formas de transparência. Parafraseamos as definições originais da ANSA, substituindo transparência de migração por transparência de mobilidade, cujo escopo é mais abrangente:

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

Transparência de localização permite que os recursos sejam acessados sem conhecimento de sua localização física ou em rede (por exemplo, que prédio ou endereço IP).

Transparência de concorrência permite que vários processos operem concorrentemente, usando recursos compartilhados sem interferência entre eles.

Transparência de replicação permite que várias instâncias dos recursos sejam usadas para aumentar a confiabilidade e o desempenho, sem conhecimento das réplicas por parte dos usuários ou dos programadores de aplicativos.

Transparência de falhas permite a ocultação de falhas, possibilitando que usuários e programas aplicativos concluam suas tarefas, a despeito da falha de componentes de *hardware* ou *software*.

Transparência de mobilidade permite a movimentação de recursos e clientes dentro de um sistema, sem afetar a operação de usuários ou de programas.

Transparência de desempenho permite que o sistema seja reconfigurado para melhorar o desempenho à medida que as cargas variam.

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.

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*.

Como exemplo da transparência de acesso, considere o uso de uma interface gráfica em um sistema de arquivo que organiza diretórios e subdiretórios em pastas; o que o usuário enxerga é a mesma coisa, independentemente de os arquivos serem contidos em uma pasta local ou remota. Outro exemplo é uma interface de programação para arquivos que usa as mesmas operações para acessar tanto arquivos locais como remotos (veja o Capítulo 12). Como exemplo de falta de transparência de acesso, considere um sistema distribuído que não permite acessar arquivos em um computador remoto, a não ser que você utilize o programa FTP para isso.

Os nomes de recurso na Web, isto é, os URLs, são transparentes à localização, pois a parte do URL que identifica o nome de um servidor Web se refere a um nome de computador em um domínio, em vez de seu endereço IP. Entretanto, os URLs não são transparentes à mobilidade, pois se a página Web de alguém mudar para o seu novo local de trabalho, em um domínio diferente, todos as referências (*links*) nas outras páginas ainda apontarão para a página original.

Em geral, identificadores como os URLs, que incluem os nomes de domínio dos computadores, impedem a transparência de replicação. Embora o DNS permita que um nome de domínio se refira a vários computadores, ele escolhe apenas um deles ao pesquisar um nome. Como um esquema de replicação geralmente precisa acessar todos os computadores participantes, ele precisará acessar cada uma das entradas de DNS pelo nome.

Para ilustrar a transparência de rede, considere o uso de um endereço de correio eletrônico, como *Fred.Flintstone@stoneit.com*. O endereço consiste em um nome de usuário e um nome de domínio. O envio de correspondência para tal usuário não envolve o conhecimento de sua localização física ou na rede, nem o procedimento de envio de uma mensagem de correio depende da localização do destinatário. Assim, o correio eletrônico, dentro da Internet, oferece tanto transparência de localização como de acesso (isto é, transparência de rede).

A transparência de falhas também pode ser vista no contexto do correio eletrônico, que finalmente é entregue, mesmo quando os servidores ou os enlaces de comunicação falham. As falhas são mascaradas pela tentativa de retransmitir as mensagens, até que elas sejam enviadas com êxito, mesmo que isso demore vários dias. Geralmente, o *middleware* converte as falhas de redes e os processos em exceções em nível de programação (veja uma explicação no Capítulo 5).

Para ilustrar a transparência de mobilidade, considere o caso dos telefones móveis. Suponha que quem faz a ligação e quem recebe a ligação estejam viajando de trem em diferentes partes de um país. Consideramos o telefone de quem chama como cliente e o telefone de quem recebe, como recurso. Os dois usuários de telefone que compõem a ligação não estão cientes da mobilidade dos telefones (o cliente e o recurso).

