Sumário da Lição de Síntese sobre Comunicação e Computação em Sistemas Distribuídos

Paulo Jorge E. Veríssimo
paulov@inesc.pt
DEEC - IST

1 Introdução

Esta lição aborda uma temática extremamente actual, que é conjunto de conceitos, técnicas, algoritmos e protocolos, que possibilita a correcta operação dos sistemas distribuídos. Isto é: como os processos comunicam entre si; como executam computações distribuídas; como o fazem de modo fiável (tolerância a faltas) e pontual (tempo-real).

Uma síntese sobre um tema tão abrangente como o que nos propomos abordar, num tempo necessariamente limitado, lança um desafio importante. No intuito de dar uma visão tão completa quanto possível da problemática que constitui o tema da lição, escolhemos abordar a generalidade dos assuntos de uma forma que, sendo resumida, não deixa de levantar as questões fulcrais de cada um. Em complemento, procuramos suscitar o interesse do leitor/ouvinte, pela descida, de quando em vez, aos pormenores de uma técnica ou protocolo bem exemplificativos da discussão. Esses pormenores serão assinalados com o título *Tecnologia Seleccionada*, para marcar claramente a mudança de nível de abstracção.

Sumário

1. A problemática dos sistemas distribuídos
2. Confiabilidade
3. Tempo-Real
4. Redes e Comunicação entre Processos
5. Projecto e Realização de Protocolos
6. Computação Distribuída: estilos de programação
7. Programação orientada para grupos
8. Programação orientada para Transacções
9. Programação orientada para Memória Partilhada Distribuída
10. Segurança em Sistemas Distribuídos

2 O que é um sistema distribuído?

> "Um sistema distribuído é aquele que nos impede de trabalhar devido à falha de uma máquina de que nem sequer ouvimos falar." [L. Lamport]

A citação com que começamos esta lição vem de um dos formalistas que mais contribuiram para o avanço do conhecimento dos sistemas distribuídos. Ilustra, talvez melhor que qualquer definição formal, a verdadeira problemática destes sistemas:

- as máquinas que compõem um sistema distribuído falham independentemente;
- no entanto, influenciam-se mutuamente, quer quando funcionam, quer, por vezes, quando falham;
- comunicam através de redes não fiáveis e com demoras pouco previsíveis;
- em consequência, um utilizador tem dificuldade em se aperceber do seu estado global.

Por isso, não será surpreendente que o simples facto de "juntar" máquinas agrave os problemas da sua operação isolada. Os sistemas distribuídos são por excelência aqueles em que deve prevalecer uma óptica sistémica, a abordagem pela "floresta" e não pela "árvore".

Essa abordagem, que vamos sintetizar nesta lição, significa que, face a uma série de *requisitos dos utilizadores*, só ultrapassando os problemas de *confiabilidade* e *pontualidade* causados pela *distribuição*, teremos o potencial para satisfazer cabalmente os primeiros.

Características de um Sistema Distribuído

Não existe uma única definição de sistema distribuído. Um sistema distribuído caracteriza-se mais por um conjunto de sintomas, e pelas consequências desses sintomas. Os sintomas "típicos" de um sistema distribuído são:

- Múltiplos computadores
- Interligação em rede
- Estado Partilhado

Podemos assim identificar uma série de vantagens e problemas inerentes aos sistemas que apresentam os sintomas acima indicados:

Falha independente - as máquinas que constituem um sistema distribuído falham, por pressuposto, independentemente; esta constitui uma vantagem que devemos saber aproveitar, não permitindo, por outro lado, que essa falha afecte as restantes máquinas.

Comunicação não fiável - a comunicação entre máquinas (chamemos-lhes *nós* a partir de agora) efectua-se sobre redes que podem perder mensagens, podem partir-se, ou podem parar (parcial ou totalmente).

Atrasos variáveis - essas redes, ao contrário, por exemplo, da propagação da luz no universo, não têm velocidade constante.

Lentidão, Largura de banda - comparavelmente à velocidade de evolução dos programas e processos (chamemos-lhes genericamente *participantes* a partir de agora), a comunicação através das redes é lenta e tem débito baixo.

Preço - o chamado *downsizing* possibilitado pelas tecnologias de sistemas distribuídos permite que o preço de um conjunto de pequenas máquinas interligadas seja normalmente inferior ao de uma grande *mainframe* com igual poder de cálculo.

Insegurança - um sistema cuja informação passa por cabos e outros meios de propagação sujeitos ao escrutínio de intrusos, é potencialmente inseguro.

Inexactidão do estado global - num sistema onde os acontecimentos se dão mais rapidamente que a comunicação entre os participantes, é impossível obter um estado global exacto a partir de qualquer nó em todas as circunstâncias.

Ordem parcial de eventos - dada a dualidade estado-evento, o problema da inexactidão do estado global tem a sua correspondência em termos dos eventos que modificam esse estado. O problema pode-se exemplificar com a situação que ocorreria com dois postos tentando dialogar por rádio a partir de dois planetas afastados do nosso sistema solar: a consequência mais óbvia é aquilo que se denomina uma ordem parcial de eventos, isto é, a ordem relativa das mensagens de ambos, observada por cada posto, seria diferente, sem que isso perturbe o diálogo.

Sistemas centralizados *vs.* distribuídos

Comparativamente aos sistemas centralizados, os sistemas distribuídos apresentam vantagens e desvantagens.

Os sistemas centralizados possuem maior *acessibilidade* e, por definição, apresentam um ambiente *homogéneo* ao(s) utilizador(es). Por estrutura, e também por essa homogeneidade, são de *fácil gestão*. Nos sistemas distribuídos, a *heterogeneidade* e o *âmbito geográfico* põem problemas à gestão dos mesmos. No entanto, são trunfos principais destes sistemas, aliados à possibilidade de *partilha* de informação entre vários lugares.

Um sistema distribuído possui uma *modularidade* inerente (as máquinas individuais), a qual, associada à heterogeneidade e ao âmbito geográfico, permite uma *escalabilidade* que não está ao alcance de máquinas centralizadas. Estas características permitem a concepção de sistemas complexos e potentes a *custo reduzido*, se comparado com a tradicional aproximação *mainframe*.

Num sistema centralizado a *congruência* da informação em operação normal é naturalmente assegurada, uma vez que se trata de uma única máquina. Num sistema distribuído, temos os problemas da *inexactidão do estado global* e da *ordem parcial de eventos*, sobre cuja resolução se tem publicado extensa literatura.

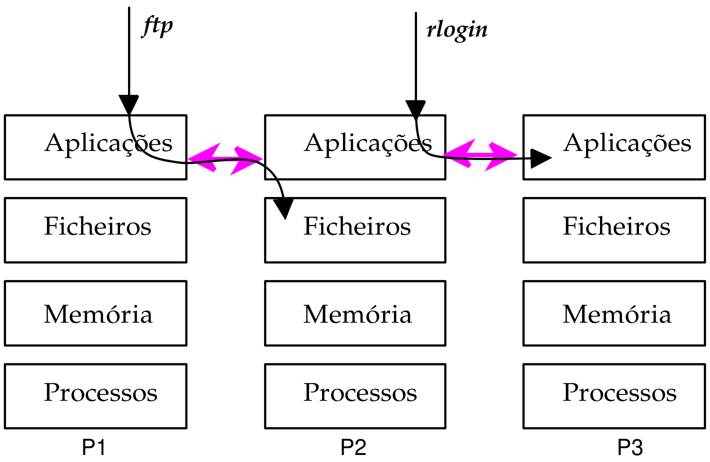
No ponto de vista de confiabilidade, uma vantagem importante dos sistemas distribuídos é que, se propriamente projectados e dimensionados, apresentam *decaimento gradual (graceful degradation)*, isto é, perante a falha ou falhas sucessivas de componentes, continuam a funcionar num modo degradado de operação, sem no entanto falhar abruptamente à primeira falha, como é normalmente o comportamento de um sistema centralizado (simples-ponto-de-falha ou *hard-core*). Sobre a *segurança*, duas questões são relevantes: enquanto que é certo que um sistema distribuído é potencialmente inseguro, não o é menos que, uma vez "quebrado", um sistema centralizado está totalmente à mercê do intruso. Alternativamente, *sobre* um sistema distribuído, podemos concretizar políticas de segurança que impliquem ao intruso "quebrar" mais de uma máquina para poder aceder ou controlar informação significativa. Sobre a *disponibilidade*, torna-se óbvio que o sindroma simples-ponto-de-falha dos sistemas centralizados pode ser anulado, com grande vantagem para os utilizadores, tirando partindo de existirem várias máquinas no sistema. Os principais mecanismos para conseguir confiabilidade, ou *tolerância a faltas*, baseiam-se em *migração* e *replicação*.

Evolução dos Sistemas Distribuídos

As tecnologias dos sistemas distribuídos têm sofrido uma evolução de um modo geral lógica, pautada pelas necessidades dos utilizadores.

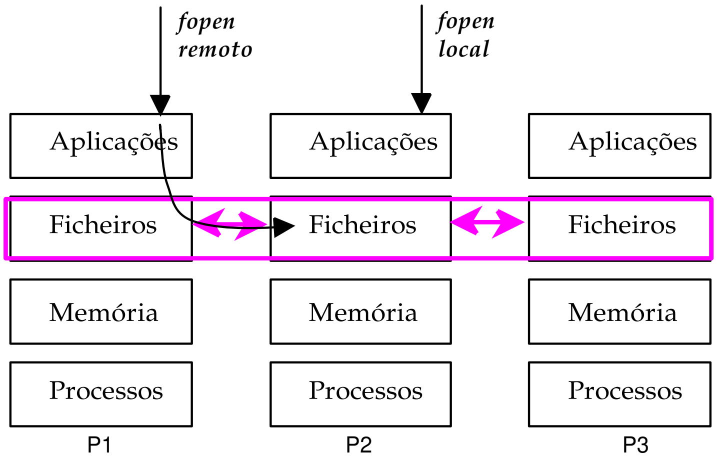
Assim, face ao aparecimento do processamento interactivo local, a primeira geração visou a **partilha de recursos**, materializada por dois mecanismos:

- sessão remota *(rlogin)*
- transferência de ficheiros



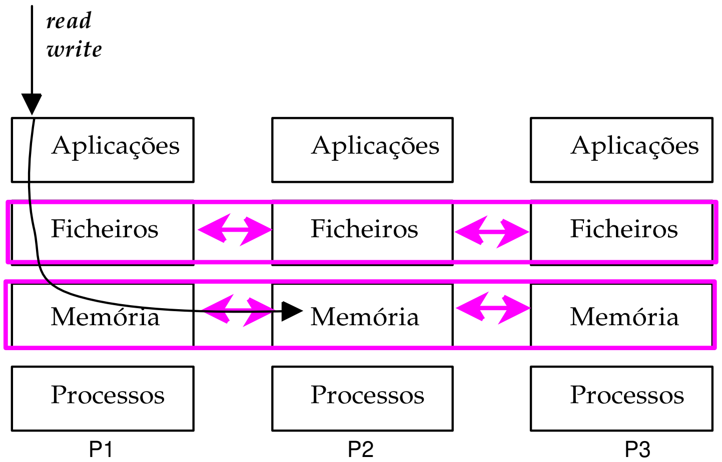
Na óptica dos sistemas operativos baseados em disco, a partilha eficaz de informação passava por evitar quando possível a transferência explícita de ficheiros. Nasceram assim os *sistemas de ficheiros distribuídos*:

- evitar a sessão remota e transferência de ficheiros
- partilha eficaz de informação



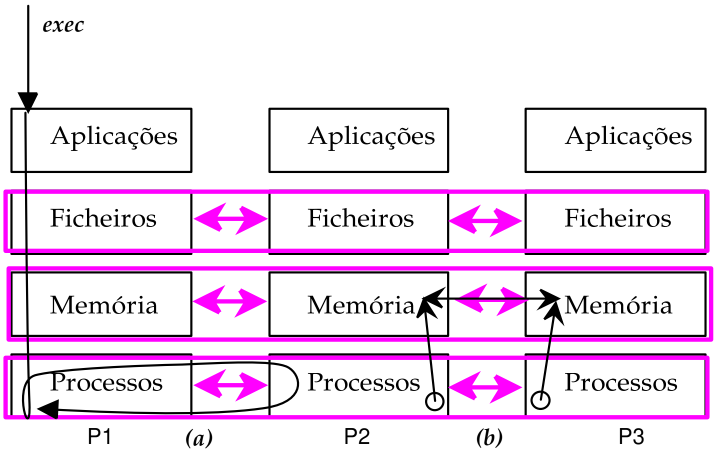
Os sistemas distribuídos evoluiram de diversas formas, mas as redes de estações de trabalho sobre redes locais em difusão e de débito elevado (10Mb/s) tornaram-se extremamente vulgares. Com tal infraestrutura, o próximo objectivo é beneficiar dos restantes recursos de uma rede de máquinas, como por exemplo memória e poder de cálculo. No sentido de poder suportar programação concorrente em diferentes máquinas, e assim abrir caminho ao processamento distribuído, surgiram sistemas com **memória partilhada distribuída** (DSM):

- beneficiar dos recursos de uma rede de máquinas
- suportar memória partilhada



Com esta abstracção, que visa dar a programas concorrentes que corram em diversas máquinas, a ilusão de verem a "mesma" memória partilhada, torna-se possível suportar actividades concorrentes sincronizadas sobre essa memória distribuída. Outro modelo consiste em ter actividades que passam de máquina para máquina, normalmente associado a um estilo de programação com chamada a procedimento remoto. Estas são as duas modalidades mais importantes de **processamento distribuído**:

- actividades concorrentes sobre DSM
- actividades que passam de
  máquina para máquina



Perante este panorama inicial, é lógico concluir que os factores-chave na evolução dos sistemas distribuídos serão aqueles que permitam tornear os problemas postos por "juntar máquinas":

- **falhas, demoras, incongruência,**

e levem à obtenção de propriedades úteis ao seu funcionamento:

- **fiabilidade, disponibilidade, segurança, animação, pontualidade, concorrência, paralelismo, partilha (dados, carga, funções).**

O domínio das técnicas conducentes a essas propriedades, isto é, a construção e operação correcta e eficiente de sistemas distribuídos – cada vez mais complexos – pertence a três campos, que a evolução tem aproximado:

- distribuição;
- tolerância a faltas;
- tempo-real.

