Distributed Systems Concepts and Design

2 Modelos de Sistemas

Modelos de arquitetura de sistemas distribuídos, estão relacionado com o local onde estão as partes e o relacionamento entre elas. Como exemplo podemos citar o modelo Cliente Servidor e o modelo peer to peer.

Modelos fundamentais, estão relacionados com uma descrição mais formal das propriedades que são comuns em todos os modelos de arquitetura.

2.1 Introdução

Sistemas que são produzidos para trabalhar em ambiente real em qualquer lugar do mundo, precisam ser projetados para funcionar corretamente no maior numero possível de circunstancias e em face das muitas possíveis dificuldades e ameaças tais como:

- Grande variação de modos de uso – As partes componentes dos sistemas estão sujeitas a grande variação da carga de trabalho. Algumas paginas web são acessadas milhões de vezes ao dia. Algumas partes dos sistemas podem se desconectar ou estar conectada com dificuldades parte do tempo em especial quando se tem comunicação móvel no sistema. Algumas aplicações requerem grande largura de banda e baixa latência por exemplo aplicações multimídia.
- Grande variação de ambientes – Um sistema distribuído precisa acomodar hardwares, sistemas operacionais e redes heterogêneas. A rede pode variar muito em performance, redes sem fio trabalham a uma velocidade fracionaria de uma rede LAN. Sistemas de diferentes tamanhos precisam ser suportados, variando de dezenas de computadores a milhões de computadores.

- Problemas internos – Relógios não sincronizados, atualização de dados conflitantes, muitas formas de falhas de hardware e software envolvendo componentes individuais do sistema.
- Ameaças externas – Ataque a integridade e segurança dos dados. Ataques do tipo *denial of service*, excessiva requisição de serviços.

2.2 Modelos de Arquitetura

Um modelo de arquitetura de sistemas distribuídos primeiro simplifica e abstrai as funções dos componentes individuais de um sistema distribuído e então considera:

- A localização dos componentes através da rede de computadores – procurando definir padrões úteis para a distribuição dos dados e da carga de processamento.
- A relação entre os componentes – isto é suas regras funcionais e os padrões de comunicação entre eles.

Uma simplificação inicial é realizada classificando os processos como processos servidores, processos clientes e processos *peer* sendo este último processo que coopera e se comunica de maneira simétrica para realizar um trabalho. Esta classificação dos processos identifica a responsabilidade de cada um e daqui nos ajuda a taxar a carga de processamento e determinar o impacto de falhas em cada um deles. O resultado desta analise pode então ser usado para especificar a localização dos processos de forma a encontrar os objetivos de performance e confiabilidade para o sistema assim configurado.

2.2.1 Camadas de Software

O termo arquitetura de software referia-se originalmente a estruturação do software como camadas ou módulos em um único computador e mais recentemente em termos de serviço oferecido e requisitado entre os processos localizados no mesmo ou em diferentes computadores. Esta visão de orientação a processo e a serviço pode ser expressada em termos de camadas de serviço. Esta visão pode ser vista na figura 2.1

Figura 2.1 Camadas de Serviço

Podemos definir o conceito de plataforma e *middleware* como segue:

- **Plataforma** – Os níveis mais baixos camadas de hardware e software são freqüentemente referidos como a plataforma para sistemas distribuídos e aplicações. Provêem serviços para as camadas acima, que são implementadas independentemente em cada computador. Como exemplos, Intel x86/Windows, Sun SPARC/Sun OS, Intel x86/Solaris, Powerpc/Maços, Intel x86/Linux.
- ***Middleware*** – É uma camada de software cujo propósito é criar uma mascara para a heterogeneidade e prover um modelo de programação conveniente para os programadores de aplicações. *Middleware* é representado por processos e objetos em um conjunto de computadores que interagem entre si para implementar suporte a comunicação e compartilhamento de recursos para aplicações distribuídas. Em particular, alcança o nível das atividades de comunicação dos programas de aplicação através do suporte a abstrações tais como chamada de métodos remotos (RMI), comunicação entre grupos de processos, notificação de eventos, replicação

de dados compartilhados e transmissão de dados multimidea em tempo real. Como ambientes mais empregados atualmente, podemos citar a chamada remota de procedimentos Sun RPC e o sistema de comunicação em grupo ISIS. Produtos e padrões orientados a objeto, incluem CORBA Common Object Request Broker Architecture, Java RMI Java Remote Object Invocation , o padrão Microsoft DCOM Distributed Component Object Model e ISO-ITU modelo de referência para processamento distribuído (RM-ODP).