O uso dos diferentes tipos de transparência oculta e transforma em anônimos os recursos que não têm relevância direta para a execução de uma tarefa por parte de usuários e de programadores de aplicativos. Por exemplo, geralmente é desejável que recursos de *hardware* semelhantes sejam alocados de maneira permutável para executar uma tarefa – a identidade do processador usado para executar um processo geralmente fica oculta do usuário e permanece anônima. Conforme mencionado na Seção 1.3.2, nem sempre isso pode ser atingido. Por exemplo, um viajante que liga um *notebook* na rede local de cada escritório visitado faz uso de serviços locais, como o envio de correio eletrônico, usando diferentes servidores em cada local. Mesmo dentro de um prédio, é normal enviar um documento para que ele seja impresso em uma impressora específica configurada no sistema, normalmente a que está mais próxima do usuário.

1.5.8 Qualidade de serviço

Uma vez fornecida a funcionalidade exigida de um serviço (como o serviço de arquivo em um sistema distribuído), podemos passar a questionar a qualidade do serviço fornecido. As principais propriedades não funcionais dos sistemas que afetam a qualidade do serviço experimentada pelos clientes e usuários são a *confiabilidade*, a *segurança* e o *desempenho*. 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.

Os problemas de confiabilidade e de segurança são fundamentais no projeto da maioria dos sistemas de computador. 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, conforme discutido nos parágrafos a seguir.

Alguns aplicativos, incluindo os multimídia, manipulam *dados críticos quanto ao tempo* – fluxos de dados que precisam ser processados ou transferidos de um processo para outro em velocidade constante. Por exemplo, um serviço de filmes poderia consistir em um programa cliente que estivesse recuperando um filme de um servidor de vídeo e o apresentando na tela do usuário. Para se obter um resultado satisfatório, os sucessivos quadros de vídeo precisam ser exibidos para o usuário dentro de alguns limites de tempo especificados.

Na verdade, a abreviação QoS (Quality of Service) foi imprópria para se referir à capacidade dos sistemas de satisfazer tais prazos finais. Seu sucesso depende da disponibilidade dos recursos de computação e de rede necessários nos momentos apropriados. Isso implica um requisito para o sistema de fornecer recursos de computação e comunicação garantidos, suficientes para permitir que os aplicativos terminem cada tarefa a tempo (por exemplo, a tarefa de exibir um quadro de vídeo).

As redes normalmente usadas hoje têm alto desempenho; por exemplo, o iPlayer, da BBC, geralmente tem desempenho aceitável, mas quando as redes estão muito carregadas, seu desempenho pode se deteriorar – e não é dada nenhuma garantia. A qualidade de serviço (QoS) se aplica tanto aos sistemas operacionais quanto às redes. Cada recurso fundamental deve ser reservado pelos aplicativos que exigem QoS e deve existir gerenciadores de recursos que deem garantias. Os pedidos de reserva que não podem ser atendidos são rejeitados. Esses problemas serão tratados com mais profundidade no Capítulo 20.

1.6 Estudo de caso: a World Wide Web

A World Wide Web [www.w3.org I, Berners-Lee 1991] é um sistema em evolução para a publicação e para o acesso a recursos e serviços pela Internet. Por meio de navegadores Web (*browsers*) comumente disponíveis, os usuários recuperam e veem documentos de muitos tipos, ouvem fluxos de áudio, assistem a fluxos de vídeo e interagem com um vasto conjunto de serviços.

A Web nasceu no centro europeu de pesquisa nuclear (CERN), na Suíça, em 1989, como um veículo para troca de documentos entre uma comunidade de físicos conectados pela Internet [Berners-Lee 1999]. Uma característica importante da Web é que ela fornece uma estrutura de *hipertexto* entre os documentos que armazena, refletindo a necessidade dos usuários de organizar seus conhecimentos. Isso significa que os documentos contêm *links* (ou *hyperlinks*) – referências para outros documentos e recursos que também estão armazenados na Web.

É fundamental para um usuário da Web que, ao encontrar determinada imagem ou texto dentro de um documento, isso seja frequentemente acompanhado de *links* para documentos e outros recursos relacionados. A estrutura de *links* pode ser arbitrariamente complexa, e o conjunto de recursos que podem ser adicionados é ilimitado – a "teia" (Web) de *links* é realmente mundial. Bush [1945] imaginou estruturas de hipertexto há mais de 50 anos; foi com o desenvolvimento da Internet que essa ideia pôde se manifestar em escala mundial.