O objectivo da lição de síntese é percorrer estes aspectos sistémicos, sob as seguintes perspectivas:

1. como os processos comunicam entre si;
2. como executam computações distribuídas;
3. como fazem 1 e 2 de modo fiável (tolerância a faltas) e pontual (tempo-real).

3 Confiabilidade

Confiabilidade de um sistema é o conjunto de propriedades do mesmo, ou *qualidade de serviço*, que permitem que se tenha, justificadamente, confiança no seu funcionamento.

O objectivo da confiabilidade é evitar falhas. A **falha** (*failure*) do sistema ocorre quando o serviço fornecido se desvia das condições mencionadas na especificação do mesmo.

A falha ocorre porque o sistema está num estado incorrecto, isto é, existe um **erro** (*error*), que pode conduzir à falha.

Finalmente, a causa de um erro – no seu sentido fenomenológico – é uma **falta** (*fault*). Uma **falta** pode existir muito antes de produzir efeitos: diz-se inactiva. Quando de algum modo é activada, pode produzir um erro que levará à falha. Ao leitor poderá espantar porquê três designações para fenómenos que no entendimento popular serão comumente "falhas". Para melhor compreensão da importância desta distinção, eis um exemplo concreto:

- Falta: bit agarrado a 'zero' numa RAM.
- Erro: escrita de ùm', seguida de leitura.
- Falha: bit errado ('zero') entregue ao utilizador.

A *falta* existe, e no entanto, pode nunca vir a originar uma falha! Agora suponha que a memória é do tipo ECC (*error checking and correcting*): o *erro* no bit, mal saia da memória, é detectado, e corrigido antes de ser enviado ao programa utilizador e ocasionar uma *falha*. Isto é um exemplo de **tolerância a faltas**. Tolerância a faltas baseia-se em prover o sistema de redundância, nomeadamente, replicação. Se bem que seja o mecanismo mais eficaz para obter confiabilidade, pois age durante a operação do sistema, existem outros meios. Genericamente:

- Supressão de faltas: por exemplo por depuração, simulação, ou validação.
- Previsão de faltas: por exemplo por injecção de faltas em software ou hardware.
- Prevenção de faltas: por exemplo por auto-teste, assinaturas.
- Tolerância a faltas: por exemplo através de replicação, votadores, transacções, assinaturas e *checkpoints*, temporização/repetição.

Existem dois grupos fundamentais de técnicas de processamento de erros para obter tolerância a faltas:

- Mascaramento de erros (*error masking*)
- Detecção/Recuperação (*detection/recovery*)

A quantificação da confiabilidade faz-se através de medidas, que são:

- **Fiabilidade (*reliability*):** medida do tempo de funcionamento de um sistema até falhar, ou da probabilidade do mesmo não falhar durante o tempo de missão (ex. MTTF, MTBF, 10^{-5} falhas/hora, 99.9%).

CONFIABILIDADE

- **IMPEDIMENTOS**
  - *FALTAS*
  - *ERROS*
  - *FALHAS*
- **MEIOS**
  - *OBTENÇÃO*
    - *PREVENÇÃO de FALTAS*
    - *TOLERÂNCIA a*
  - *VALIDAÇÃO*
    - *SUPRESSÃO*
    - *PREVISÃO*
- **MÉTRICAS**
  - *FIABILIDADE*
  - *REPARABILIDADE*
  - *DISPONIBILIDADE*
  - *SEGURANÇA DE FUNC.*
  - *SEGURANÇA DE DADOS*

Figura 1: Conceitos de Confiabilidade de Sistemas

- **Reparabilidade (*maintainability*):** medida do tempo de reposição em serviço de um sistema (ex. MTTR).
- **Disponibilidade (*availability*):** medida do tempo ou da percentagem deste, em que um sistema está operacional, durante um intervalo de missão (ex. MTBF/(MTBF+MTTR), 5000h/ano).
- **Segurança (*security, safety*):** segurança de dados, *security* – medida da fiabilidade do sistema com respeito a faltas específicas contra: integridade, confidencialidade, autenticidade; segurança de operação, *safety* – idem, em relação a faltas que ocasionem efeitos catastróficos.

Estes conceitos estão globalmente representados na figura 1.

4 Tempo-Real

Tempo-real é a qualidade de um sistema que lhe permite reagir atempadamente, isto é, com pontualidade, a estímulos do ambiente.

Alguns preconceitos sobre sistemas tempo-real têm persistido ao longo dos anos, levando quer à minimização dos problemas causados em sistemas supostamente não tempo-real, quer ao projecto incorrecto de sistemas supostamente tempo-real. Vamos comentar alguns exemplos desses preconceitos:

- *Tempo-real é rapidez, engenharia do desempenho; o avanço tecnológico (mais MIPS do CPU, mais MBaud da rede) é a solução básica.*

O tempo-real é acima de tudo *determinismo*. Significa agir ou produzir resultados "após X microssegundos", "às Y horas", "dentro de Z milissegundos".

- *Sistemas tempo-real são estáticos, pré-programados (autómatos).*

  Existe uma gama muito importante de sistemas em que as condições de operação não se podem prever com precisão. Esses sistemas têm que ser dinâmicos, adaptativos. Têm que produzir resultados como descrito no parágrafo anterior, mas fazê-lo na presença de factores perturbadores, como sobrecarga ou eventos inesperados.

- *Para quê prazos tempo-real, se o hardware falha, o software é imprevisível, as mensagens atrasam-se, etc.*

  Ignorar o factor confiabilidade num sistema tempo-real é uma hipótese perigosa. Em sistemas críticos, a correcção e robustez do hardware e do software são importantes, mas tal não é suficiente. É necessário prover o sistema com a capacidade de sobreviver a falhas de componentes, isto é, os mecanismos de tolerância a faltas discutidos atrás. Estes mecanismos serão moderados, para aplicações de baixa criticalidade, mas bastante exigentes, em aplicações críticas para a vida, como o controlo de voo ou de centrais nucleares.

Em consequência, o que é importante perceber é que existe hoje uma sistemática bem organizada do Tempo-Real, e para cada classe de sistema, um conjunto de soluções e técnicas adequadas:

- Tempo-Real Estrito *(HARD R/T)* (ex. controlo de trânsito, fabrico por computador, controlo de voo ou de automóvel)
- Melhor-esforço *(BEST-EFFORT)* (ex. sistema de controlo de tráfego aéreo, sistema de controlo de funções vitais de um navio, sistema de comando, controlo e comunicações (C^3))
- Tempo-Real Lato *(SOFT R/T)* (ex. base de dados de reservas de bilhetes, sinópticos, partes do controlo de processos contínuos)

Para conseguir estes objectivos, existem componentes do sistema que têm de observar um determinado comportamento, do ponto de vista temporal:

- comunicação fiável tempo-real – demora de entrega de mensagens limitada;
- escalonamento de processos tempo-real, no sistema operativo local;
- noção de grau de urgência relativa entre acções, e propagação dessa noção a todo o percurso de uma acção, mesmo que distribuída;
- gestão de eventos gerados por faltas temporais (prazos não cumpridos) e tolerância a essas faltas.
- existência de um relógio global.

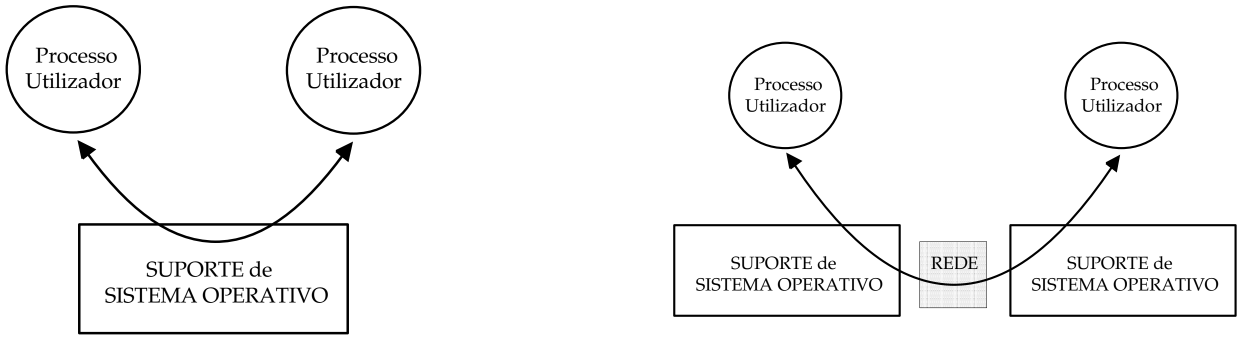


Figura 2: Comunicação entre processos: (a) local; (b) remota

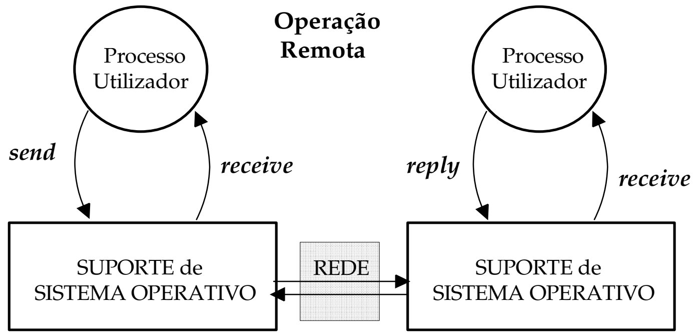


Figura 3: Operação remota ou comunicação pedido resposta

5 Comunicação entre processos (IPC)

Em sistemas distribuídos, a partilha de estado e informação é fundamental. Para haver partilha de qualquer coisa, é necessário comunicação. A comunicação entre processos (IPC) é normalmente suportada pelo sistema operativo (fig. 2a). Se aqueles estão em máquinas diferentes, é necessária uma rede e um serviço de comunicação (fig. 2b).

Esse serviço, dependendo do tipo de sistema, pode ser disponibilizado ao utilizador a diversos níveis de abstracção:

- *envio/recepção* de mensagem;
- *chamada/retorno* de procedimento remoto;
- *invocação* de objecto remoto.

Estas duas últimas formas de comunicação são *operações remotas*, também designadas por **comunicação cliente/servidor** ou *pedido/resposta* (figura 3).

Desempenho, Confiabilidade e Tempo-real na Comunicação

Para um sistema distribuído ser atraente, a comunicação não deve ser um factor de detrimento do desempenho. Por outro lado, e como foi dito atrás, a fiabilidade e

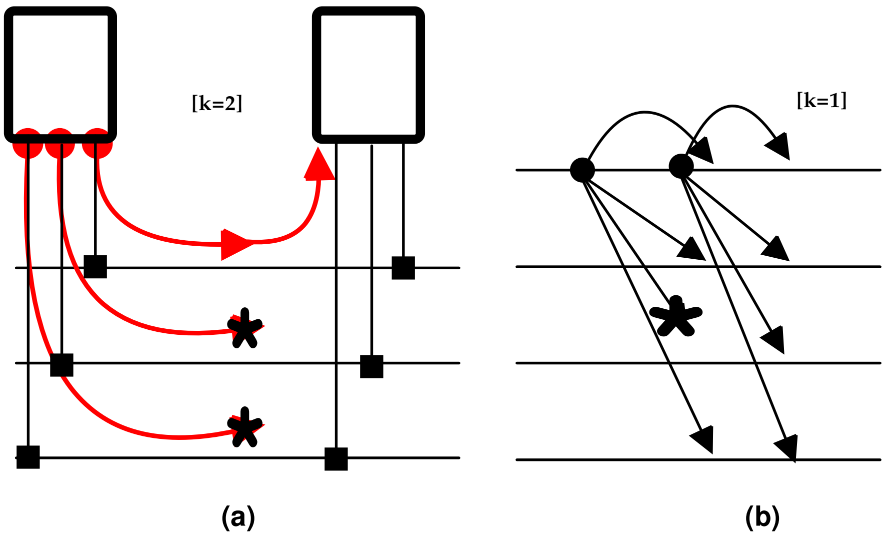


Figura 4: Comunicação confiável com mascaramento: (a) redundância espacial; (b) redundância temporal

a pontualidade desempenham um papel muito importante na eficácia e robustez dos protocolos de um sistema distribuído. Há um conjunto de pontos que se nos afiguram fulcrais para o arquitecto e o projectista de sistemas de comunicação.

Desempenho - os requisitos de desempenho traduzem-se em baixa latência e elevado débito — em consequência, a redução do número de camadas da estratificação (rebatimento), ou da complexidade dos serviços, pode contribuir para o aumento de desempenho. O caminho crítico da execução de um protocolo é a soma dos tempos de execução parciais que intervêm na formação do tempo de execução máximo – intervindo sobre aquele pode baixar-se este tempo.