2.2.2 Arquitetura de Sistemas

A divisão de responsabilidade entre os componentes do sistema (aplicação, servidor e outros processos) e a localização dos componentes nos computadores na rede é talvez o mais evidente aspecto do projeto de sistemas distribuídos. Tem a maior implicação para performance, confiabilidade e segurança do sistema resultante. A seguir vemos os principais modelos de arquitetura nos quais esta distribuição de responsabilidades é baseada:

- **Modelo Cliente-Servidor** – Esta é a arquitetura que é mais freqüentemente citada quando discutimos sistemas distribuídos. A figura 2.2 ilustra a estrutura simples na qual o processo cliente interage com processos servidores individuais em computadores separados de forma a acessar os recursos compartilhados que o servidor gerencia. Um servidor pode por sua vez ser cliente de outro servidor. Por exemplo um servidor web é freqüentemente cliente de um servidor de arquivos onde as paginas estão armazenadas. Servidores web e outros serviços na internet são clientes do serviço DNS que traduz os nomes de domínio internet em endereços de rede.

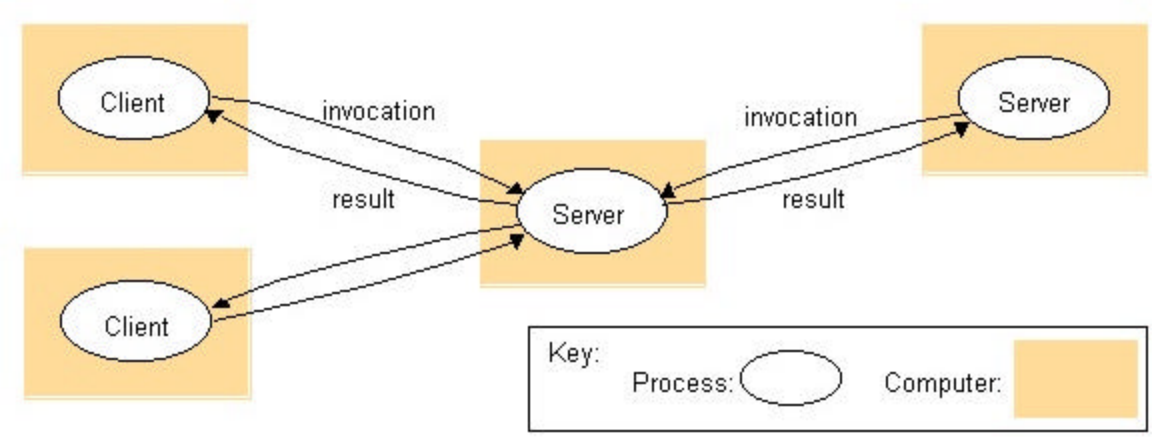

Figura 2.2 Clientes chamando servidor individual

- **Serviços atendidos por múltiplos Servidores** – Serviços podem ser implementados como diversos processos servidores em computadores separados interagindo quando necessário para prover um serviço para um processo cliente. Na figura 2.3 o servidor pode particionar o conjunto de objetos no qual o serviço é baseado e distribuir eles no próprio servidor ou pode manter cópias replicadas deles nos diversos servidores em diferentes maquinas. Como exemplo a web mostra uma situação onde o particionamento de dados existe e cada servidor web controla seu próprio conjunto de recursos. A replicação é usada para aumentar a performance, disponibilidade e aumentar a tolerância a falha. Assim provê múltiplas cópias consistentes dos dados em processos rodando em diferentes computadores. Exemplo as consultas no altavista são mapeadas em diversos servidores que tem o banco de dados replicado em memória.

