Capítulo 1

Introdução aos Sistemas Distribuídos

Elias P. Duarte Jr. - versão do dia 21/09/2021

Nas últimas décadas a Internet revolucionou o mundo, modificando efetivamente a forma como seres humanos se comunicam, trabalham e se divertem. A Internet consiste, na verdade, de múltiplas redes de computadores conectadas em todo o planeta como se fosse uma única rede. O software que executa sobre essas redes é um sistema distribuído, que consiste de um conjunto de processos, que são programas em execução em múltiplos computadores da rede, cooperando para a realização das mais diversas tarefas.

Encontramos sistemas distribuídos dos mais diversos tipos. Temos, por exemplo: bancos de dados distribuídos, sistemas operacionais distribuídos, e muitas, muitas aplicações distribuídas, algumas delas fazendo parte do nosso dia-a-dia. Podemos citar os sistemas bancários que nos permitem fazer movimentações bancárias a partir de qualquer dispositivo ou terminal, ou os sistemas de reserva de passagens áreas, além de tantas aplicações na Web de comércio eletrônico, colaboração, comunicação. Para construir sistemas distribuídos os desafios vão desde a construção da interface até os mecanismos internos de comunicação entre os diversos componentes espalhados pelas redes subjacentes.

Neste livro tratamos, por assim dizer, da “alma” que permeia os sistemas distribuídos: os *algoritmos distribuídos*. O objetivo do livro é apresentar uma introdução – a mais didática que conseguirmos – aos algoritmos distribuídos. Na medida em que um algoritmo é uma sequência finita de instruções para resolver um problema, um algoritmo distribuído é uma sequência de instruções que é executada de forma colaborativa por múltiplos processos para resolverem um problema específico. Os desafios para o desenvolvimento de algoritmos distribuídos são, algumas vezes, insuperáveis. Não são poucos os resultados de impossibilidades provados na resolução de problemas por múltiplos processos comunicando em uma rede. Talvez por este motivo, o tratamento dado aos algoritmos distribuídos tem sido eminentemente teórico, fortemente calcado em especificações que apresentam rigoroso formalismo matemático. É realmente notável como, por

exemplo, a maioria dos livros que já foram publicados sobre Algoritmos Distribuídos tem viés absolutamente teórico, com foco principal em especificações formais e provas de correção. Uma lista comentada de alguns dos mais importantes livros publicados na área é apresentada como apêndice.

Acreditamos sinceramente que este livro fecha uma lacuna importante. Concordamos que a apresentação formal de algoritmos distribuídos, permeada de resultados expressos através de lemas e teoremas, é fundamental para o estabelecimento das bases e o avanço da área. Entretanto, ela pode representar uma barreira formidável para aqueles que estão iniciando o estudo da área. Desta forma, o objetivo deste livro é justamente ser uma introdução gentil, ainda que rigorosa, aos algoritmos distribuídos. O objectivo número um do livro é ser **didático**, de forma a permitir que o leitor *compreenda* com facilidade os assuntos que são apresentados. Depois de algumas décadas trabalhando na área, é fácil perceber que há pouco material para apoiar aquele que está dando os primeiros passos. O livro explica modelos básicos e uma série de algoritmos clássicos que são, por assim dizer, o núcleo de tantos sistemas distribuídos que usamos no nosso dia-a-dia. Fizemos, como professor da área ao longo de décadas, um esforço sincero para **explicar** o funcionamento de alguns dos mais importantes algoritmos distribuídos de uma maneira clara, e objetiva. Esperamos sinceramente que a forma como o livro foi escrito ajude o leitor a efetivamente compreender os meandros dos trabalhos que são apresentados aqui, e que represente uma base, para aqueles que se interessarem pela área de seguir em frente, quiçá como pesquisadores ou desenvolvedores dos mais diversos tipos de aplicações baseadas em algoritmos distribuídos.

Neste capítulo começamos definindo exatamente o que chamamos de sistema distribuído neste livro. A seguir focamos na noção de “tempo” e as restrições temporais nos modelos de sistemas distribuídos.

1.1 Afinal, o que é um sistema distribuído?

Um sistema distribuído é um conjunto de processos que se comunicam e colaboram para realizar uma tarefa. A tarefa podem ser de qualquer natureza, desde aplicações específicas como um sistema de compras de uma loja virtual, até mesmo genéricas, como os sistemas de middleware que dão suporte à criação de aplicações distribuídas. **Um processo é um programa em execução**. Onde é executado o processo? Pode ser qualquer tipo de dispositivo, como um computador, ou um

celular ou até mesmo um relógio ou aparelho de televisão, qualquer dispositivo de hardware que possa executar programas.