Confiabilidade - é necessário começar por elaborar o modelo de falhas da rede física — falhas simultâneas? partições? — e traçar a semântica de falha dos protocolos — falhas por omissão? temporais? falhas de valor? — para em seguida, delinear as estratégias para obter fiabilidade — redundância espacial ou temporal? recuperação ou mascaramento? redes fiáveis (redundantes)? segurança por criptografia?

Tempo-real - é necessário começar por definir as hipóteses sobre padrões de tráfego; definir se é tempo-real estrito ou lato; definir classes de urgência de transmissão; e daí chegar aos parâmetros temporais da comunicação (prazo de entrega, pontualidade).

Tecnologia Seleccionada

Estratégias de Confiabilidade em Redes

As estratégias de confiabilidade em redes dividem-se entre:

- **redundância espacial**, que consiste em replicar uma rede, de modo a que a informação passe correctamente em pelo menos uma réplica (figura 4a);

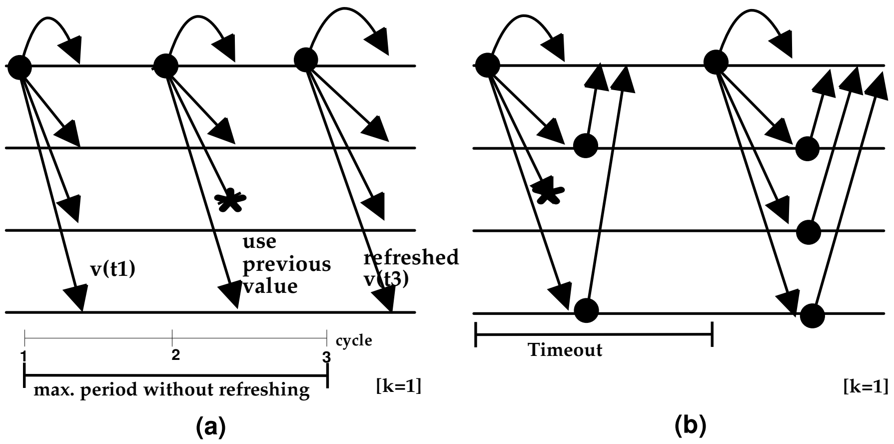


Figura 5: Comunicação confiável:

- **redundância temporal**, para redes *não* replicadas, que consiste em repetir a transmissão de uma mensagem, tantas vezes quantas as necessárias para esta passar (figura 4b).

Existem dois grupos de técnicas de protocolo para a gestão da redundância:

- **mascaramento de erros -** ambas as estratégias exemplificadas na figura 4 se referem a este tipo de técnica, que consiste em não deixar o erro ser aparente; na figura 4b, numa rede "normal" (não replicada), o mascaramento consegue-se repetindo sistematicamente a transmissão, haja ou não erros;

- **detecção e recuperação -** técnica exemplificada na figura 5, consiste em detectar (p.ex. por temporização) a falta de uma mensagem, derivada de um erro de transmissão, e retransmitir a mensagem original (recuperação para trás - fig. 5a) ou em esperar pela próxima mensagem (recuperação para a frente - fig. 5b).

Tecnologia Seleccionada

Classes de comunicação Tempo-real

É importante realçar que existem duas grandes classes de sistemas de comunicação tempo-real. A comunicação tempo-real estrito *(hard)* (figura 6a) pertence aos domínios críticos, como controlo e automatização de processos industriais discretos, de aeronaves, de centrais e produção de energia, etc. É caracterizada por exigências de determinismo e fiabilidade que obrigam a ambientes muito controlados em dimensão e tráfego, e redes tempo-real e por vezes replicadas. A comunicação tempo-real lato *(soft)* (figura 6b) pertence aos domínios em que, não obstante uma necessidade de cumprimento de especificações temporais, a infraestrutura e o ambiente são tais que essas especificações não podem ser cumpridas com uma alta garantia. Estão nessas condições aplicações com tráfego isócrono (ex. multimédia) sobre redes grande-escala, que pela sua tecnologia e dimensão estão longe do determinismo das redes tempo-real estrito.

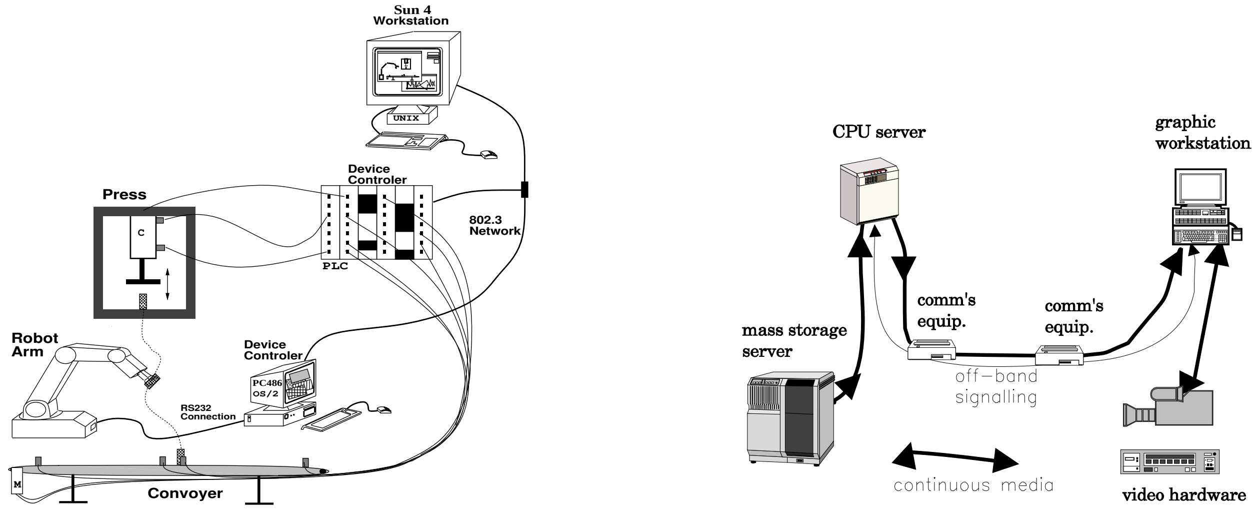


Figura 6: Comunicação Tempo-real: (a) Estrito *(hard)*; (b) Lato *(soft)*

O Modelo OSI

O suporte de sistema operativo para comunicação entre processos (IPC) é normalmente estruturado, senão fisicamente, pelo menos conceptualmente, em diversas camadas de funcionalidade. Esta estruturação resulta de um Modelo de Referência para Interligação de Sistemas Abertos, o modelo OSI. Neste modelo, tenta-se representar os níveis de abstração envolvidos na comunicação entre duas máquinas, possivelmente de marcas distintas, possivelmente em redes distintas.

Cada camada tem uma funcionalidade bem definida. O modelo é o de utilizador/servidor, em que cada camada $n-1$ é uma caixa preta para a camada n, à qual fornece os serviços que constam da sua especificação (fig. 7). Cada camada é concretizada por um protocolo, cujo propósito é atender um pedido de serviço da camada superior. Para tal, tem de dialogar com as camadas correspondentes noutras máquinas, utilizando os serviços das camadas inferiores. As camadas, com as funções mais relevantes, são as seguintes:

Física - redes locais e metropolitanas, rede pública *(wide-area)*

Comunicação de dados - comunicação na mesma rede; HDLC (ponto-a-ponto), 802.x (difusão)

Rede - interligação de redes, endereçamento, encaminhamento, controlo de congestão

Transporte - transporte fiável e ordenado de informação; fronteira tradicional entre *host* e comunicação

Sessão - interacção *end-to-end* entre aplicações

Apresentação - conversão de formato de dados entre máquinas heterogéneas, compressão, criptografia

Aplicação - transferência de ficheiros, RPC, directório, correio, acesso remoto a bases de dados

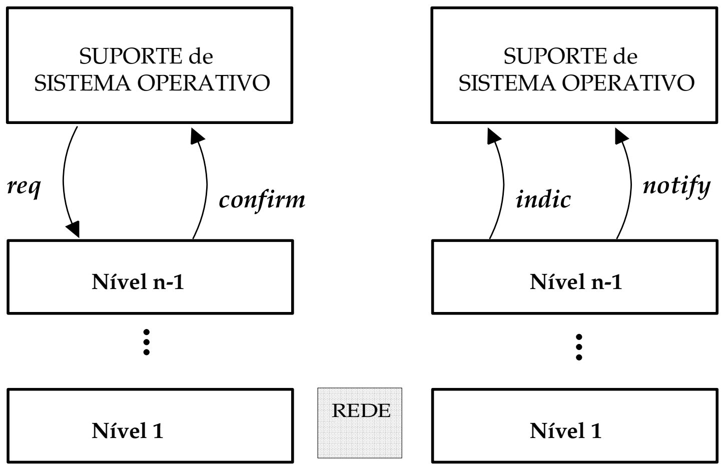


Figura 7: Comunicação no modelo OSI

As tecnologias de redes e ambientes de comunicação mais relevantes nos sistemas distribuídos são as seguintes:

- redes locais — CSMA/CD (Ethernet), Token-bus para ambientes industriais, Token-ring da IBM;
- redes de alta velocidade — rede FDDI de fibra óptica a 100Mb/s, ATM (Asynchronous Transfer Mode);
- redes metropolitanas — com redes do tipo DQDB;
- *internetworking* — rede mundial Internet com cerca de 2 milhões de nós, rede nacional Telepac, rede móvel digital GSM, (futura) rede de satélites Iridium;
- protocolos de Transporte — os clássicos ISO-TP4 e Internet-TCP/IP, os inovadores SIP, VMTP, XTP.

O modelo OSI deu origem a um esforço de normalização pela ISO (International Standards Organisation) no sentido de desenvolver protocolos para cada uma das camadas. Esse esforço tem-se desenvolvido nos últimos 15 anos, e a lentidão na saída de normas teve duas consequências importantes: a obsolescência de alguns protocolos; a substituição de outros por protocolos *de facto*. Segundo os níveis, assim variaram os efeitos, que numa análise sintética descrevemos:

- Pilha Completa:
  útil para garantir interoperabilidade; cada vez mais produtos; serviços de comunicação pré-definidos (FTAM, MMS, X400, X500, RDA).
- Níveis Físico (1) e de Com. Dados (2):
  normas de jure (ISO8802/3) são seguidas de perto ou à risca pelas normas de facto (ex. Ethernet).

- Níveis Rede (3) e Transporte (4):
  cada vez mais oferta de protocolos rede ISO-CLNP em coexistência com Internet-IP, por exemplo em *routers*; menos frequente, também ISO-TP4 a par de TCP.

- Nível Sessão (5):
  pouco útil; apenas enquadrada em oferta de pilhas completas, que não usam mais que o ISO-BAS; a dificuldade é que entra já no domínio do processamento distribuído e tem características obsoletas. Orientado para *rlogin*, ultrapassado por modelos mais estruturantes para computação distribuída, como o Modelo de Referência para Computação Distribuída Aberta, ODP, e por ofertas de perfis com tecnologias de facto, orientadas para chamada a procedimento remoto, como o *Distributed Computing Environment*, DCE da OSF.

- Níveis Apresentação (6) e Aplicação (7):
  ISO-ASN.1 é um protocolo para especificar tipos abstractos de dados, bastante utilizado, mesmo fora do contexto OSI; em ofertas de produtos sobre a pilha ISO completa, os protocolos de nível 7 tomam o nome de *application service elements*, ASE. Os ASEs mais gerais são utilizados nesses produtos (ACSE, FTAM, MSS, RDA); os mais específicos (ROSE, CCR, DASE) são pouco úteis.

6 Questões sobre o Projecto e Realização de Protocolos para Sistemas Distribuídos

Estas insuficiências do modelo OSI levantam algumas questões relacionadas com o projecto e realização de protocolos para sistemas distribuídos, que vale a pena debater:

Especificação vs. comportamento - a concepção de um protocolo/sistema de comunicação deve obedecer à espiral conhecida de projecto: requisitos, funcionalidade, projecto, concretização; a validação irá assim, fundamentadamente, incidir sobre a comparação entre a especificação e o comportamento analisado.

Argumento ***end-to-end*** **-** este argumento defende que qualquer qualidade de serviço do subsistema de comunicação deve obedecer a requisitos do utilizador, caso contrário é contraproducente a sua inclusão no suporte de sistema; no entanto, existindo classes de aplicações que possam ser servidas por um protocolo sofisticado, fará sentido incluí-lo no sistema, porque terá sido melhor concretizado e testado, e evitará a repetição sistemática da sua concretização pelos programadores.

Ambientes de suporte local (LSE) - ferramentas de suporte ao desenvolvimento e execução de protocolos são cruciais; sistemas clássicos (ex. UNIX) não fornecem grande suporte, pois não foram pensados para ser distribuídos; sistemas recentes começam a ser apetrechados com tais ambientes (*ex. x-kernel*), que permitem manipulação sofisticada de tampões, threads e temporizadores, e combinação de protocolos.