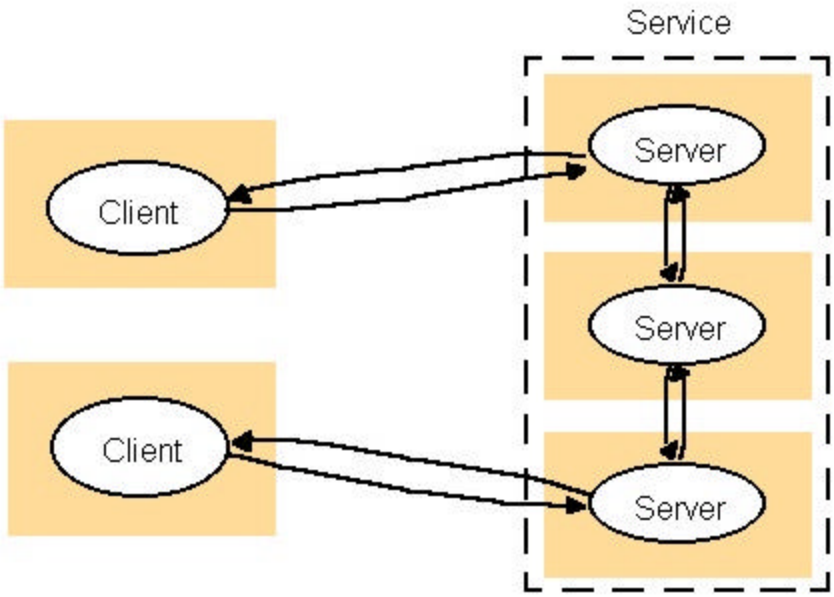

Figura 2.3 Serviço atendido por múltiplos servidores

- **Servidores Caches e Proxy** – Cache é o armazenamento de objetos de dados recentemente usados que estão mais perto do que o próprio objeto de dado. Quando um novo objeto de dado é recebido ele é armazenado no disco do servidor cache, substituindo algum antigo se necessário. Quando um objeto é solicitado pelo cliente primeiro é verificado se existe uma cópia atualizada no cache. Cache pode ser colocado em cada cliente ou em um servidor proxy que pode ser compartilhado para diversos clientes. O propósito do servidor proxy é aumentar a disponibilidade e performance do serviço reduzindo a carga no trafego da rede e no servidor web. Servidores proxy podem também ser usados para acessar servidores web remotos através do firewall.

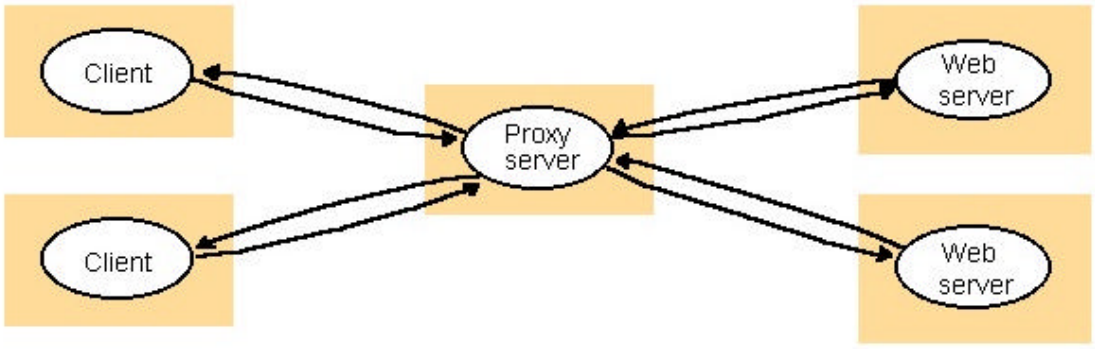

Figura 2.4 Servidor Proxy

- **Processos *Peer*** – nesta arquitetura todos os processos executam regras similares, interagindo cooperativamente como pares para executar uma atividade ou computação distribuída sem nenhuma distinção ente cliente e servidor. Neste modelo a programação no processo *peer* mantém a consistência do recursos a nível de aplicação e sincroniza as ações a nível de aplicação quando necessário. A figura 2.5 mostra uma instância com três processos. De forma geral n processos *peer* podem interagir entre si e o padrão de comunicação vai depender dos requisitos da aplicação.