Como dito no parágrafo anterior, um sistema distribuído é então um *conjunto de processos* que se comunicam para realizar uma tarefa. Quantos são os processos do conjunto? Este número varia? Neste livro vamos considerar que o sistema distribuído tem um número fixo e conhecido de processos. Diz-se que se trata de um sistema distribuído **estático**. O número exato de processos varia de sistema para sistema, assim vamos dizer que há **N** processos em execução. Claro que N deve ser maior que 1: se tivermos um único processo não temos um sistema distribuído realmente. Assim *n* pode ser 2, 3, 1000, 1 milhão, ou 1 bilhão, calma comecei a exagerar. Mas realmente existem sistemas distribuídos com milhares e até mesmo milhões de processos. Estes processos são e estão lá em execução. E se o sistema não cumprir estas restrições? Veja que é fácil pensar em um sistema que não cumpre estas restrições, em que novos processos aparecem e outros processos deixam o sistema. Estes sistemas são ditos **dinâmicos**, mas não tratamos deles no livro. Reforçando: neste livro tratamos exclusivamente de sistemas distribuídos estáticos.

Podemos dar um nome ao conjunto de processos que forma o sistema distribuído, por exemplo chamando-o de *S*, como um apelido curto. Vamos dar nomes aos processos também: se um sistema tiver N processos, vamos de alguma forma identificar cada um deles como processo 1, processo 2, … processo N. Às vezes vamos abreviar como p_1, p_2, …, p_N. Vale a pena reforçar que os processos têm também seus identificadores de rede -- por exemplo, um processo que comunica na Internet é identificado pelo par (endereço IP, porta). Os identificadores sequenciais 1, 2, …, N não tem nada a ver com aqueles identificadores de rede, são apenas identificadores para facilitar para nós, humamos que estamos trabalhando com sistemas distribuídos. Assim fica fácil fazermos referência aos processos individualmente. Quando se implementa um sistema distribuído basta mapear os identificadores sequenciais para os identificadores da rede.

1.2 Paradigmas de Comunicação: Troca de Mensagens & Memória Compartilhada

Como ocorre a comunicação entre os processos? Há basicamente duas formas distintas para os processos comunicarem, que são os **paradigmas de comunicação**: **troca de mensagens** e **memória compartilhada**. A primeira delas é a que vamos usar ao longo de todo este livro: a **troca**

de mensagens. Uma mensagem é uma unidade de informação que trafega sobre uma rede que conecta os processos que se comunicam. Para transmitir mensagens entre processos, as redes implementam protocolos em camadas, sendo hoje os protocolos TCP/IP da Internet adotados em praticamente todas as redes que encontramos no nosso dia-a-dia, no mundo todo. Desta forma, a rede oferece a infraestrutura e os mecanismos para os processos trocarem mensagens entre si.

A Figura 1 mostra os processos A e B, representados como círculos, conectados entre si através de um canal de comunicação. O canal de comunicação pode ser de qualquer tecnologia. Inclusive podem haver diversos roteadores no caminho entre os processos, passando por redes das mais diversas tecnologias. Alternativamente os dois processos podem estar executando em uma mesma rede local, ou até mesmo na mesma máquina (*host*, no jargão da Internet). No livro vamos abstrair estes diferentes tipos de comunicação: não importa como examente os processos estão conectados, simplemente vamos considerar que há um **canal de comunicação** entre eles, através do qual se comunicam trocando mensagens.

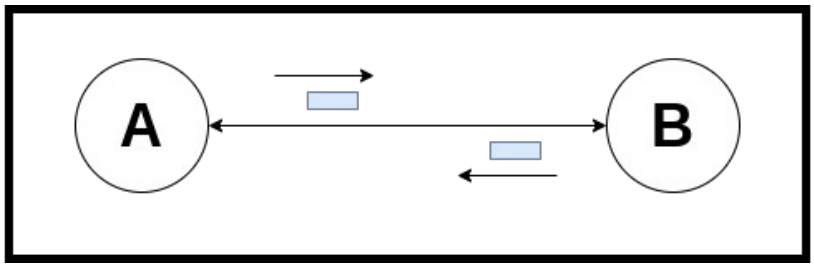

Figura 1: Os processos A e B trocam mensagens sobre um canal de comunicação.