Arquitectura de protocolos - os protocolos de transporte tradicionais não são adaptados ao estilo *operações remotas* que suporta o modelo cliente/servidor – muito utilizado este último porque facilita a programação; requerem-se protocolos que se baseiem

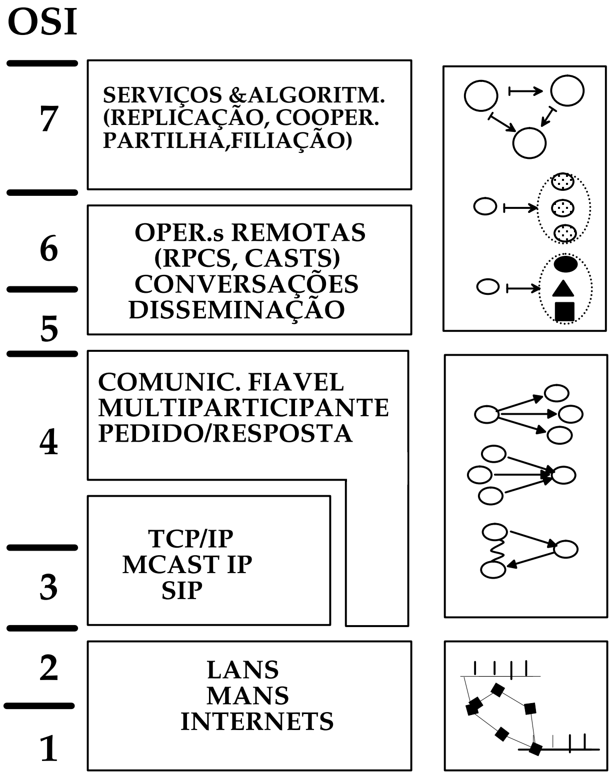


Figura 8: Alternativas ao modelo OSI

em comunicação pedido/resposta; sendo este modelo de comunicação bloqueante, tem o inconveniente de parar o processamento; uma técnica muito utilizada é o lançamento de uma nova actividade (*thread*), para se bloquear na comunicação, enquanto o processamento local prossegue; por outro lado, na comunicação multiparticipante, as técnicas de difusão são muito utilizadas, sendo inerentemente não bloqueantes; nesta operação em "circuito-aberto", a fiabilidade é em contrapartida dada por regras que disciplinam a entrega de mensagens aos vários participantes (quem recebe, e em que ordem).

Actividade Multiparticipante - as actividades multiparticipante têm adquirido importância crescente, devido à evolução do tipo de operação dos sistemas distribuídos; um sistema projectado para suportar este tipo de actividade, também denominada *orientada para grupos*, deve contemplar infraestruturas de rede multiponto, esquemas de endereçamento lógico, comunicação multiparticipante, gestão da filiação e actividade de grupos, e tempo global.

Alternativas à estruturação ISO-OSI

Em consequência dos comentários feitos atrás, algo aponta para alternativas ao modelo OSI. Em relação aos principais paradigmas apontados pelo modelo OSI que, recorde-se, fazia sentido em 1978, a evolução tecnológica apetrecha hoje em dia o projectista de sistemas distribuídos com novas e mais versáteis tecnologias. Alguns exemplos, que estão pictoricamente enquadrados no nível relevante do modelo OSI, na figura 8:

- infraestrutura para multiponto, compreendendo redes locais, metropolitanas e *wide area*;
- protocolos de rede multiponto (multicast-IP, SIP);
- protocolos de comunicação fiável, multiparticipante e orientados para operações remotas (XTP);
- sessões ponto-a-ponto tipo RPC, mas também multiponto (RPCs replicadas), *CASTs* (RPCs não bloqueantes), ou conversações;
- elementos de serviço para suporte de actividades multiparticipante, isto é, aplicações distribuídas orientadas para grupos, sob a forma de protocolos de gestão da replicação, cooperação e partilha.

7 Protocolos de Transporte para Computação Distribuída

Os actuais protocolos de transporte (tipo OSI) foram previstos para regimes de utilização e tecnologias que entretanto evoluiram:

Utilização

◦ acesso por terminal ◦ transferência de ficheiros	$\stackrel{evoluiupara}{\Longrightarrow}$	◦ operações remotas (RPC) ◦ comunicação multiparticipante ◦ multimedia tempo-real

Tecnologia

◦ meio: qualidade telefónica	$\stackrel{evoluiupara}{\Longrightarrow}$	◦ qualidade *data-rate*, fibra óptica
◦ BER (*bit error rate*) elevada		◦ BER: LANS 10^{-7}, fibra 10^{-15}
◦ débito baixo (64KBaud)		◦ débito elevado (FDDI 100MB/s, ATM)
◦ demoras elevadas		◦ alta velocidade

Protocolos de Transporte para Operações Remotas

Assim, os protocolos de transporte para operações remotas necessitam de um conjunto de atributos para se comportarem eficientemente:

- bom comportamento impulsivo *(sobreescrita, latência)*
  controlo de fluxo por taxa (*rate-based*)
  retransmissão selectiva
  ligações de curta duração (*short-lived connections*)
- pedido/resposta de baixo custo mas fiável;
  datagrama fiável
  endereços estáveis

- suporte para tolerância a faltas e tempo-real:
  prioridades
  difusão fiável
  segurança

Tecnologia Seleccionada

Protocolo de Transporte XTP - Express Transfer Protocol

O Express Transfer Protocol, XTP, encontra-se neste momento em normalização, e apresenta um conjunto de atributos que respondem aos requisitos apontados atrás.

- orientado para operações remotas e comunicação multiparticipante;
- concretiza o *nível de transferência*, i.e. rede+transporte;
- indicado para redes rápidas e de débito elevado (FDDI, ATM);
- prevista concretização em VLSI; em curso a normalização;
- mecanismos:
  ◦ ligações tempo-real;
  ◦ controlo de fluxo por taxa;
  ◦ retransmissão selectiva;
  ◦ tráfego fora-de-banda;
  ◦ prioridades;
  ◦ difusão fiável.

Ferramentas de suporte ao desenvolvimento e execução de protocolos

A ideia subjacente a estas ferramentas é: constituirem um complemento de funções do kernel; permitirem uma desejável harmonização de interfaces (portabilidade); melhorarem a fiabilidade do protocolo, desejavelmente sem perda de desempenho; aumentarem a modularidade de protocolos conhecidos, quando reconstruídos sobre o ambiente, facilitando a reutilização e combinação de protocolos.

Tecnologia Seleccionada

O ambiente de suporte x-Kernel

O ambiente de suporte x-Kernel é um exemplo, representado na figura 9. Tem as seguintes características:

- fornece um conjunto de abstracções para encapsular protocolos;
- o modelo de interacção é obviamente o de passagem de mensagens;

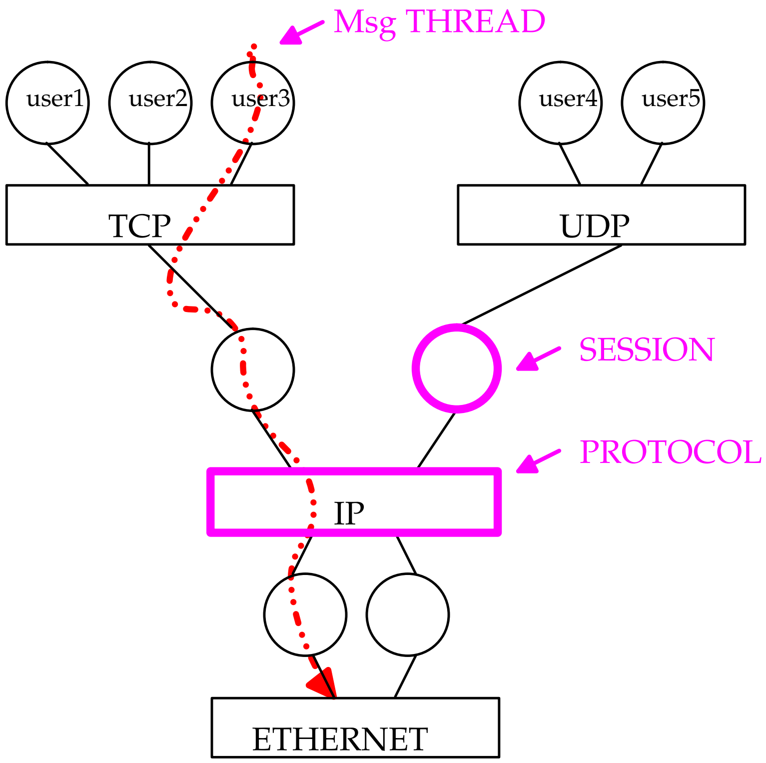


Figura 9: O ambiente de suporte x-Kernel

- a chegada de uma mensagem ou um pedido de transmissão desencadeiam a criação de uma actividade (thread), que acompanha essa mensagem até ao processo receptor ou até à rede;
- as primitivas permitem facilmente a composição de protocolos (ex. níveis OSI, TCP+UDP+IP);
- os *objectos* x-kernel são: grafos de protocolos interligados; sessões, isto é, instanciações de pontos de acesso a cada protocolo; mensagens;
- as *operações* sobre os objectos são: *push* para transmitir, *pop* para receber, e *demux*, para desmultiplexar o tráfego de vários protocolos para um só;
- o x-kernel tem vários grupos de primitivas de gestão acessíveis ao programador:

  Gestor de tampões;

  Gestor de projecções;

  Gestor de eventos.

8 Módulos para suporte de actividades multiparticipante

As tecnologias mais recentes de sistemas distribuídos visam satisfazer os requisitos de aplicações com elevado nível de distribuição e interactividade. Estas aplicações são amiúde multiparticipante, o que põe em questão as tecnologias tradicionais, orientadas

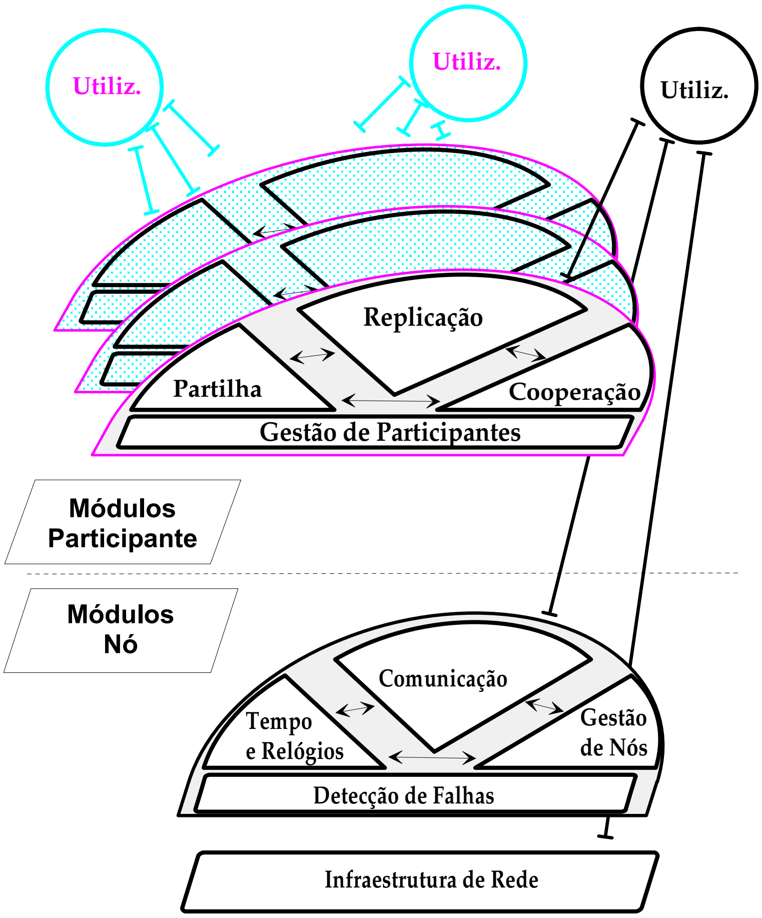


Figura 10: Módulos e organização de um sistema orientado para grupos

para interacção ponto-a-ponto. No entanto, o contrário não é verdade, isto é, num sistema multiparticipante, o ponto-a-ponto não é mais que um caso particular.

O conjunto das tecnologias e paradigmas relacionados com actividades multiparticipante é também denominado genericamente *orientação para grupos*. Estas questões põem-se verticalmente em todo o sistema, assumindo diferentes facetas consoante o nível de abstracção, desde a comunicação à computação. A figura 10 representa a organização de um sistema orientado para grupos, cujos módulos discutimos em seguida:

- Nível Nó (comunicação, entre processadores):
  - Infraestrutura de rede multiponto - deve procurar obter-se capacidade de comunicar em multiponto, quer sobre redes locais (multicast), quer sobre redes públicas (multicast-IP).
  - Tabela de estações activas - a detecção de falhas em ponto-a-ponto é trivial; em multiponto, é obrigatório gerir o conjunto de nós da rede, mantendo uma tabela de estações activas congruentemente em todos os nós.
  - Comunicação em grupo - os protocolos ditos de difusão fiável assumem, por maioria de razão, um papel crucial nas aplicações multiparticipante, ou orientadas para grupos.

- Tempo e temporizações - o papel do tempo, nomeadamente de uma noção de tempo global a todo o sistema, tem sido menosprezado em sistemas centralizados; é no entanto uma ferramenta indispensável de sincronização em sistemas distribuídos.

- Nível Participante (computação, entre processos):

  - Gestão da filiação de participantes - para cada grupo, isto é, actividade distribuída, pode existir um número variável de participantes, inclusivé mais que um sobre o mesmo nó; a gestão automatizada da evolução da filiação desses participantes significa dar a conhecer coerentemente a todos os membros quem entra, quem sai, quem falha; intuitivamente, compreende-se que pode simplificar extraordinariamente a correcta execução de uma aplicação distribuída.
  - Gestão da actividade de grupos - por outro lado, a construção de aplicações pode ser feita ad-hoc, sobre ferramentas básicas de IPC (sockets, RPC, etc.); a correcção de uma tal aplicação, nos vários cenários possíveis, é no entanto difícil de assegurar; alternativamente, podemos imaginar uma caixa-de-ferramentas facultando a programação de actividades básicas distribuídas que vão além da clássica operação remota cliente-servidor; por exemplo, protocolos de controlo de:

    replicação;
    cooperação;
    partilha

No que se segue, vamos dar exemplos das tecnologias subjacentes aos vários módulos.