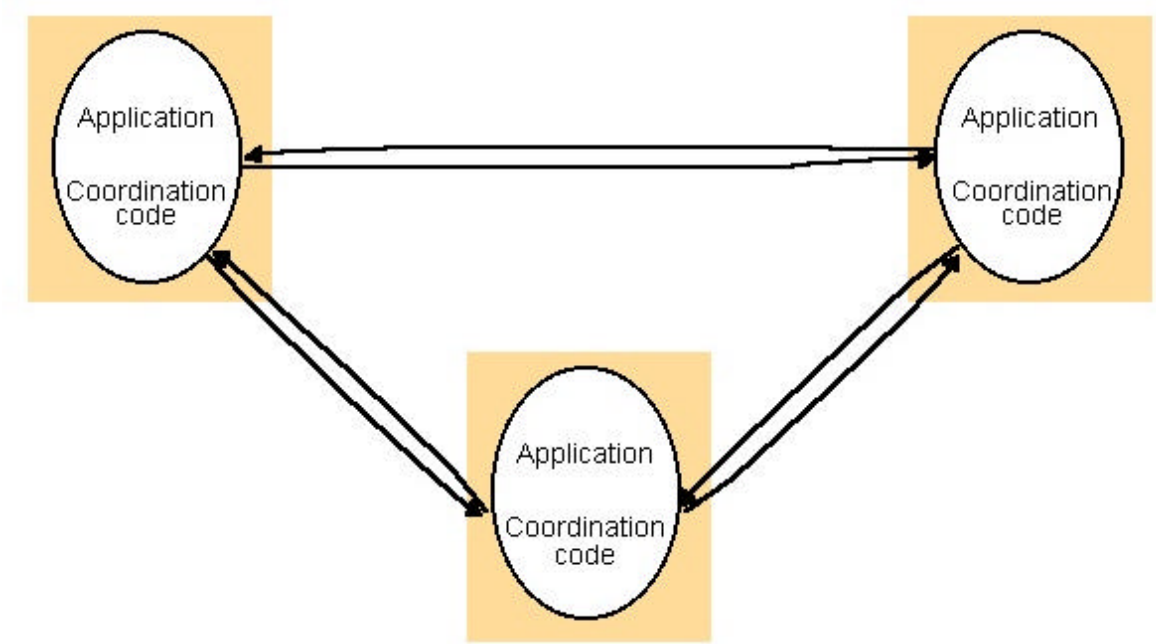

Figura 2.5 Aplicação distribuída baseada em processos par (*Peer Processes*)

2.2.3 Variação do modelo Cliente-Servidor

Diversas variações no modelo cliente servidor podem ocorrer se considerarmos os seguintes fatores:

- O uso de códigos moveis e agentes moveis
- Necessidades do usuário em ter computadores de baixo custo com recursos limitados de hardware e que sejam simples de gerenciar.
- A necessidade de adicionar e remover dispositivos móveis de forma conveniente.

Códigos móveis – Applets são exemplos bem conhecidos e largamente usados de codgo móvel. O usuário executando um browser seleciona um link para um applet cujo código está armazenado no servidor, este código é recebido no browser e roda nele, como visto na figura 2.6. Uma vantagem de rodar o código localmente que foi baixado do servidor é que pode ter um bom tempo de resposta interativa pois não depende mais dos atrasos e variações na velocidade da rede de comunicação.