O paradigma alternativo à troca de mensagens é o da memória compartilhada. Neste caso, os processos se comunicam acessando uma área comum de memória, por exemplo uma variável compartilhada. Se o processo origem escreve a informação na memória compartilhada, em um momento posterior o processo destino pode ler aquela informação da memória compartilhada. Na Figura 2, os processos A e B compartilham uma variável inteira na qual o processo A escreve o valor 10, que é lido pelo processo B, assim ocorre a transferência de informação entre os processos. O paradigma da memória compartilhada tem se tornado cada vez mais importante, com a popularização de processadores com múltiplos núcleos que executam processos independentes que podem se comunicar com facilidade compartilhando a memória da máquina. Reforçamos, entretanto, que neste livro vamos focar exclusivamente no paradigma da troca de mensagens.

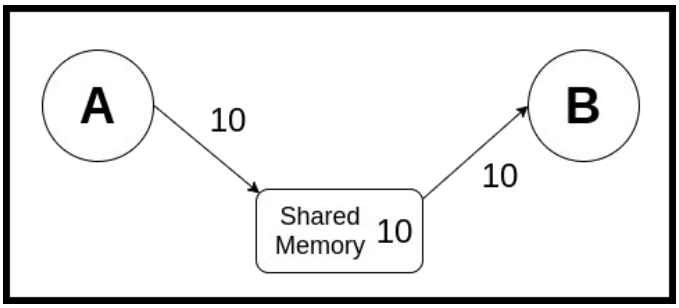

Figura 2: Os processos A e B se comunicam através de memória compartilhada.

1.3 Topologias de Sistemas Sistemas Distribuídos: *Fully Connected* & *Multi-Hop*

O conjunto de processos que constituem os sistema distribuído, podem se organizar de duas maneiras em termos da **topologia** que formam. Esta classificação é muito usada, e mesmo no Brasil é comum chamá-las pelos seus nomes em inglês: *fully-connected* e *multi-hop*, descritas a seguir. A topologia mais comum, que é a única considerada neste livro, é a ***fully-connected***, em que todos os pares de processos são conectados entre si. Uma possível tradução para o português é sistema totalmente conectado. Um sistema distribuído com topologia *fully-connected* é representável por um grafo completo, no qual todos os vértices (representando os processos) são adjacentes entre si. A Figura 3 ilustra esta topologia: todos os 5 processos são adjacentes entre si**.** Em um sistema *fully-connected*, qualquer processo pode comunicar diretamente com qualquer outro processo sem precisar que as mensagens passem por processos intermediários. Veja que não necessariamente há enlaces físicos conectando as máquinas nas quais os processos executam. Esta topologia representa um conjunto de processos executando na mesma máquina, ou na mesma rede local. Na verdade, as arestas não representam enlaces físicos. Por exemplo, uma rede Ethernet corresponde a um sistema *fully-connected:* todos os hosts comunicam diretamente entre si e estão conectados por um único canal de comunicação, neste caso compartilhado. Em outro exemplo, se os processos estão espalhados na Internet, ainda assim é possível usar a topologia como fully-connected para representar a capacidade de qualquer par de processos de comunicarem diretamente entre si, sem utilizar outros processos como intermediários.

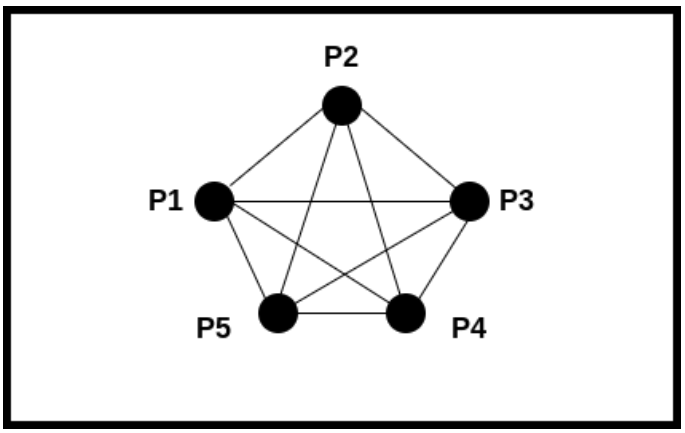

Figura 3: Um sistema distribuído com topologia *fully-connected*.