Tecnologia Seleccionada

Infraestrutura de rede para comunicação multiparticipante

Embora a infraestrutura de rede de comunicações em grande-escala seja maioritariamente pública (ex. Internet), existem duas noções tecnológicas importantes que podem modificar o modo por vezes não eficiente como vemos as redes, exemplificadas na figura 11:

- **a dualidade** *participante–nó* ou processo–processador, quando falamos de comunicação em sistemas distribuídos; o seu reconhecimento introduz o conceito de subsistema de comunicação, negligenciado devido à preponderância de sistemas monolíticos (ex. UNIX); este conceito tem vantagens do ponto de vista de desempenho, detecção de falhas, e multiplexagem explícita entre protocolo de comunicação, um por nó, e os vários participantes de um mesmo grupo de comunicação; é tecnicamente viável hoje em dia devido à abertura progressiva dos núcleos UNIX e ao advento dos microkernels.

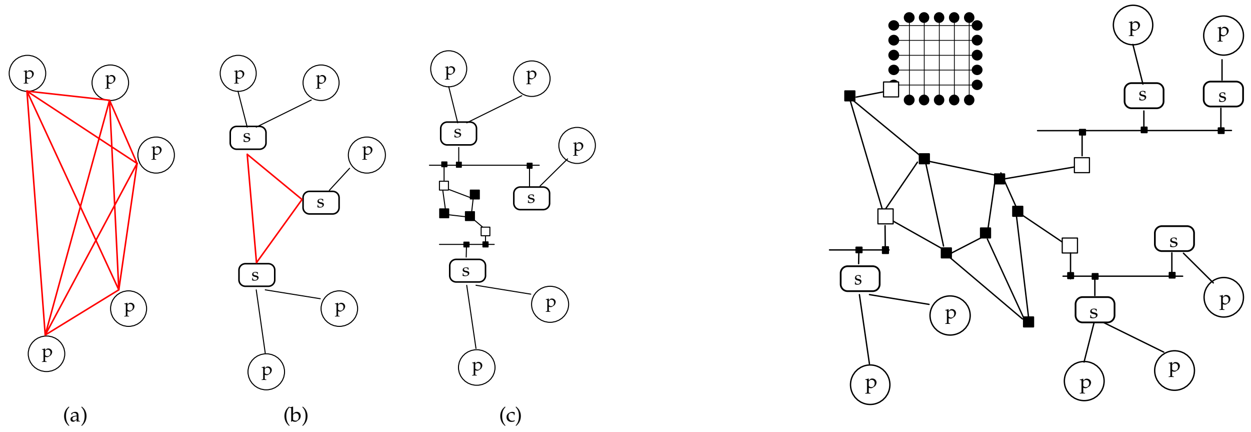


Figura 11: Exemplificação dos conceitos: (a) nó-participante; (b) *WAN-of-LANs*

- **a estrutura** *WAN-of-LAN/MANs*, que corresponde perfeitamente a uma rede grande-escala presente e num futuro pelo menos próximo, e cujo reconhecimento tem um impacto significativo na arquitectura de protocolos e algoritmos para sistemas distribuídos, em relação à aproximação tradicional, em que as redes grande-escala eram consideradas puramente ponto-a-ponto; de facto, os utilizadores não querem pagar a independência da tecnologia quando a têm ao alcance.

Tecnologia Seleccionada

Protocolos de Comunicação em Grupo

Como veremos adiante, a fiabilidade e correcção da programação em difusão dependem da disciplina imposta pelo protocolo de comunicação fiável entre os participantes. Existem actualmente vários tipos de protocolos, revestindo de várias facetas, que são combinações das seguintes propriedades básicas:

Endereçamento: todos no grupo; selectivo (para alguns);

Fiabilidade: unânime (chega a todos); pelo-menos-(N,Para) (chega pelo menos a N destinatários, ou ao(s) destinatário(s) X,...); melhor-esforço-(N,Para), melhor-esforço-(0) é equivalente a datagrama;

Ordenação: FIFO (*first-in-first-out*); causal, isto é, por causa-efeito; total, isto é, na mesma ordem em todos os lugares; total e unânime corresponde à conhecida difusão atómica;

Sincronismo: assíncrono ou síncrono, isto é, sem ou com demora conhecida e limitada;

Temporais: dependendo de o protocolo ser construído à custa de relógios ou não, será *timer-driven* ou *clock-driven*; a comunicação pode igualmente ser disparada por tempo ou por eventos.

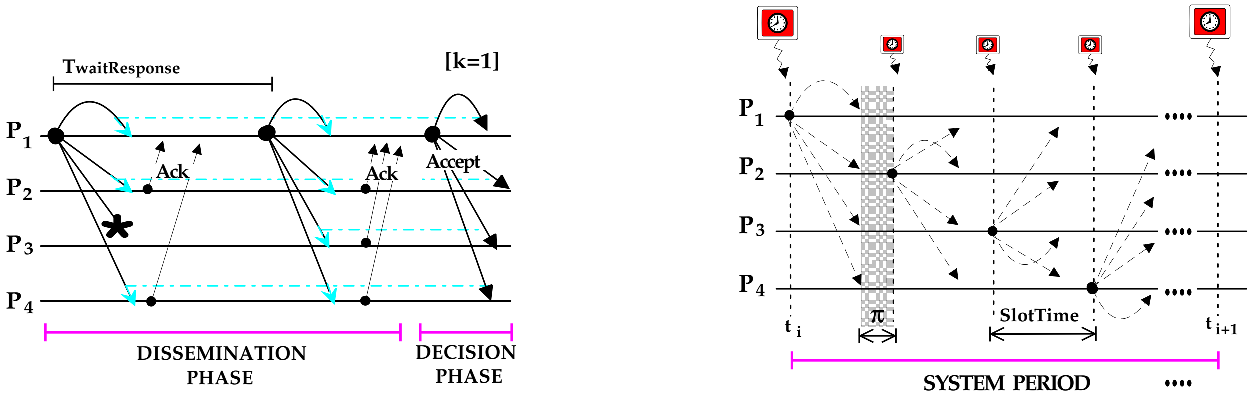


Figura 12: Protocolos de comunicação em grupo: (a) AMp; (b) MARS

Damos exemplos de alguns protocolos de comunicação em grupo: AMp, um protocolo atómico *timer-driven* síncrono (fig. 12a); AAS, um protocolo atómico *clock-driven* síncrono (fig. 15); MARS, um protocolo disparado por tempo igualmente síncrono (fig. 12b).

Tecnologia Seleccionada

Ordem causal com vectores de estampilhas lógicas (vector-clocks)

O objectivo de um protocolo de ordem causal é o de ordenar mensagens segundo a sua relação potencial de causa-efeito. Nomeadamente, $m1 \rightarrow m2$, se:

- $m1$ enviada antes de $m2$ pelo mesmo processo (FIFO);
- $m1$ chega ao remetente de $m2$ antes deste enviar $m2$.

Uma técnica muito importante para assegurar esta ordem são os chamados vectores de estampilhas lógicas ou *vector-clocks*. São vectores mantidos por cada elemento do grupo, em que cada posição é um contador do número de mensagens recebidas localmente, de cada elemento. A actualização dos vectores segue determinadas regras, tendentes a preservar as relações de causalidade entre mensagens trocadas no grupo (fig. 13).

Regras de realização:

- início: $VTi[k] = 0 \; \forall i, k$
- a cada envio de m, incrementar $VTi[i]$ (i-local), estampilhar m com VTi: $VT(m)$
- a cada recepção, incrementar $VTi[i]$, e actualizar cada posição de VTi com o máximo de $[VTi, VT(m)]$.

Mostra-se que estas regras garantem que os vectores representam fielmente a causalidade potencial, ou precedência. Os protocolos causais utilizando vectores de estampilhas apenas têm de assegurar a entrega por ordem de precedência, o que é feito comparando os vectores das mensagens.

Regras de precedência:

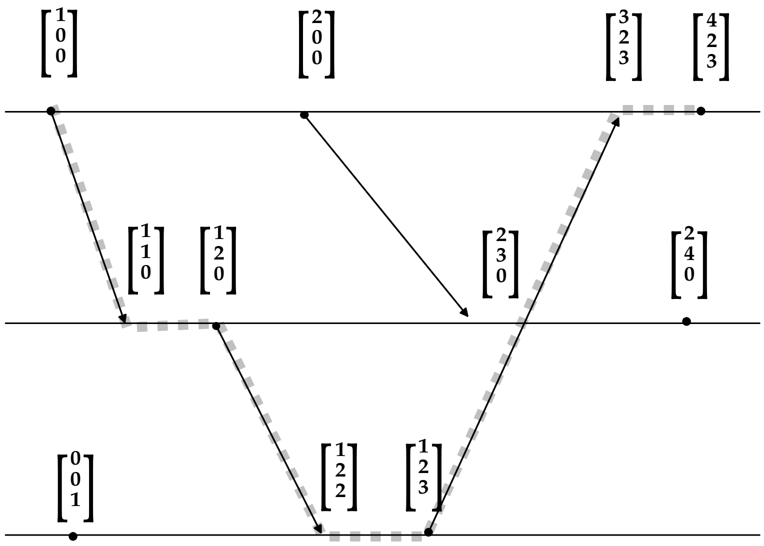


Figura 13: Ordem causal com vectores de estampilhas lógicas

- $m1 \rightarrow m2$ sse $VT(m1) < VT(m2)$
- $m1 \parallel m2$ sse $VT(m1) \parallel VT(m2)$

Tecnologia Seleccionada

Propriedades de um Relógio Global

Um dos módulos muito importantes de um sistema orientado para grupos é um relógio global (fig. 14). Por diversas razões aliadas à qualidade do tempo obtido, e à fiabilidade, deve ser evitado um relógio centralizado lido globalmente por todos os nós. Nesse caso, a solução passa por ter todos os relógios em todos os nós, sincronizados mutuamente. À medida do desvio entre relógios chama-se *Precisão* (π).

Por outro lado, num número de sistemas, é igualmente necessário sincronizar esse tempo interno com uma medida do tempo universal (ex. TAI, UTC, etc.). À medida do desvio entre qualquer relógio e essa base de tempo absoluta chama-se *Exactidão* (α).

Dado que os relógios locais são imperfeitos, estes derivam com o tempo. Para os manter "certos", utilizam-se protocolos de *sincronização de relógios*, que se executam periodicamente.

Tecnologia Seleccionada

Protocolo de difusão atómica com relógios (Δ-protocol)

O objectivo de um protocolo atómico é entregar mensagens enviadas por vários remetentes, a todos os destinatários, na mesma ordem. Alguns destes protocolos

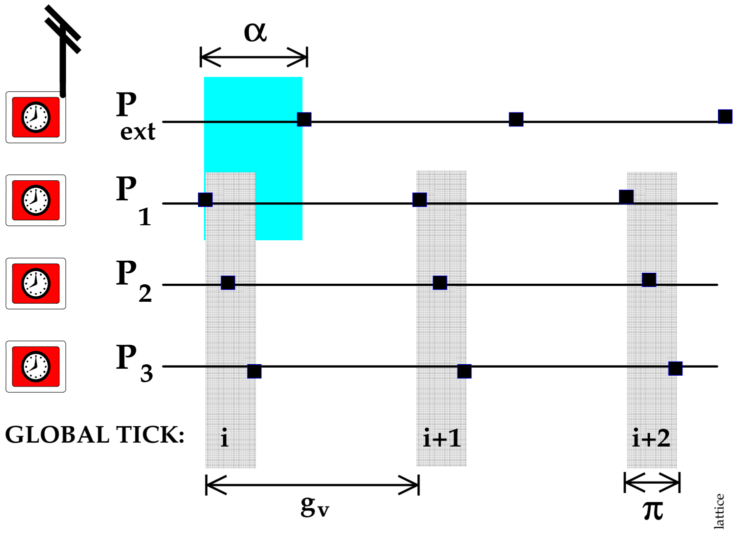


Figura 14: Propriedades de um relógio global: π, precisão; α, exactidão; g, granularidade

procuram igualmente fazê-lo ao "mesmo" tempo. Para isso são controlados por relógios globais, isto é, *clock-driven*, ou Δ-*protocols*.

Os parâmetros do sistema são: a latência máxima de comunicação, δ; a precisão dos relógios globais, π. A técnica baseia-se em escolher apriori o instante, igual em todos os relógios, em que uma dada mensagem será entregue, e fazê-lo de modo a que aquela espere um intervalo suficientemente grande para garantir que após a entrega não vai chegar nenhuma mensagem que seja anterior. Essa demora de entrega é uma constante: $\Delta = \delta + \pi$.

Esboço do Protocolo:

- o remetente estampilha a mensagem m a enviar, com $T(m)$ no relógio local;
- a transmissão é fiável;
- os destinatários recebem m e aguardam;
- os destinatários entregam m no instante $T_D = T(m) + \Delta$, medido nos relógios locais.