A Web é um sistema *aberto*: ela pode ser ampliada e implementada de novas maneiras, sem perturbar a funcionalidade já existente (consulte a Seção 1.5.2). A operação da Web é baseada em padrões de comunicação e de documentos livremente publicados. Por exemplo, existem muitos tipos de navegadores, em muitos casos, implementados em várias plataformas; e existem muitas implementações de servidores Web. Qualquer navegador pode recuperar recursos de qualquer servidor, desde que ambos utilizem os mesmos padrões em suas implementações. Os usuários têm acesso a navegadores na maioria dos equipamentos que utilizam, desde telefones móveis até computadores de mesa.

A Web é aberta no que diz respeito aos tipos de recursos que nela podem ser publicados e compartilhados. Em sua forma mais simples, um recurso da Web é uma página ou algum outro tipo de *conteúdo* que possa ser armazenado em um arquivo e apresentado ao usuário, como arquivos de programa, arquivos de mídia e documentos em PostScript ou Portable Document Format (pdf). Se alguém inventar, digamos, um novo formato de armazenamento de imagem, então as imagens que tenham esse formato poderão ser publicadas na Web imediatamente. Os usuários necessitam de meios para ver imagens nesse novo formato, mas os navegadores são projetados de maneira a acomodar nova funcionalidade de apresentação de conteúdo, na forma de aplicativos "auxiliares" e "*plugins*".

A Web já foi além desses recursos de dados simples e hoje abrange serviços como a aquisição eletrônica de bens. Ela tem evoluído sem mudar sua arquitetura básica e é baseada em três componentes tecnológicos padrão principais:

- HTML (HyperText Markup Language), que é uma linguagem para especificar o conteúdo e o leiaute de páginas de forma que elas possam ser exibidas pelos navegadores Web.
- URLs (Uniform Resource Locators), que identificam os documentos e outros recursos armazenados como parte da Web.
- Uma arquitetura de sistema cliente-servidor, com regras padrão para interação (o protocolo HTTP – HyperText Transfer Protocol) por meio das quais os navegadores e outros clientes buscam documentos e outros recursos dos servidores Web. A

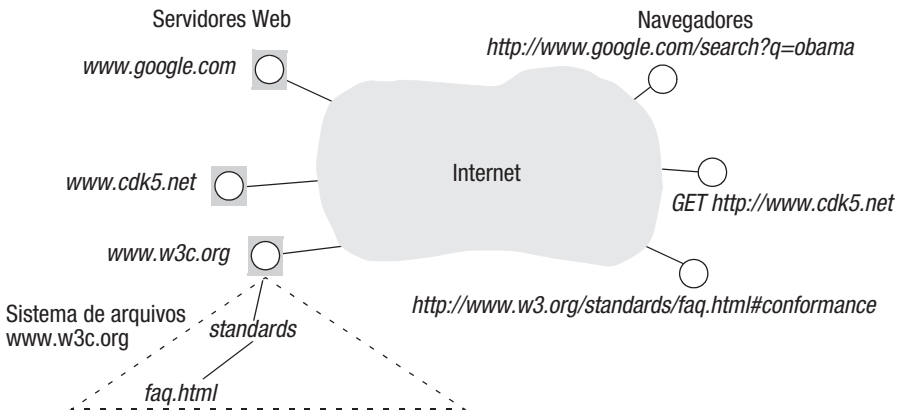

Figura 1.7 Servidores e navegadores Web.

Figura 1.7 mostra alguns navegadores realizando pedidos para servidores Web. É uma característica importante o fato de os usuários poderem localizar e gerenciar seus próprios servidores Web em qualquer parte da Internet.

Discutiremos agora cada um desses componentes e, ao fazermos isso, explicaremos o funcionamento dos navegadores e servidores Web quando um usuário busca páginas Web e clica nos *links* nelas existentes.