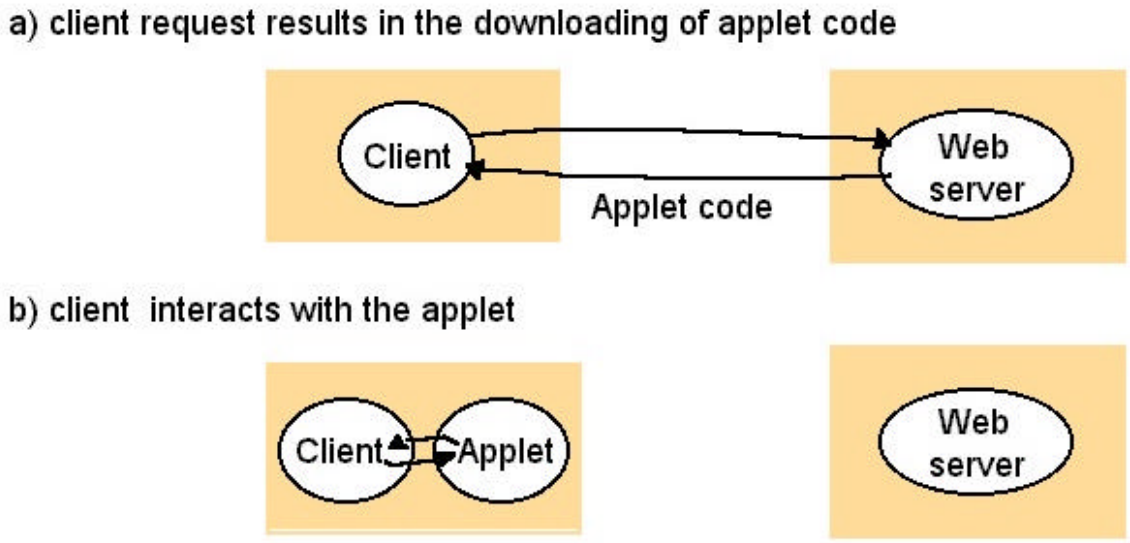

Figura 2.6 Applet Web

Agentes móveis – É um programa executando (incluindo tanto código como dados) que viaja de um computador para outro na rede carregando uma tarefa para quem possa executar de favor, tal como coleta de informação, eventualmente retornando com o resultado. Um agente móvel pode fazer muitos acesso a um recurso local em cada site que ele visita, por exemplo acessando um banco de dados. Se compararmos esta arquitetura com a estática cliente-servidor fazendo chamadas remotas a algum recurso, possivelmente transferindo grande quantidade de dados, existe uma redução no custo de comunicação e tempo se substituímos a chamada remota por processamento local do agente.

Computadores em rede – É a arquitetura mais conhecida onde as aplicações rodam em um computador de mesa junto ao usuário. O sistema operacional e os programas aplicativos geralmente estão no disco local. A aplicações rodam localmente mas os dados são gerenciados por um servidor de arquivos remoto.

Cliente magro – O termo cliente magro (thin client) refere-se a camada de software que suporta um interface para o usuário baseado em janelas no computador que é local enquanto a execução da aplicação é feita em um computador remoto (Figura 2.7). Semelhante aos computadores em rede, porém toda a execução é feita em grandes servidores, eventualmente cluster, geralmente multiprocessados, rodando sistemas como Unix ou Windows NT. Uma dificuldade desta arquitetura é quando temos alta atividade de interação gráfica como CAD e processamento de imagem onde as demoras experimentadas pelos usuários aumentam com a necessidade de transferir imagens e informações vetoriais entre o cliente magro e o processo da aplicação. O conceito de clientes magros é simples e sua implementação em alguns é direcionada como no Unix, usando o X-11, Xwindows padrão neste ambiente. Para ambiente Windows Microsoft um produto chamado Win Frame da Citrix (www.citrix.com) é largamente empregado. Outra implementação importante que existe para muitas plataformas de hardware é o VNC – Teleporting and Virtual Network Computer desenvolvido no laboratório AT&T (www.uk.research.att.com/vnc).