Caso haja no sistema distribuído um ou mais pares de processos que não conseguem comunicar diretamente entre si, então a topologia que representa o sistema é dita ***multi-hop***. A Figura 4 ilustra um sistema distribuído com a topologia *multi-hop*. Observe, por exemplo, que uma mensagem transmitida do processo P1 para o processo P4 tem que necessariamente passar por outro processo, como por exemplo pelo processo P2, não há alternativa de comunicação direta entre P1 e P4. Observe que P2 é um processo mesmo, não se trata de roteador ou dispositivo de qualquer outra natureza, é um processo como os demais. É neste sentido que o sistema é dito ser constituído de múltiplos "saltos" (*hops*) para a comunicação de certos pares de processos. Um salto ocorre sempre de um processo para outro do mesmo sistema distribuído. Os caminhos que as mensagens percorrem entre pares de processos vão variar de acordo com a topologia específica. No caso dos sistemas distribuídos multi-hop, dizemos que as topologias são "arbitrárias" ou "gerais", não há um tipo de grafo específico que representa o sistema, varia caso-a-caso.

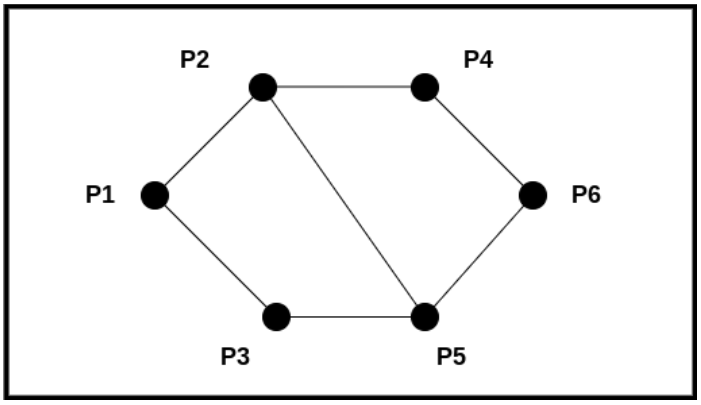

FIGURA 4: Um sistema distribuído com topologia *multi-hop*.

1.4 Primitivas de Comunicação

Para programar a comunicação de processos, o programador tem acesso normalmente a três operações, que são chamadas no código através de **primitivas de comunicação**. Duas delas são muito intuitivas, e vamos começar por elas. Uma é a operação executada para transmitir, enviar uma mensagem de um processo origem para um processo destino. A outras é a operação executada para receber uma mensagem. Vamos adotar a nomenclatura universal para estas operações:

- ***send*(*msg*)**: operação utilizada para transmitir uma mensagem *msg*
- ***receive*(*msg*)**: operação executada para receber uma mensagem *msg*

Observe que a primitiva específica para programar estas operações pode envolver outros parâmetros, implícita e explicitamente. Por exemplo, o endereço do destinatário é essencial para fazer a transmissão de uma mensagem. Enviar a mensagem para quem? Os endereços de processos são seus identificadores, e se considerarmos a Internet, um processo é identificado pelo par (endereço IP, porta) que são os identificadores das camadas de redes (protocolo IP) e transporte (protocolos TPC, UDP ou outro). No livro algumas vezes vamos explicitar os endereços na primitivas e, ocasionalmente, outros parâmetros também. Por exemplo, podemos usar *send(origem, destino, msg)* para indicar que o processo *origem* está transmitindo a mensagem *msg* para o processo *destino.* Reforçando: neste caso tanto *origem* como *destino* são endereços dos processos.

Programadores de diferentes liguagens/sistemas operacionais têm acesso a diferentes bibliotecas para programar processos que se comunicam. Sem sombra de dúvidas, a biblioteca *socket* [socket-ref] original do sistema BSD Unix, mas hoje muito popular. Presente em diversos sistemas, o *socket* foi também influênca para virtualmente todas as outras atualmente em uso. A biblioteca *socket* tem as primitivas *send()* e *receive()*, além de diversas outras.