HTML • A HyperText Markup Language [www.w3.org II] é usada para especificar o texto e as imagens que compõem o conteúdo de uma página Web e para especificar como eles são dispostos e formatados para apresentação ao usuário. Uma página Web contém itens estruturados como cabeçalhos, parágrafos, tabelas e imagens. A HTML também é usada para especificar *links* e os recursos associados a eles.

Os usuários produzem código HTML manualmente, usando um editor de textos padrão ou um editor *wysiwyg* (*what you see is what you get*) com reconhecimento de HTML, que gera código HTML a partir de um leiaute criado graficamente. Um texto em HTML típico aparece a seguir:

```
<IMG SRC = "http://www.cdk5.net/WebExample/Images/earth.jpg">          1
<P>                                                                    2
Welcome to Earth! Visitors may also be interested in taking a look at the  3
<A HREF = "http://www.cdk5.net/WebExample/moon.html">Moon</A>.         4
</P>                                                                   5
```

Esse texto em HTML é armazenado em um arquivo que um servidor Web pode acessar – digamos que seja o arquivo *earth.html*. Um navegador recupera o conteúdo desse arquivo a partir de um servidor Web – neste caso específico, um servidor em um computador chamado *www.cdk5.net*. O navegador lê o conteúdo retornado pelo servidor e o apresenta como um texto formatado com as imagens que o compõe, disposto em uma página Web na forma que conhecemos. Apenas o navegador – não o servidor – interpreta o texto em HTML. Contudo, o servidor informa ao navegador sobre o tipo de conteúdo que está retornando, para distingui-lo de, digamos, um documento em Portable Document Format. O servidor pode deduzir o tipo de conteúdo a partir da extensão de nome de arquivo '.html'.

Note que as diretivas HTML, conhecidas como *tags*, são incluídas entre sinais de menor e maior, como em <*P*>. A linha 1 do exemplo identifica um arquivo contendo uma imagem de apresentação. Seu URL é *http://www.cdk5.net/WebExample/Images/earth.jpg*. As linhas 2 e 5 possuem as diretivas para iniciar e terminar um parágrafo, respectivamente. As linhas 3 e 4 contêm o texto a ser exibido na página Web, no formato padrão de parágrafo.

A linha 4 especifica um *link* na página Web. Ele contém a palavra *Moon*, circundada por duas tags HTML relacionadas <*A HREF...*> e </*A*>. O texto entre essas tags é o que aparece no *link* quando ele é apresentado na página Web. Por padrão, a maioria dos navegadores é configurada de modo a mostrar o texto de *links* sublinhado; portanto, o que o usuário verá nesse parágrafo será:

Welcome to Earth! Visitors may also be interested in taking a look at the Moon.

O navegador grava a associação entre o texto exibido do *link* e o URL contido na tag <*A HREF...*> – neste caso:

http://www.cdk5.net/WebExample/moon.html

Quando o usuário clica no texto, o navegador recupera o recurso identificado pelo URL correspondente e o apresenta para o usuário. No exemplo, o recurso está em um arquivo HTML que especifica uma página Web sobre a Lua.

URLs • O objetivo de um URL (Uniform Resource Locator) [www.w3.org III] é identificar um recurso. Na verdade, o termo usado em documentos que descrevem a arquitetura da Web é URI (Uniform Resource Identifier), mas, neste livro, o termo mais conhecido, URL, será usado quando não houver uma possível ambiguidade. Os navegadores examinam os URLs para acessar os recursos correspondentes. Às vezes, o usuário digita um URL no navegador. Mais comumente, o navegador pesquisa o URL correspondente quando o usuário clica em um *link*, quando seleciona um URL de sua lista de *bookmarks* ou quando o navegador busca um recurso incorporado em uma página Web, como uma imagem.

Todo URL, em sua forma completa e absoluta, tem dois componentes de nível superior:

esquema: identificador-específico-do-esquema

O primeiro componente, o "esquema", declara qual é o tipo desse URL. Os URLs são obrigados a identificar uma variedade de recursos. Por exemplo, *mailto:joe@anISP.net* identifica o endereço de *e-mail* de um usuário; *ftp://ftp.downloadIt.com/software/aProg.exe* identifica um arquivo que deve ser recuperado com o protocolo FTP (File Transfer Protocol), em vez do protocolo mais comumente usado, HTTP. Outros exemplos de esquemas são "tel" (usado para especificar um número de telefone a ser discado, o que é particularmente útil ao se navegar em um telefone celular) e "tag" (usado para identificar uma entidade arbitrária).