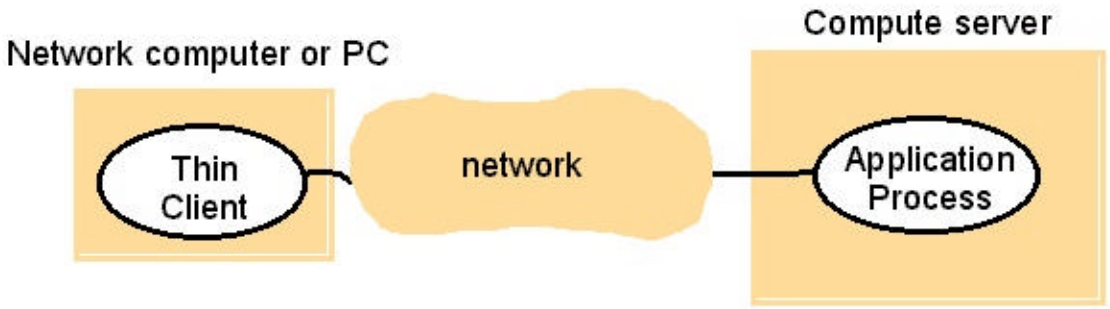

Figura 2.7 Clientes magros e servidores de computação

Dispositivos móveis e redes espontâneas – O grande aumento de dispositivos móveis e portáteis como laptops, dispositivos de mão como assistentes digitais pessoais (PDA) telefones móveis e câmaras digitais, computadores que podemos vestir, como relógios avançados e dispositivos com sistema dedicado tais como maquina de lavar, formam um novo contexto a ser estudado. Muitos deles tem capacidade de se interligar via rede sem fio, com possibilidade de comunicação a distância metropolitana (GSM, CDPD), a distâncai de centenas de metros (WaveLAN), ou poucos metros (BlueTooth, Infravermelho e HomeRF). As redes de curta distância tem largura de banda da ordem de 10 megabits/segundo: GSM promete ser da ordem de centenas de kilobits/segundo. Com apropriada integração em nosso sistema distribuído estes dispositivos provêem suporte para computação móvel. A forma de distribuição que integra dispositivos móveis e outros dispositivos em uma determinada rede é talvez a melhor descrição para o termo redes espontâneas. Um exemplo de rede espontânea, hipotética para um hotel é vista na figura 2.8. Assim precisamos ter facilidade de conexão a redes locais e facilidade de integração com os serviços locais quando um deste dispositivos móveis tiver a sua presença detectada em uma ambiente integrado. Devemos também lembrar que para usuários móveis temos conectividade limitada, em virtude de possíveis interrupções (passando por um túnel), alem de preocupação com segurança e privacidade. Também uma forma de descobrir os serviços disponíveis neste ambiente é o ideal para quando o cliente móvel se integrar a rede poder registrar se em um serviço.

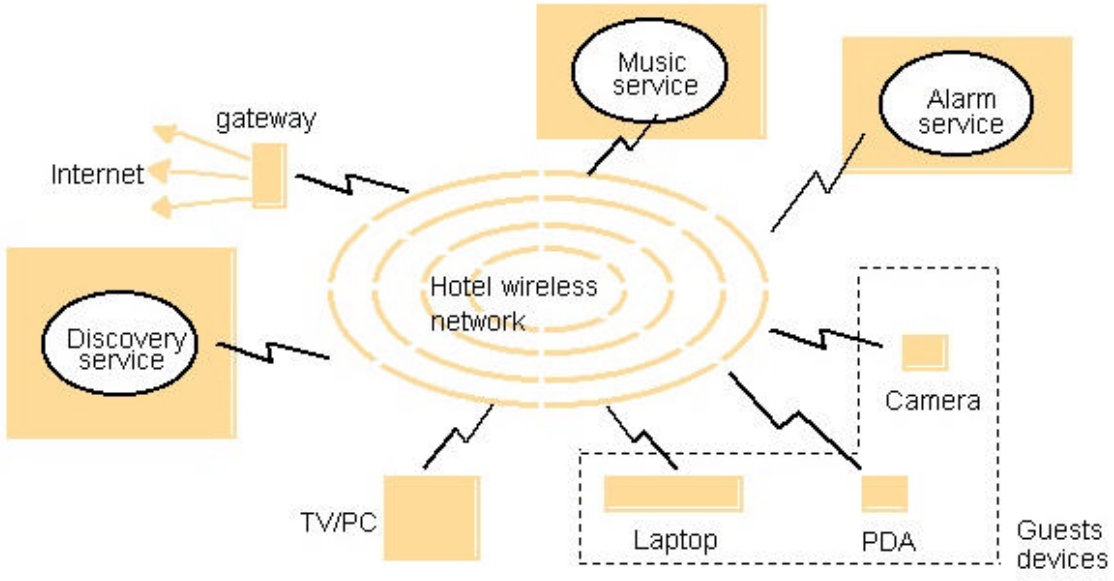

Figura 2.8 Rede espontânea em um Hotel

2.3 Modelos Fundamentais

Nesta seção serão apresentados modelos baseados nas propriedades fundamentais que nos permitem ser mais específico sobre suas características as falhas e os riscos de segurança. De forma geral um modelo contém somente os ingredientes essenciais que nos necessitamos para considerar e entender a razão de determinados comportamentos nos sistemas. Um modelo de sistemas tem que tratar os seguintes pontos:

- Quais são as principais entidades dos sistemas?
- Como elas interagem?
- Quais são as características que afeta seu comportamento individual e coletivo?