Tecnologia Seleccionada

Como determinar correctamente o Estado Global de um sistema

O objectivo de um protocolo de recolha de estado coerente é conseguir tirar uma fotografia ao estado (distribuído) do sistema *(snapshot)*, que seja correcta. De facto, essa recolha de estado pode ser incoerente, se feita sem regras, como está exemplificado na figura 16:

◦ 1 - ler S1:A após iniciar transferência, ler S2:B antes desta chegar; resultado: $450 + 200 \neq 700$

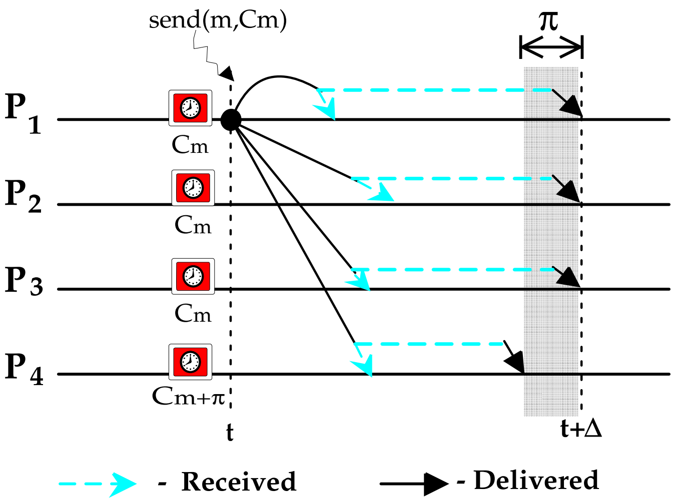


Figura 15: Protocolo de difusão atómica com relógios, AAS

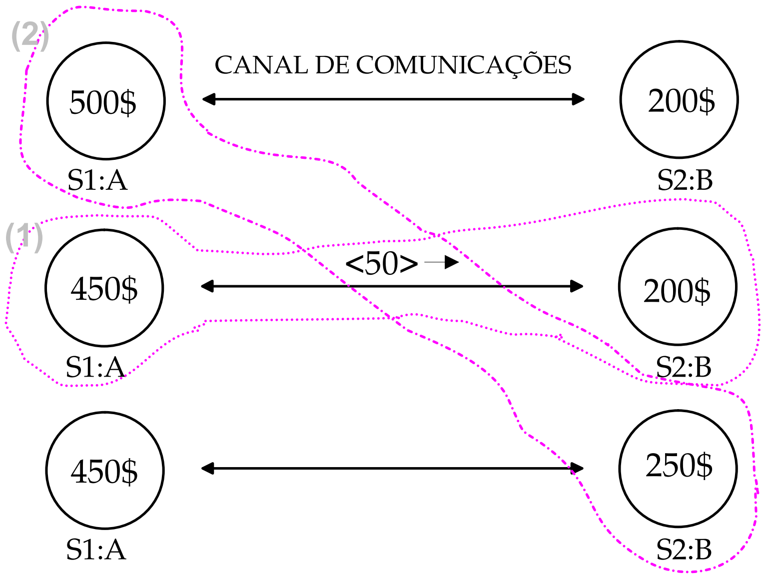


Figura 16: Exemplos de incoerência

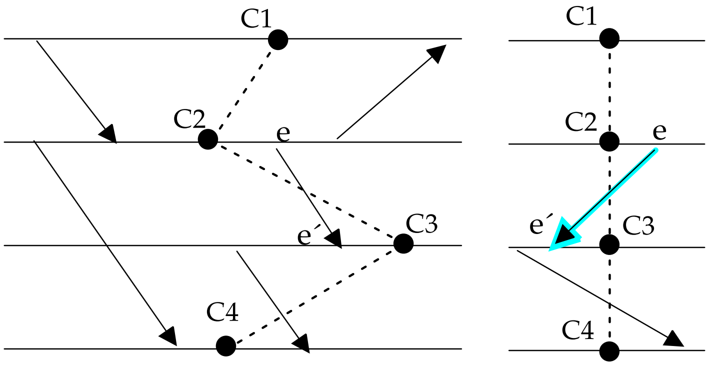


Figura 17: Exemplo de corte incoerente, CC

◦ 2 - ler S1:A antes de iniciar transferência, ler S2:B depois desta chegar; resultado: $500 + 250 \neq 700$

O estado global correcto chama-se *estado global coerente, EGC*. Um *corte coerente, CC* (fig. 17), é então uma linha poligonal que atravessa o conhecido diagrama espaço-tempo da computação, unindo um evento em cada processo, de modo a que todos os seus eventos sejam concorrentes, isto é, não estejam causalmente relacionados, e esses eventos representem o estado de cada processo. Existe inclusivamente uma regra prática bastante intuitiva para observar se um corte é coerente: *um CC não tem atravessamentos de mensagens (significando mensagens em trânsito) para trás.*

9 Computação Distribuída: estilos de programação

Na computação em sistemas distribuídos, há essencialmente quatro estilos/modelos de programação:

- **operações remotas**, que suportam o conhecido modelo cliente/servidor, uma distribuição que designaremos por *sequencial*, dadas as características bloqueantes, em que o fluxo de controlo salta sequencialmente para as diversas máquinas servidoras, na sequência de invocações sucessivas;

- **transacções**, interessantes na perspectiva de serem um modo de materializar operações remotas fiáveis com semântica exactamente-uma-vez;

- **memória partilhada distribuída**, que suporta programação concorrente transparente à distribuição dos processos por várias máquinas, pelo que a designaremos por distribuição *indirecta*;

- **difusão**, que é um paradigma em circuito-aberto, baseado em passagem de mensagens subordinada a uma semântica de entrega forte, que garante a correcção de execução.

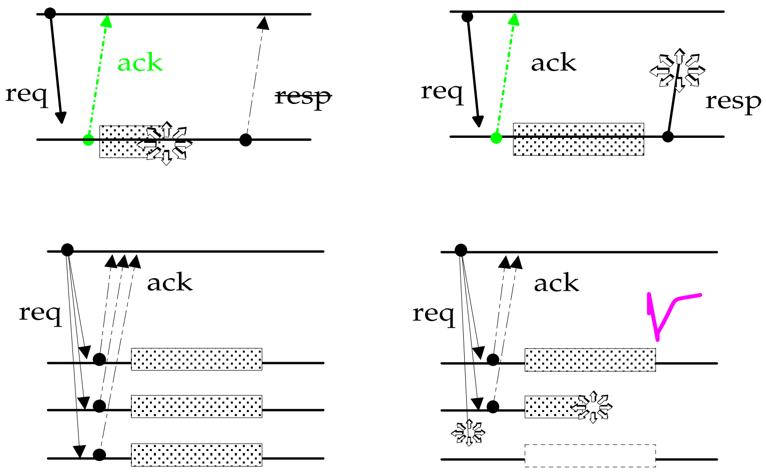


Figura 18: Fiabilidade comparada de operações remotas e difusão

Onde o modelo de Operações Remotas não chega

Os dois primeiros modelos, nomeadamente o cliente/servidor, são mais conhecidos. No entanto, têm problemas tanto do ponto de vista de funcionalidade como de confiabilidade.

Do lado 'funcionalidade', a interactividade e descentralização põem problemas ao modelo cliente/servidor puro, baseado em *operações remotas*, de que a RPC é o exemplo por excelência. A RPC é por definição: bloqueante, unilateral e assimétrica. No entanto, cada vez mais se necessita suportar interacções não-bloqueantes, multilaterais, e simétricas ou *peer-to-peer* (ex. conversações, casts, processamento replicado):

- sistemas reactivos tempo-real, por exemplo, reagem com o ambiente, são melhor tratados no domínio dos eventos, através de *message-interrupts*, e com concorrência;
- sistemas altamente interactivos (ex. CSCW), necessitam tanto de passagem de mensagens como de outros paradigmas.

Os sistemas baseados em RPC têm modos de obviar as suas limitações, como o conhecido *fork-and-call*, que cria uma actividade que se bloqueia enquanto a outra prossegue a computação. No entanto, será desejável que esses problemas se possam resolver ao nível do modelo de actividade distribuída, com alternativas que complementem o modelo de operações remotas.

O estilo de programação em difusão, como já foi dito, é um paradigma em circuito-aberto, cuja correcção de execução depende, não de uma confirmação do servidor, para fechar o circuito, mas da semântica de entrega de mensagens. Isto é, à vantagem intuitiva de programar sistemas distribuídos com passagem de mensagens, junta-se o efeito disciplinador dos protocolos de comunicação fiável, para obviar à reconhecida entropia e hostilidade deste estilo de programação, se tomado "ao natural". A semântica de entrega de mensagens disponível ao programador em difusão é variada, podendo ir desde o datagrama não fiável até à difusão atómica. O modelo possibilita ainda ultrapassar as dificuldades de assimetria e unilateralidade da operações remotas com RPC.

Do ponto de vista da fiabilidade, a RPC tem uma semântica básica *pelo-menos-uma-vez*, dada a dificuldade de destrinçar entre uma falha de comunicação e uma falha do servidor (fig. 18). Como mecanismos adicionais, como NVRAM para guardar identificadores de serviços, pode atingir-se uma semântica *no-máximo-uma-vez*, que em determinadas situações é insuficiente.

O modelo de transacções vem possibilitar a execução de operações remotas com semântica exactamente-uma-vez, e é extremamente útil para suportar interacção fiável com repositórios de dados persistentes (ex. bases de dados). Além disso, permite a execução de múltiplas operações de modo atómico.

O modelo de memória partilhada distribuída permite preservar a ilusão de programação concorrente num sistema centralizado, sobre um sistema distribuído. É muito útil em sistemas paralelos distribuídos.

O modelo de difusão, como já foi dito, é um paradigma em circuito-aberto, cuja fiabilidade depende, não de uma confirmação do servidor, para fechar o circuito, mas da replicação das acções respeitantes à computação, e consequentemente, da relação entre o número de elementos do grupo de difusão e as hipóteses de falhas de elementos (fig. 18). Na hipótese de falha por paragem, é necessário ter um grupo de $f + 1$ réplicas, em que f é o número máximo de elementos que podem falhar. No modelo de difusão é possível, sem complexidade adicional, produzir uma semântica *exactamente-uma-vez (exactly-once)*, que é a que reflecte mais precisamente os requisitos de uma operação fiável.

Súmula dos estilos de programação de aplicações distribuídas

Em suma, traçamos uma comparação entre os vários estilos de programação de aplicações distribuídas, quer do ponto de vista da semântica operacional, quer do comportamento face a falhas.

- **Operações Remotas –** circuito fechado, ponto-a-ponto, semântica FIFO:
  RPC - comportamento pelo-menos-uma-vez *(at-least-once)*;
  RPC com memória não volátil - comportamento no-máximo-uma-vez *(at-most-once)*
  Transacções - comportamento exactamente-uma-vez *(exactly-once)*
- **Difusão –** circuito aberto, multiponto, semântica variada, de datagrama a atómica:
  máquina de estado (ME) replicada - comportamento exactamente-uma-vez *(exactly-once)*
  sincronismo virtual (*ME multiactividade*) - comportamento exactamente-uma-vez *(exactly-once)*
  conversações multiparticipante
- **Operações Remotas com Difusão –** ponto-a-multiponto, semântica cliente FIFO, semântica servidor variada, de datagrama a atómica:
  RPC replicada
  Aplicações orientadas para grupos

10 Programação orientada para grupos

O conceito de grupo é inerente a actividades distribuídas, como podemos ver por exemplos variados:

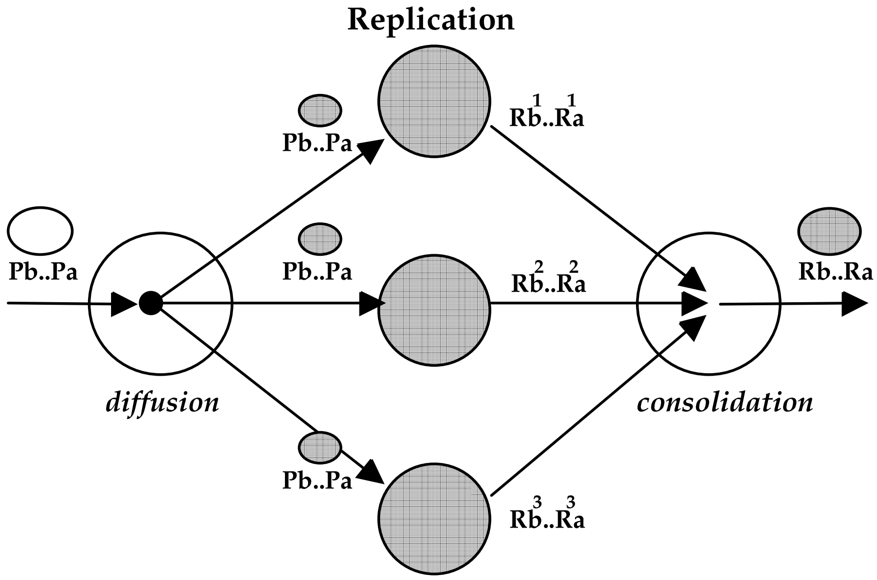


Figura 19: Replicação

COOPERAÇÃO ⇒
- servidor de BD's distrib. (fragmentado);
- edição cooperativa de documentos;
- teleconferência;
- operação de uma célula flexível de fabrico;
- processamento paralelo distribuído;

PARTILHA ⇒
- exclusão mútua num grupo;
- partilha por recursos do sistema: serialização, controlo da concorrência;

REPLICAÇÃO ⇒
- servidor de BD's distrib. (replicado);
- sensor ou actuador replicado;
- sistema de ficheiros distribuído e replicado;
- controlo de processos replicado (votação).