Na verdade há uma terceira primitiva de comunicação básica, além do *send()* e *receive()*, como mencionamos acima. Esta terceira primitiva é a seguinte:

- ***deliver*(*msg*)**: operação utilizada para entregar uma mensagem recebida para a aplicação

Um exemplo bastante intuitivo para você compreender a operação *deliver*(*msg*) é pensando em sistemas de e-mail. Considere que você está usando um sistema para ler e-mail, que na figura acima é a aplicação. Considere também que o processo que recebe uma nova mensagem vinda da rede executa um classificador que detecta *spam*. Assim, quando chega uma nova mensagem, o processo identifica se é legítima ou *spam.* Se a conclusão é que se trata de uma mensagem legítima, ela vai ser entregue (através da execução da operação *deliver*) para que você a receba no seu leitor de e-mail. Caso contrário, se a conclusão é que se trata de *spam*, a mensagem é simplesmente descartada, a operação *deliver*(*msg*) não é executada neste caso.

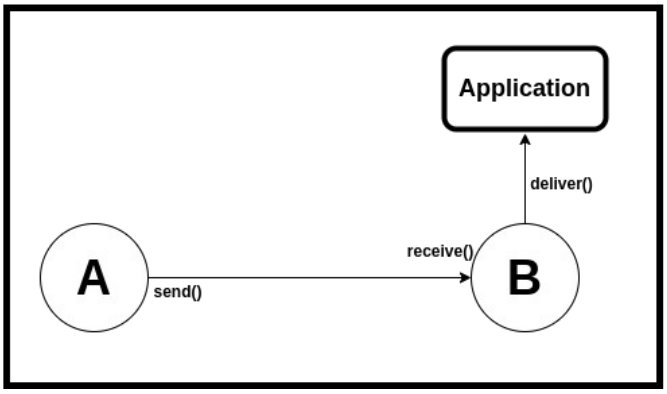

FIGURA 5: As três primitivas básicas de comunicação de processos: *send*, *receive* & *deliver.*

1.5 O Tempo e Os Sistemas Distribuídos

Agora já temos uma definição bem precisa do que é um sistema distribuído neste livro: trata-se de um conjunto de N processos que se comunicam trocando mensagens, colaborando para solucionar um problema. Lembre-se que N corresponde ao número fixo e conhecido de processos, e que cada processo pode se comunicar diretamente com qualquer outro, isto é, não precisa passar por intermediários. Então agora vamos pensar como funciona nosso sistema distribuído. Os N processos estão em execução, trocando mensagens. Um processo digamos *a* envia uma mensagem para outro processo *b*. A mensagem é transmitida através de um canal de comunicação até ser recebida pelo processo *b*. Ao receber a mensagem, o processo *b* executa algumas operações e decide se vai entregar a mensagem para a aplicação. Na verdade, um processo está sempre executando tarefas, indicadas pelas instruções dos programas que executam. De acordo com os dados locais ou informações trocadas nas mensagens, tarefas diferentes podem ser executadas. Após enviar a mensagem para *b*, muito provavelmente *a* aguarda uma resposta de *b*.

A pergunta que surge é: quanto tempo demora para a mensagem do processo *a* chegar em *b*? E quanto tempo demora para *b* processar a mensagem e retornar uma resposta para *a*? Observe que este tempo é importante inclusive para os processos definirem até quando vão esperar uma resposta, por exemplo. Pode parecer que não, mas estes tempos são fundamentais em sistemas distribuídos. Por exemplo, se uma resposta demora muito a chegar e o processo que estava esperando desiste de esperar e toma alguma decisão a partir da desistência, podemos ter problemas graves. Por exemplo, o processo remoto pode ter alterado o valor de um objeto replicado, que o processo que desistiu de esperar pela resposta pode não aplicar aquela alteração em sua cópia local.

Como lidar com a noção de “tempo” no sistema distribuído? Observe que mesmo para nós humanos é uma noção complexa – pense comigo, não é fácil definir o que é “tempo”. Calma, nos sistemas distribuídos não vamos investigar filosoficamente o que é o tempo, as coisas vão ser mais simples. O tempo é marcado pelo relógio local do processador (CPU) sobre o qual o processo está executando. Vamos tratar de limites – na verdade, só há dois limites de tempo realmente relevantes: o limite de tempo para transmitir uma mensagem e o limite de tempo para executar uma tarefa. O limite de tempo de transmissão de mensagem é contado desde que a operação *send()* é executada na origem, até que a operação *receive()* complete com sucesso no destino. Quanto ao limite de tempo

para executar uma tarefa, definindo uma tarefa como uma sequência de instruções, efetivamente estamos afirmando que cada instrução sempre completa sua execução dentro de um limite máximo de tempo.