O propósito de um modelo é:

- Tornar explicito todas as suposições relevantes sobre o sistema que estamos modelando.
- Fazer generalizações relativamente ao que é possível ou impossível mostrando estas suposições. A generalização pode tomar a forma de algoritmos de propósito geral, ou propriedades desejadas com existência garantida. As garantias são dependentes de analise lógica e onde apropriado, de prova matemática.

Os aspectos de sistemas distribuídos que queremos captar em nossos modelos fundamentais têm a intenção de nos ajudar a discutir e ver as razões sobre Interação, Falhas e Segurança.

2.3.1 Modelos de Interação

Sistemas distribuídos são compostos de muitos processos interagindo entre si de formas complexas. Múltiplos servidores podem cooperar entre si para prover um serviço. Um conjunto de processos pares pode cooperar entre si para atingir um objetivo comum. Muitos programadores estão familiarizados com o conceito de algoritmo – uma seqüência de passos que devem ser tomados para que se execute uma determinada computação. Programas únicos são controlados por algoritmos onde os passos são estritamente seqüenciais. Sistemas distribuídos compõem-se de múltiplos processos e o comportamento e estado podem ser descritos por um algoritmo distribuído – uma definição dos passos a serem seguidos por cada processo do qual o sistema é composto incluindo a transmissão de mensagens entre os processos. Nesta seção vamos discutir dois fatores significantes que afetam a interação de processos em um sistema distribuído:

- **Performance na comunicação** – A comunicação em nosso modelo se realiza de diferentes formas, por exemplo pela implementação de fluxos, ou pela passagem de mensagens através da rede. Na comunicação através da rede temos preocupações como:
  - Latência – A diferença de tempo entre o início de uma transmissão em um processo e o início da recepção da mensagem em outro processo.
  - Largura de banda – Quantidade de informação que passa por unidade de tempo.
  - *Jjitter* – Variação de tempo para entregar uma serie de mensagens. Mais relevante em dados multimídia.
- **Impossibilidade de manter sistema único de relógio** – Cada computador em um sistema distribuído tem o seu relógio próprio. O controle dos eventos nos diferentes processos pode ser feito com tempo associado ao evento. Nem sempre os relógios dos diferentes sistemas marcam um tempo único. Mesmo que todos os computadores acertassem o seu relógio com base em um relógio padrão, com o decorrer do tempo pequenas variações podem ocorrer pois não existe uma precisão

exata no relógio de cada computador. Nos sistemas distribuídos é difícil de determinar limites de tempo para a execução de um processo, entrega da mensagem ou diferença ocorrida no relógio do computador. Duas posições extremas podem ser adotadas para este modelo:

- **Sistemas Distribuídos Síncronos** – Todas as operações, processamento e transmissão de mensagens tem limites de tempos conhecidos e são controladas por tempo. Podemos detectar falhas devido a intervalos de tempo estabelecidos para determinadas operações.
- **Sistemas Distribuídos Assíncronos –** Alguns sistemas distribuídos, como a internet, não tem limites bem definidos de tempo para velocidade de execução e tempos de transmissão das mensagens.

Em alguns casos precisamos saber se um evento (envio ou recebimento de mensagens) em um processo ocorre antes, depois ou concorrentemente com outro evento em outro processo. A execução do sistema pode ser descrita em termos dos eventos e sua ordenação independente da falta de relógios precisos. A figura 2.9 mostra um exemplo desta ordenação de eventos. Temos quatro usuários de uma lista de email, X, Y, Z e A. X manda uma mensagem, Y lê esta mensagem e responde, Z recebe a mensagem e responde porem A recebe fora de ordem, primeiro a resposta de Z, depois a mensagem de X e por fim a resposta de Y. Esta preocupação de ordenamento de mensagens é muito comum e necessária em comunicação de grupo.