A Web não tem restrições com relação aos tipos de recursos que pode usar para acesso, graças aos designadores de esquema presentes nos URLs. Se alguém inventar um novo tipo de recurso, por exemplo, *widget* – talvez com seu próprio esquema de endereçamento para localizar elementos em uma janela gráfica e seu próprio protocolo para acessá-los – , então o mundo poderá começar a usar URLs da forma *widget:...* É claro que os navegadores devem ter a capacidade de usar o novo protocolo *widget*, mas isso pode ser feito pela adição de um *plugin*.

Os URLs HTTP são os mais comuns para acessar recursos usando o protocolo HTTP padrão. Um URL HTTP tem duas tarefas principais a executar: identificar qual servidor Web mantém o recurso e qual dos recursos está sendo solicitado a esse servidor. A Figura 1.7 mostra três navegadores fazendo pedidos de recursos, gerenciados por três servidores

Web. O navegador que está mais acima está fazendo uma consulta em um mecanismo de busca. O navegador do meio solicita a página padrão de outro *site*. O navegador que está mais abaixo solicita uma página Web especificada por completo, incluindo um nome de caminho relativo para o servidor. Os arquivos de determinado servidor Web são mantidos em uma ou mais subárvores (diretórios) do sistema de arquivos desse servidor, e cada recurso é identificado por um nome e um caminho relativo (*pathname*) para esse servidor.

Em geral, os URLs HTTP são da seguinte forma:

http:// nomedoservidor [:porta] [/nomedeCaminho] [?consulta][#fragmento]

onde os itens entre colchetes são opcionais. Um URL HTTP completo sempre começa com o *string http*://, seguido de um nome de servidor, expresso como um nome DNS (Domain Name System) (consulte a Seção 13.2). Opcionalmente, o nome DNS do servidor é seguido do número da porta em que o servidor recebe os pedidos (consulte o Capítulo 4) – que, por padrão, é a porta 80. Em seguida, aparece um nome de caminho opcional do recurso no servidor. Se ele estiver ausente, então a página Web padrão do servidor será solicitada. Por fim, o URL termina, também opcionalmente, com um componente de consulta – por exemplo, quando um usuário envia entradas de um formulário, como a página de consulta de um mecanismo de busca – e/ou um identificador de fragmento, que identifica um componente de um determinado recurso.

Considere os URLs a seguir:

http://www.cdk5.net
http://www.w3.org/standards/faq.html#conformance
http://www.google.com/search?q=obama

Eles podem ser decompostos, como segue:

Server DNS name	*Path name*	*Query*	*Fragment*
www.cdk5.net	(default)	(none)	(none)
www.w3.org	standards/faq.html	(none)	intro
www.google.com	search	q=obama	(none)

O primeiro URL designa a página padrão fornecida por *www.cdk5.net*. O seguinte identifica um fragmento de um arquivo HTML cujo nome de caminho é *standards/faq.html*, relativo ao servidor *www.w3.org*. O identificador do fragmento (especificado após o caractere # no URL) é *conformance* e um navegador procurará esse identificador de fragmento dentro do texto HTML, após ter baixado o arquivo inteiro. O terceiro URL especifica uma consulta para um mecanismo de busca. O caminho identifica um programa chamado de "search" e o *string* após o caractere ? codifica um *string* de consulta fornecido como argumento para esse programa. Discutiremos os URLs que identificam recursos de programa com mais detalhes a seguir, quando considerarmos os recursos mais avançados.