Uma das classificações mais importantes dos sistemas distribuídos sãoé baseada justamente nestes dois limites de tempo (para transmitir uma mensagem e para executar uma tarefa). De um lado temos os sistemas **síncronos**, em que há limites *conhecidos* de tempo tanto para transmitir uma mensagem, quanto para executar uma tarefa. Por favor preste atenção na palavra “conhecidos” que coloquei em itálico na frase anterior. Para um sistema ser dito síncrono, não basta que exista um limite máximo dos tempos, precisa mais! Os limites de tempo devem ser *conhecidos*. Quem usa o sistema deve saber com precisão absoluta quais são tais limites.

Me chama a atenção o tanto que esta definição de **sistema distribuído síncrono** não é normalmente compreendida da forma correta pelos alunos que a escutam pela primeira vez. Arrisco identificar uma razão: imaginamos de forma ingênua que um sistema que obedece limites de tempo com muita frequência (digamos, 99.999% das vezes) seria síncrono, mas *não é*! Não basta que os limites temporais sejam obedecidos com muita frequência, para um sistema ser dito síncrono, os limites temporais tem que ser obedecidos *sempre*. Sempre é sempre, 100% das vezes. Por exemplo, considere que em uma rede específica existe a noção que demora no máximo 0.1 segundo para transmitir uma mensagem entre dois processos. Se você, durante um ano inteiro, fica medindo o tempo na rede e observan que todas as mensagens levaram menos de 0.1 segundo para chegar, exceto uma única vez… Pronto, o sistema já não é síncrono, o limite foi violado. Por outro lado, em um **sistema distribuído assíncrono** nenhum limite temporal é conhecido. Mais que isso, um sistema assíncrono é definido sem levar em conta a noção de tempo.

Agora vamos pensar nesta classificação dos sistemas distribuídos em síncronos e assíncronos no contexto da Internet. Se você está achando que não é fácil saber qual o limite máximo de transmissão de mensagens entre processos na Internet, é isso mesmo. É muito frequente vivermos situações por exemplo em que estamos usando um *browser* para acessar o conteúdo em algum servidor Web. O *browser* então mostra uma mensagem de erro com uma carinha triste dizendo que o servidor não respondeu. Ora, o que será que aconteceu? Será que a mensagem esperada do servidor que queremos acessar não vai chegar mesmo? E se chegar um pouquinho atrasada? Pode ser inclusive que a rede esteja desconectada, ou que o servidor tenha caído. Aqui no contexto dos

modelos temporais precisamos reforçar que a hipótese do tempo de espera da resposta insufiente é sólida: o *browser* espera durante um intervalo calculado de forma heurística, não há limites precisos conhecidos.

Situação parecida ocorre para o limite de tempo de execução de tarefas. Na verdade os limites de tempo para um processo executar uma tarefa também são difíceis de precisar. Pense só na seguinte situação: tem um processo executando agora no seu computador pessoal. De repente, começa a rodar um antivirus, ou outra aplicação qualquer que o sistema operacional dispara ocasionalmente. Pronto. O tempo para executar um programa agora já vai ser diferente daquele sem este estorvo. Veja que neste exemplo sequer estamos considerando que a máquina na qual o processo executa é compartilhada por múltiplos usuários, caso em que prever a carga em um instante de tempo específico se torna ainda mais difícil.

Desta forma, o modelo síncrono pode ser considerado pouco realista, no sentido que não corresponde aos sistemas reais. Veja que mesmo com tecnologias diversas disponíveis, como sistemas distribuídos de tempo real, como RT Linux e tecnologias para garantir a qualidade de serviço de rede (QoS - *Quality of Service*) na Internet é difícil conseguir sistemas distribuídos que se mantém verdadeiramente síncronos de forma contínua permanente e indefinidamente, sem qualquer violação dos limites. Seria interessante um *benchmark* mundial, especialmente no contexto da Internet, do quanto é possível conseguir neste sentido, usando toda a tecnologia disponível hoje. Fica o desafio para meus leitores!

Por outro lado, os modelo assíncrono nos diz muito pouco sobre o sistema. Claramente, é viável implementar e manter sistemas distribuídos que assumem certos limites temporais, inclusive na Internet. Vamos pensar em uma representação em que sistemas distribuídos são pontos de um segmento de reta cujos extremos correspondem aos sistemas síncronos e assíncronos. Os sistemas reais estão claramente *entre* os dois extremos. A Figura 6 ilustra esta analogia.