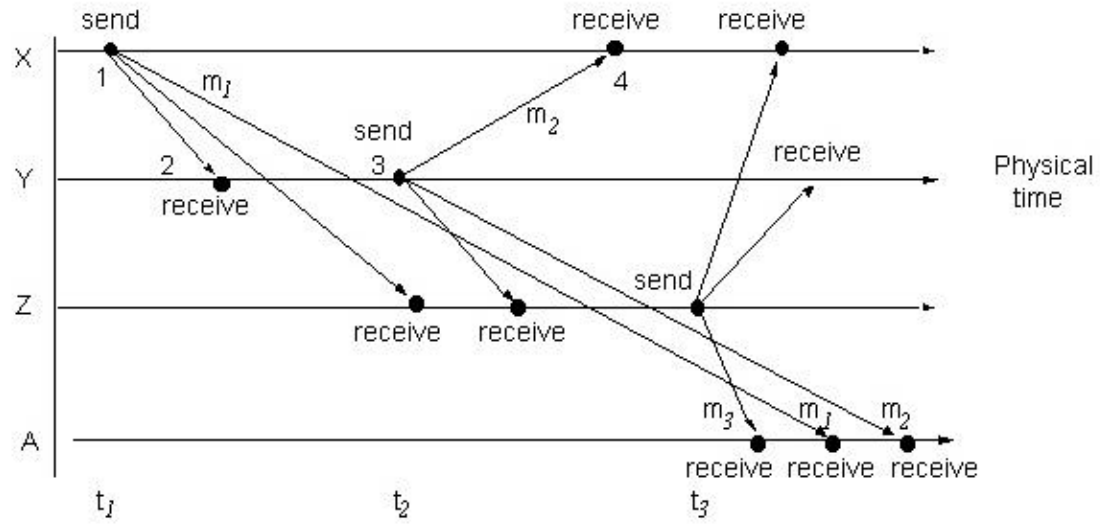

Figura 2.9 Ordenação de eventos em tempo real

2.3.2 Modelo de Falhas

Em um sistema distribuído, tanto os processos como os canais de comunicação podem falhar. O modelo de falhas define as formas como podem ocorrer as falhas de maneira a entendermos os efeitos das falhas. Podemos ter:

- **Falhas de omissão** – São Falhas nos casos onde um processo ou canal de comunicação falha na execução da ação que deveria fazer.
- **Falhas arbitrárias** – O termo falha arbitraria é empregado para descrever o pior falha de semântica, na qual quelaer tipo de erro pode ocorrer. Por exemplo um processo pode colocar valores errados em seus itens de dados, ou podem retornar valores errados em resposta a uma chamada. Canais de comunicação podem sofrer falhas arbitrarias. Por exemplo o conteúdo de uma mensagem pode ser corrompido, uma mensagem não existente pode ser entregue ou uma mensagem real pode ser entregue mais de uma vez.
- **Falhas de tempo** – Falhas de tempo são aplicáveis a sistemas distribuídos síncronos onde limites de tempo são estabelecidos para tempo de execução do processo, tempo de entrega da mensagem e taxa de erro do relógio é conhecida.

2.3.3 Modelo de Segurança

A segurança de um sistema distribuído pode ser alcançada se temos segurança nos processos e nos canais de comunicação usados para sua interação e protegendo os objetos que eles encapsulam contra acesso não autorizado. Assim um cliente que acessa um servidor é necessário que o objeto acessado possa proteger seus dados privados e o acesso a informações compartilhadas deve ocorrer se existir o direito de acesso para este cliente.

Quanto aos canais de comunicação, temos que ver a segurança dos mesmos em termos de ataques pois sendo uma rede aberta podemos ter ataques externos de usuários malfeitores. Temos ameaças aos processos, ameaça aos canais de comunicação e sobrecarga de uso do recurso. Podemos ter copia de mensagens, inserção de mensagens falsas e ataques de volume de requisição exagerada. Um forma de proteger os canais de comunicação é com o uso de criptografia e autenticação de acesso.

Itajaí , 11 de março de 2002

Tradução (resumida) do Capitulo 2
Livro DISTRIBUTED SYSTEMS Concepts and Design
George Coulouris, Jean Dollimore, Tim Kindberg
Por
Ademir Goulart