A programação orientada para grupos é um campo inovador de interesse crescente. Obedece ao modelo de difusão e suas variantes, e baseia-se num conjunto simples de premissas:

1. definição de tipos de actividades básicas a desempenhar por grupos de participantes (como vimos nos exemplos acima):

   ◦ **replicação, cooperação, partilha**

2. concepção dos protocolos de gestão dessas actividades:

   ◦ **replicação (fig. 19): activa, semi-activa, passiva**
   ◦ **cooperação (fig. 20): fragmentação funcional ou estrutural, despacho;**
   ◦ **partilha (fig. 21): serialização, controlo da concorrência**

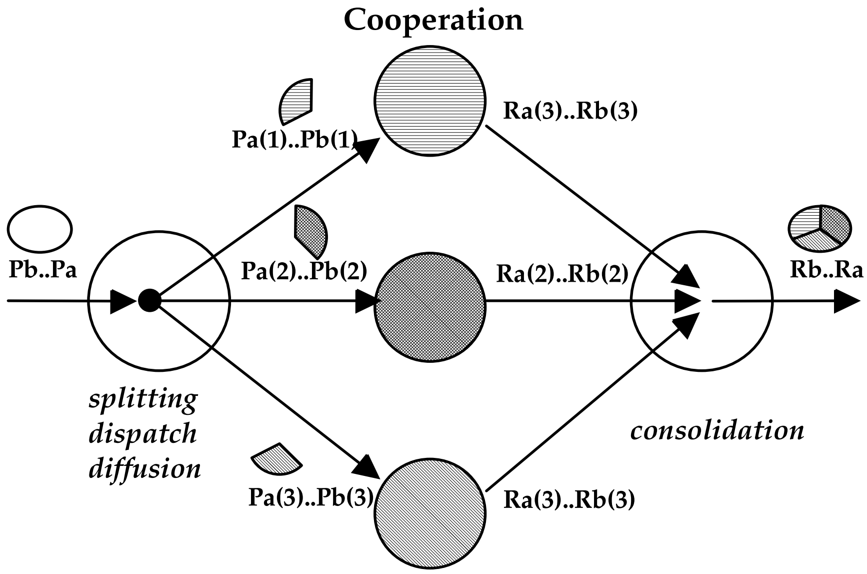


Figura 20: Cooperação

3. definição da semântica de comunicação fiável com/entre os elementos do grupo:

   ○ **melhor-esforço**
   ○ **difusão fiável**
   ○ **difusão causal**
   ○ **difusão atómica**

Alguns destes protocolos são complexos, mas a aproximação adequada é serem fornecidos numa caixa-de-ferramentas no sistema. Uma vez isto feito, torna-se expedito percorrer os pontos enumerados acima, e assim programar correctamente, e com fiabilidade, aplicações complexas.

Por outro lado, os grupos (ex. de objectos, de processos, de réplicas de páginas de memória) são um modelo bastante estruturante no panorama da comunicação e computação distribuída, pois podem suportar, de um modo *visível*, ou *invisível* — isto é, aparentes, enquanto grupos, ao utilizador — todos os diferentes estilos de programação de que falámos atrás:

- memória partilhada
- RPC
- servidores replicados
- dados replicados
- grupos de processos (conversações)

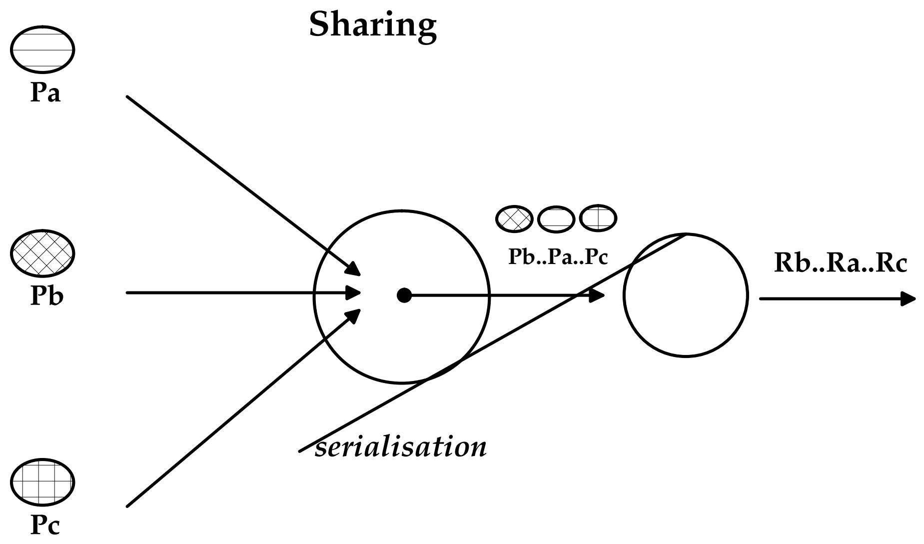


Figura 21: Partilha

11 Programação orientada para Transacções sobre Dados

As transacções atómicas são um modelo de programação, que possibilita essencialmente duas coisas, em relação ao modelo simples de operações remotas:

- semântica de falhas *exactamente-uma-vez* num modelo cliente/servidor;
- conjunto indivisível de acções executado fiavelmente.

Uma transacção atómica (T.A.) é definida por um conjunto de propriedades, as chamadas propriedades *ACID* em anglo-saxónico:

- **A**tomicity - atomicidade, define que ou todas as acções de uma T.A. se executam, ou não se executa nenhuma (e a transacção aborta).
- **C**onsistency - coerência, define que um conjunto de dados correcto antes de uma T.A., é correcto após esta.
- **I**solation - isolamento, define que nenhum resultado parcial de uma T.A. é conhecido antes do fim desta.
- **D**urability - permanência, define que os resultados de uma T.A. são persistentes (não voláteis).

As T.A.s visam dados, normalmente as chamadas Bases de Dados (BD) Transaccionais. Duas grandes vantagens das T.A.s são: o encapsulamento fiável de múltiplas operações; e possibilitarem o entrelaçamento de várias T.A.s sobre a mesma BD. Denominam-se tais sistemas de Transacções Concorrentes.

Obviamente, tal modo de funcionamento impõe regras para respeitar as propriedades ACID, nomeadamente, aquilo que se chama *serializabilidade*, que especifica que uma execução concorrente deve ser equivalente a uma dada execução em série, o que é uma condição

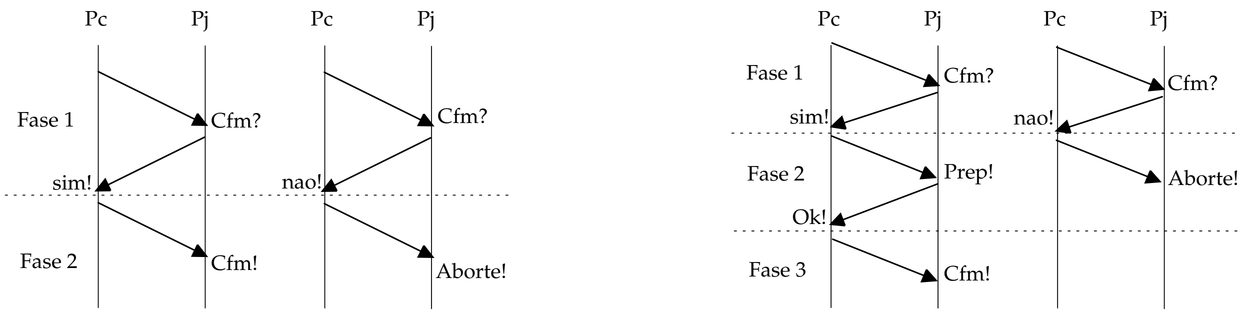


Figura 22: Protocolos de confirmação atómica: (a) duas fases; (b) três fases

de correcção intuitiva. Nas T.A.s, isto é feito por mecanismos de *controlo da concorrência*, desempenhados por trincos ou *locks*. O mais conhecido é o trinco em duas fases, *two-phase lock*.

As Transacções Distribuídas apresentam três vantagens importantes: a concorrência pode vir de diferentes nós; as acções múltiplas da T.A. podem agora abranger diversos nós; a disponibilidade potencial dos dados aumenta.

No entanto, as T.A.s distribuídas têm o problema de garantir uma das propriedades ACID, a atomicidade, na presença de falhas de alguns nós, ou partições da rede, isto é, com falhas parciais. Tal é conseguido executando a terminação — confirmação ou aborto (*commit/abort*) — através de protocolos de confirmação, em duas fases ou em três fases (2ϕ-, 3ϕ-*commit*).

Finalmente, observe-se que, se bem que a semântica de falhas de uma T.A. seja exactamente-uma-vez, a verdade é que quando o(s) servidor(es) falha(m), o sistema fica indisponível. Por razões de disponibilidade é então conveniente replicar a BD.

As Transacções Distribuídas sobre BDs Replicadas têm um problema adicional: a congruência das cópias. Este requisito é formulado especificando que a T.A. tem que ter *serializabilidade-1-cópia*, isto é, comportar-se como se apenas houvesse uma cópia. Isto é conseguido à custa de vários mecanismos, como: políticas *read-one/write-all*; *quorums* ou votações. Estes visam essencialmente assegurar que a transacção só é bem sucedida quando consegue escrever em todas as cópias, no primeiro caso, ou numa maioria das cópias, no segundo.

Tecnologia Seleccionada

Protocolos de confirmação (*commit*) atómica

Os protocolos de confirmação atómica (*atomic commit*) visam terminar uma transacção distribuída por vários nós.

O protocolo de confirmação em duas fases (*two-phase commit*) é bastante conhecido, e utilizado já em alguns sistemas gestores de bases de dados distribuídas (SGBDD) comerciais (fig. 22a). No entanto, tem uma janela de vulnerabilidade importante, que afecta o desempenho esses SGBDDs: bloqueia em diversos cenários de falha de um dos nós durante a confirmação, ou com partições da rede.

O protocolo de confirmação em três fases (*three-phase commit*) tem uma janela de vulnerabilidade bastante mais reduzida, à custa de alguma complexidade acrescida. No entanto, não bloqueia por falha de nós, e bloqueia apenas em cenários infrequentes de partições da rede (fig. 22b).

12 Programação orientada para Memória Partilhada Distribuída

A programação concorrente, com comunicação e sincronização entre processos através do mesmo espaço de endereçamento sobre memória partilhada, é um paradigma conhecido e de fácil utilização, uma vez que evita a programação com passagem de mensagens.

Em consequência, se essa memória partilhada fôr distribuída, DSM, temos um meio de distribuição *indirecta* entre processos em várias máquinas.

Do ponto de vista de desempenho da DSM, há que procurar aproximar o seu comportamento o mais possível do de uma memória local. Três questões se destacam acerca do desempenho:

- optimizar o tráfego das mensagens para gestão da memória;
- mascarar a maior latência de acessos remotos em relação aos locais;
- evitar falsa-partilha, isto é, dois items que estão na mesma página, sem terem qualquer relação entre si.

Arquitectura da DSM

Existem diversos modos de concretizar esta abstracção. Do ponto de vista de organização da DSM nas várias máquinas, temos quatro classes de aproximações:

- servidor centralizado – uma máquina possui toda a memória, escreve todas as actualizações e devolve os blocos (ex. páginas) que lhe peçam para leitura; PROBLEMA**: engarrafamento**;
- migração – o dono da memória migra de máquina para máquina, consoante os pedidos de acesso; a "propriedade" pode ser modularizada à página, e o acesso é exclusivamente do dono corrente; PROBLEMA**:** *thrashing* **devido a pingue-pongue**;
- replicação para leitura – páginas replicadas nos lugares dos potenciais donos; controlo de concorrência escritor/leitores; bom para tipo de operação RD$\gg$WR; PROBLEMA**: dispendioso em escrita**;
- replicação total – páginas replicadas nos lugares dos potenciais donos; controlo de concorrência escritores/leitores; PROBLEMA**: requer protocolos de congruência**;

Quando se permitem acessos concorrentes em escrita e leitura às várias cópias da memória, são necessários protocolos que: (a) actualizem correctamente as várias cópias; (b) mantenham a congruência dessas mesmas cópias ou réplicas.

Protocolos de coerência de DSM

Os protocolos de coerência de DSM pertencem a um de dois tipos:

- coerência rígida *(strict coherence)* – uma leitura retorna o valor mais recentemente escrito (ordem total);
- coerência frouxa *(loose coherence)* – formas de ordenação relaxadas;

Existem diversos paradigmas de congruência de cópias nas operações sobre a memória de coerência frouxa:

- congruência sequencial *(sequential consistency)* – operações concorrentes são serializáveis; ordem é a de cada programa individual;
- congruência geral *(general consistency)* – todas as cópias terão eventualmente o mesmo conteúdo após todas as escritas se completarem;
- congruência por processador *(processor consistency)* – ordem FIFO das escritas de cada processador;
- congruência fraca *(weak consistency)*, que distingue entre dois tipos de acessos — sincronização e normais; os acessos para sincronização são guardados entre primitivas *acquire* e *release*:
  - congruência na entrada *(entry consistency)* – ordem total entre acessos para sincronização e conjuntos de acessos normais;
  - congruência na saída *(release consistency)* – ordem parcial entre: *release* e acessos subsequentes; *acquire* e acessos anteriores;

Protocolos de actualização de DSM

Existem duas classes de protocolos de actualização de DSM:

- invalidação *(write-invalidate)* – aquando de uma escrita, invalidam-se todas as páginas menos a do escritor;
- actualização *(write-update)* – aquando de uma escrita, actualizam-se todas as páginas;

Em relação à velocidade da actualização, estes últimos dividem-se em:

- pronta *(eager-update)* – a actualização é feita aquando da escrita;
- retardada *(lazy-update)* – a actualização de cada página é retardada até o processo local a esta fazer um acesso.