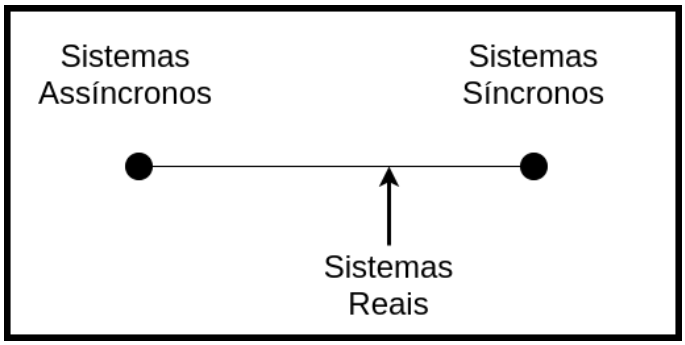

FIGURA 6: Sistemas distribuídos reais: em algum ponto entre síncrono e assíncrono.

Ao longo das últimas décadas, foram propostos diversos modelos ditos *parcialmente síncronos* justamente com o objetivo de melhor aproximar o modelo do sistema real. Estes modelos parcialmente síncronos variam imensamente sobre a forma que tratam tempo e os limites temporais. Por exemplo, há modelos mais simples que consideram por exemplo que há um limite conhecido para a transmissão de mensagens e não há limite conhecido para execução de tarefas, ou vice-versa. Outros modelos são mais elaborados, inclusive aqueles com tratamento probabilístico do tempo. Aqui vamos nos concentrar no modelo parcialmente síncrono que é talvez o modelo temporal mais usado atualmente na área de sistemas sistemas distribuídos. Trata-se do modelo GST, sigla que vem do inglês: *Global Stabilization Time.* No modelo GST, o sistema inicia assíncrono, sem respeitar qualquer limite de tempo. Entretanto, a partir de um certo instante de tempo, justamente denominado GST, o sistema se torna síncrono, passando a respeitar limites de tempo máximo de transmissão de mensagens e para a execução de tarefas. O instante exato em que o sistema faz a transição não é conhecido *a priori*, mas existe. Além disso, depois que a transição para o modelo síncrono ocorre, o sistema permece síncrono “para sempre”. Este termo “para sempre” é muito usado em sistemas distribuídos, e não significa até o final dos tempos! Significa apenas que o sistema permance síncrono enquanto isso for importante, por exemplo até o final da execução de um algoritmo.

O modelo GST é muito prático, na prática mesmo ele diz que o sistema distribuído vai ter condições temporais para fazer o que é necessário. Vale a pena fazer um questionamento: e se eu assumir o modelo GST, mas o sistema não se comportar da forma prevista no modelo? Por exemplo, se o

sistema insistir em ficar assíncrono de novo depois do instante de tempo GST, como fica a situação? Bom, se você construiu o seu sistema distribuído com as premissas do modelo GST e uma execução real do sistema não respeita aquelas premissas, então o sistema pode não fazer aquilo para o qual foi feito. Simples assim. O impacto deste fato depende de cada sistema. Veja que se a realidade não reflete o modelo basta trocar para outro sistema que seja baseado em outro modelo que sim reflita o que está acontecendo na prática.

1.6 Concluindo

Neste capítulo definimos precisamente que um sistema distribuído é um conjunto de N processos que se comunicam e cooperam para a realização de alguma tarefa. Um processo é um programa em execução. Se o número de processos é constante, como na definição acima em que temos N processos, então o sistema distribuído é dito estático, caso contrário se sua composição varia ao longo do tempo, é dito dinâmico. Vimos que os processos podem se comunicar de acordo com um de dois paradigmas: troca de mensagens ou memória compartilhada. No livro vamos tratar exclusivamente dos sistemas baseados em troca de mensagens, e estudamos as primitivas de comunicação: *send(msg)*, *receive(msg)* e *deliver(msg).* A topologia dos sistemas distribuídos também é classificada em duas alternativas: ou o sistema é *fully-connected*, em que qualquer par de processos se comunica diretamente sem intermediários, ou é multi-hop, com topologia arbitrária em que mensagens entre alguns pares de processos devem ser roteadas passando por intermediários. Por fim, estudamos os modelos temporais de sistemas distribuídos, definindo sistemas síncronos, assíncronos e parcialmente síncronos, nesta última categoria com foco especial no modelo GST.

No próximo capítulo, o foco continua em modelos de sistemas distribuídos, agora definindo as falhas que podem ocorrer e as técnicas de tolerância a falhas.