Publicação de um recurso: Embora a Web tenha um modelo claramente definido para acessar um recurso a partir de seu URL, os métodos exatos para publicação de recursos dependem da implementação do servidor. Em termos de mecanismos de baixo nível, o método mais simples de publicação de um recurso na Web é colocar o arquivo correspondente em um diretório que o servidor Web possa acessar. Sabendo o nome do servidor, *S*, e um nome de caminho para o arquivo, *C*, que o servidor possa reconhecer, o usuário constrói o URL como *http://S/C*. O usuário coloca esse URL em um *link* de um documento já existente ou informa esse URL para outros usuários de diversas formas como, por exemplo, por *e-mail*.

É comum tais preocupações ficarem ocultas dos usuários, quando eles geram conteúdo. Por exemplo, os *bloggers* normalmente usam ferramentas de *software*, elas próprias implementadas como páginas Web, para criar coleções de páginas de diário organizadas. As páginas de produtos do *site* de uma empresa normalmente são criadas com um *sistema de gerenciamento de conteúdo*; novamente, pela interação direta com o *site*, por meio de páginas Web administrativas. O banco de dados, ou sistema de arquivos, no qual as páginas de produtos estão baseadas é transparente.

Por fim, Huang *et al*. [2000] fornece um modelo para inserir conteúdo na Web com mínima intervenção humana. Isso é particularmente relevante quando os usuários precisam extrair conteúdo de uma variedade de equipamentos, como câmeras, para publicação em páginas Web.

HTTP • O protocolo HyperText Transfer Protocol [www.w3.org IV] define as maneiras pelas quais os navegadores e outros tipos de cliente interagem com os servidores Web. O Capítulo 5 examina o protocolo HTTP com mais detalhes, mas destacaremos aqui suas principais características (restringindo nossa discussão à recuperação de recursos em arquivos):

Interações requisição-resposta: o protocolo HTTP é do tipo requisição-resposta. O cliente envia uma mensagem de requisição para o servidor, contendo o URL do recurso solicitado. O servidor pesquisa o nome de caminho e, se ele existir, envia de volta para o cliente o conteúdo do recurso em uma mensagem de resposta. Caso contrário, ele envia de volta uma resposta de erro, como a conhecida mensagem *404 Not Found*. O protocolo HTTP define um conjunto reduzido de operações ou *métodos* que podem ser executados em um recurso. Os mais comuns são GET, para recuperar dados do recurso, e POST, para fornecer dados para o recurso.

Tipos de conteúdo: os navegadores não são necessariamente capazes de manipular todo tipo de conteúdo. Quando um navegador faz uma requisição, ele inclui uma lista dos tipos de conteúdo que prefere – por exemplo, em princípio, ele poderia exibir imagens no formato GIF, mas não no formato JPEG. O servidor pode levar isso em conta ao retornar conteúdo para o navegador. O servidor inclui o tipo de conteúdo na mensagem de resposta para que o navegador saiba como processá-lo. Os *strings* que denotam o tipo de conteúdo são chamados de tipos MIME e estão padronizados na RFC 1521 [Freed e Borenstein 1996]. Por exemplo, se o conteúdo é de tipo *text/html*, então um navegador interpreta o texto como HTML e o exibe; se o conteúdo é de tipo *image/GIF*, o navegador o representa como uma imagem no formato GIF; se é do tipo *application/zip*, dados compactados no formato zip, o navegador ativa um aplicativo auxiliar externo para descompactá-lo. O conjunto de ações a serem executadas pelo navegador para determinado tipo de conteúdo pode ser configurado, e os leitores devem verificar essas configurações em seus próprios navegadores.

Um recurso por requisição: os clientes especificam um recurso por requisição HTTP. Se uma página Web contém, digamos, nove imagens, o navegador emite um total de 10 requisições separadas para obter o conteúdo inteiro da página. Normalmente, os navegadores fazem vários pedidos concorrentes para reduzir o atraso global para o usuário.

Controle de acesso simplificado: por padrão, qualquer usuário com conectividade de rede para um servidor Web pode acessar qualquer um de seus recursos publicados. Se for necessário restringir o acesso a determinados recursos, isso pode ser feito configurando o servidor de modo a emitir um pedido de identificação para qualquer cliente que o solicite. Então, o usuário precisa provar que tem direito de acessar o recurso, por exemplo, digitando uma senha.