Figura 23: Protocolos para DSM: (a) coerência rígida; (b) coerência frouxa, com congruência na saída

Protocolos para Memória Partilhada Distribuída

Como vemos na figura 23a, os protocolos de coerência estrita visam emular o melhor possível o comportamento de um sistema de memória partilhada centralizado. A ordem de operações sobre a memória é total em todos os nós. Semanticamente, esta política é robusta, pois é equivalente a uma memória centralizada. Um modo de a concretizar é através de um protocolo de difusão atómica. No entanto, e sobretudo em redes clássicas (i.e. excluindo malhas ATM) representa uma penalidade em desempenho elevada.

Como já vimos, existem outras soluções, de coerência frouxa, em que se relaxa a semântica de correcção da memória, nomeadamente a congruência das cópias da memória. A figura 23b ilustra um protocolo de congruência na saída. Neste tipo de protocolos a programação sobre a memória partilhada não é totalmente transparente, porquanto se têm que encapsular operações que podem conflituar, numa secção crítica, a *aquisição* e *libertação* de uma zona de memória. Consoante a congruência das cópias é assegurada na entrada da zona, ou na saída, assim o protocolo respectivo se denomina de *congruência na entrada*, ou *na saída* (o caso da figura).

13 Segurança em Sistemas Distribuídos

A segurança em sistemas distribuídos reveste de diversas facetas.

Confidencialidade - salvaguarda de um item ou conjunto de items de informação, do acesso não autorizado, embora não destrutivo. Normalmente através de técnicas de criptografia.

Autenticação - certificação da origem e/ou autenticidade de um item de informação. Normalmente através de técnicas de assinatura criptográfica.

Integridade - salvaguarda de um item ou conjunto de items de informação, do acesso destrutivo. Através de meios físicos, como o isolamento de sistemas, que pode ser inconveniente num sistema distribuído, ou através de meios lógicos, como a replicação ou salvaguarda dos repositórios de dados, a despeito do funcionamento ou não das máquinas.

Disponibilidade - manutenção do acesso a items de informação e/ou da operação de um serviço, a despeito de ataques visando o seu impedimento. Através de muitas

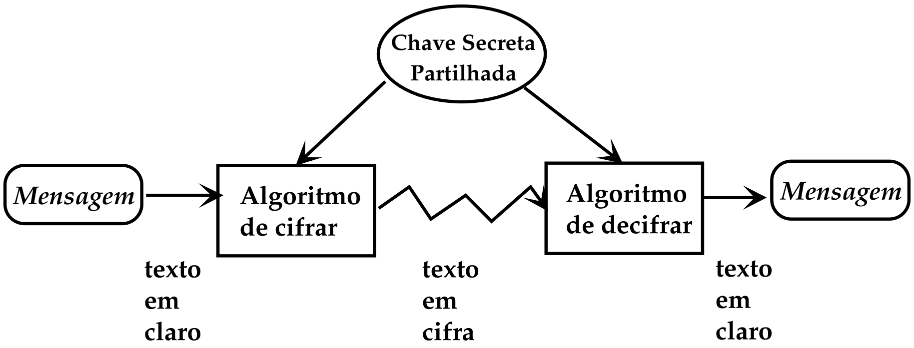


Figura 24: Operação de cifra com chave secreta (DES)

das técnicas visando integridade, com a necessidade adicional de manter o sistema operacional.

Criptografia

A criptografia abrange o conjunto de técnicas destinadas a tratar informação de modo a torná-la ilegível por intrusos, mas possibilitando a sua recuperação posterior pelo(s) legítimo(s) utilizador(es). Dada a insegurança potencial dos sistemas distribuídos, a criptografia assume um papel importantíssimo. De facto, os dados num sistema distribuído dificilmente podem ser postos ao abrigo do escrutínio de um intruso, o qual pode:

- penetrar na máquina através da rede;
- escutar a informação que passa na rede (*eavesdropping*).

Qualquer sistema criptográfico assenta na validade das condições abaixo:

- $Enc(K_1, m) = x$, isto é, cifrando m com chave K_1, obtem-se x;
- $Dec(K_2, y) = n$, isto é, decifrando y com chave K_2, obtem-se n;
- $Dec(K_2, Enc(K_1, m)) = m$, isto é, decifrando o produto da cifra de m, utilizando chaves correspondentes nas duas operações, obtem-se m de volta.
- é computacionalmente difícil inverter o resultado dessas funções sem a chave.

Existem dois tipos de chaves, partilhada ou secreta, e pública.

O sistema de chave **partilhada** é simétrico — isto é, $K_1 = K_2$ — sendo a chave secreta, e partilhada pelos dois interlocutores. É utilizada nas operações de cifrar e decifrar. Uma vez quebrada em qualquer lado, o intruso domina o canal de comunicações e os dados. A figura 24 ilustra uma operação de cifra com chave secreta.

O protocolo de chave secreta mais conhecido é a norma Americana DES, *Data Encription Standard*. Existem alguns sistemas utilizando o DES, como o Kerberos, que é o servidor de segurança do sistema de ficheiros distribuído grande-escala AFS, e o sistema de transacções interactivas Multibanco da SIBS.

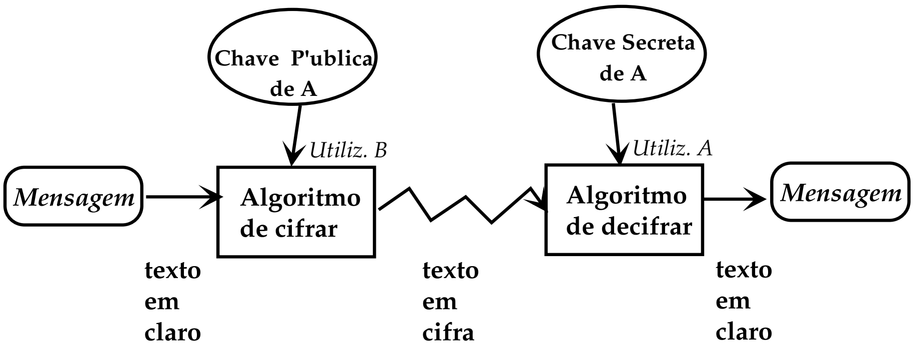


Figura 25: Operação de cifra com chave pública (RSA)

Os sistemas chave **pública**, por outro lado, baseiam-se no método RSA, que consiste em ter duas chaves assimétricas, de cifrar e de decifrar, apenas uma das quais é secreta. O método funciona do seguinte modo: o destinatário, D, cria uma chave de decifrar K que mantém secreta, e gera a partir dela a sua chave de cifrar K^{-1}, que torna pública. O remetente, R, apenas tem que ir a um servidor de chaves públicas, obter a chave de D, cifrar e enviar. O método baseia-se na utilização de funções matemáticas não reversíveis, de modo que, uma vez cifrada, uma mensagem não pode ser decifrada com a chave de cifra, nem um seu derivado. Na verdade, só o pode ser com a chave (secreta) de D. A figura 24 ilustra uma operação de cifra com chave pública.

Este método dá também para autenticação, de uma forma curiosa. Entenda-se 'autenticação' como o método de geração de assinaturas digitais não forjáveis sobre um texto digital (incluindo uma assinatura em ascii), que impeçam a sua modificação. Para o fazer, os papéis invertem-se. Suponhamos que o destinatário da mensagem no exemplo anterior, D, queria enviar uma resposta assinada a R. Escreveria a resposta, e "decifraria" a mensagem. Por estranho que isto pareça, trata-se apenas de aplicar a função *decifrar* ao seu texto, com a sua chave secreta K. R, recebendo a resposta, não tem mais que recuperar a chave pública de cifrar de D, aplicar a função *cifrar*, e obter o texto incólume e assinado. Como anteriormente, tratando-se de uma função não reversível, é impossível a um interceptor modificar o texto em claro e 'decifrá-lo' de novo, de modo a que o receptor não desconfie da falsificação.

Metodologias para Servidores de Segurança

Um servidor de segurança tem como propósito autenticar os utilizadores de um sistema, e controlar o seu acesso às várias partes do mesmo. Existem essencialmente três tipos de servidores de segurança:

- Servidor centralizado de autenticação (ex. Kerberos) — simples de concretizar e operar. PROBLEMA**: acesso em grande-escala; simples ponto de intrusão**;
- Servidores em rede cooperantes (ex. NTCB Red Book) — melhor acesso em grande-escala. PROBLEMA**: têm de confiar uns nos outros, o que os fragiliza**;
- Servidor de segurança distribuído (ex. Delta-4) — servidor descentralizado; necessário quebrar todos para ter sucesso.

14 Conclusão

Nesta lição, procurámos fazer uma síntese, necessariamente superficial, das questões relacionadas com a comunicação e computação em sistemas distribuídos, e as suas condições de correcção, pontualidade e confiabilidade.

As técnicas distribuídas são uma realidade cada vez mais palpável na tecnologia de sistemas de computadores.

Os exemplos de produtos comerciais, *freeware*, ou précomerciais que embarcam algumas das técnicas discutidas aqui são inúmeros:

- redes ATM e sistemas computacionais para aplicações multimédia (Cambridge K-Net AVA-2000).
- redes de computadores para PCs (Novell);
- correio electrónico (Email Internet ou X.400);
- sistemas de ficheiros distribuídos (Sun NFS);
- sistemas de ficheiros distribuídos grande-escala e tolerantes a faltas (Transarc AFS, CODA);
- ambientes de computação cliente-servidor (OSF DCE);
- protocolos de acesso remoto e/ou distribuído a bases de dados homogéneas ou heterogéneas (ISO RDA, SQL/Net, Ingres/Net);
- protocolos de interacção distribuída com dispositivos de automatização (ISO MMS);
- sistemas transaccionais interactivos distribuídos (Transarc Camelot/Encina);
- servidores de informação *wide-area*, *WAIS* (Mosaic WorldWideWebb);
- protocolos de distribuição da visualização para programas baseados em janelas (Sun ToolTalk);
- ferramentas para construção de aplicações fiáveis orientadas para grupos (IDS-Stratus ISIS);
- ferramentas para construção de aplicações fiáveis sobre memória partilhada (Teknekron TIB).

A evolução dos sistemas, do ponto de vista da penetração das tecnologias mais inovadoras, vai ser pautada em grande parte pelos requisitos do utilizador. Vamos assistir à penetração mais fácil das tecnologias que representem um grande saldo custo/benefício em termos de qualidade de serviço final.

A evolução da investigação e do ensino é difícil de prever. As matérias debatidas nesta lição prefiguram uma área que podemos denominar de SISTEMAS DISTRIBUÍDOS CONFIÁVEIS, isto é, uma área sistémica, que reune o conhecimento (teoria, arquitectura e engenharia) relevante para a construção e operação correcta e eficiente de sistemas distribuídos – cada vez mais complexos. Dos tipos de sistemas reais mais relevantes no futuro, onde a actividade de uma tal área poderá deixar rasto, citam-se:

- sistemas embebidos distribuídos (controlo industrial, *desk-area*, domótica);
- sistemas grande-escala — sistemas transaccionais interactivos (OLTP), sistemas de trabalho cooperativo assistido por computador (CSCW), computação móvel;
- algoritmos e protocolos visando melhorar a: congraência, fiabilidade, transparência, pontualidade, segurança;
- ambientes de suporte para desenvolvimento e gestão de aplicações distribuídas grande-escala e confiáveis.

Referências

[Mullender 93] S.J. Mullender, editor. *Distributed Systems, 2nd Edition*. ACM-Press. Addison-Wesley, 1993.

[Powell 91] D. Powell, editor. *Delta-4 - A Generic Architecture for Dependable Distributed Computing*. *ESPRIT Research Reports*, Springer Verlag, November 1991.

[Pimentel 90] *Communication Networks for Manufacturing*, J. Pimentel, Prentice-Hall, 1990.

[Comer 91] *Internetworking with TCP/IP*, D. Comer, Prentice-Hall, 1991.

[Sanders 90] Robert M. Sanders. *The Xpress Transfer Protocol (XTP)- A Tutorial*. Technical Report TR-89-10, University of Virginia, University of Virginia, Charlottesville, Virginia 22903, January 1990.

[Ross 89] Floyd Ross. An Overview of FDDI: The Fiber Distributed Data Interface. *IEEE J. on Selected Areas in Comm.*, 7(7), 1989.

[Hutchinson 90] Norman C. Hutchinson and Larry L. Peterson. *The x-Kernel: An Architecture for Implementing Network Protocols*. Technical Report, University of Arizona, Tucson, USA, 1990.

[Verissimo 92] P. Veríssimo and L. Rodrigues. Group orientation: a paradigm for distributed systems of the nineties. In *3rd Workshop on Future Trends of Distributed Computing Systems*, Taipe, Taiwan, April 1992.

[Birman 94] Kenneth Birman and Robbert van Renesse, editors. *Reliable Distributed Computing With the ISIS Toolkit*. Number ISBN 0-8186-5342-6. IEEE CS Press, March 1994.

[Schwarz 92] R. Schwarz and F. Mattern. *Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail*. Technical Report, Univ. Kaiserslautern, Kaiserslautern, Germany, February 1992.