UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

Suporte à computação orientada aos grupos em sistemas distribuídos tolerantes a faltas

Luís Eduardo Teixeira Rodrigues

(Mestre)

Dissertação para obtenção do grau de doutor em

Engenharia Electrotécnica e de Computadores

Setembro de 1995

Suporte à computação orientada aos grupos em sistemas distribuídos tolerantes a faltas

Luís Eduardo Teixeira Rodrigues

Tese submetida para provas

de doutoramento em

Engenharia Electrotécnica e de Computadores

Departamento de Engenharia Electrotécnica e de Computadores

Instituto Superior Técnico

Lisboa

Setembro de 1995

Este trabalho foi parcialmente financiado pelas seguintes instituições:

FSE/ Ministério da Educação (DGES) / IST
(ao abrigo do programa PRODEP/SEAD)

CE
(ao abrigo do programa ESPRIT/ BRA Project 6360 (Broadcast)
e
ao abrigo do projecto ESPRIT/ BRA Working Group 26 (GODC))

Tese realizada sob a orientação do

Prof. Doutor Paulo Jorge Esteves Veríssimo

Professor Associado com Agregação do Departamento de Informática da
Faculdade de Ciências da Universidade de Lisboa

e co-orientação do

Prof. Doutor Nuno Manuel de Carvalho Ferreira Guimarães

Professor Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores
do Instituto Superior Técnico da Universidade Técnica de Lisboa

Resumo

Esta dissertação contribui para a definição, concepção e desenvolvimento de uma plataforma de grupos capaz de operar sobre *redes de grande escala* geográfica. Estas redes destinguem-se das redes locais principalmente pelas suas limitações em termos de latência, baixa fiabilidade e vulnerabilidade a partições. Assim, justifica-se a necessidade de desenvolver novos serviços e protocolos susceptíveis de oferecer um desempenho aceitável sobre este tipo de redes.

A grande maioria das redes de grande-escala hoje existentes são constituídas por aglomerados de redes locais interligados por redes ponto-a-ponto. Para oferecer comunicação em grupo de modo eficiente sobre este tipo de infra-estrutura, propõe-se uma arquitectura hierárquica, na qual se utilizam protocolos dedicados à rede local em combinação com protocolos orientados a redes interligadas. Dentro desta linha, a dissertação estende e aperfeiçoa resultados anteriores na área da comunicação em grupo sobre redes locais. Para além disso, apresenta soluções inovadoras na área da comunicação em grupo sobre redes de grande escala. Finalmente, como exemplo de aplicação destas tecnologias, a dissertação sugere uma arquitectura genérica para suportar interacções do tipo cliente-servidor, permitindo a invocação remota e fiável de componentes que podem estar replicados. Esta arquitectura permite suportar de um modo integrado diversos protocolos de gestão da replicação.

Abstract

The dissertation contributes to the definition, design and implementation of a group technology platform capable of operating over *large-scale networks*. These networks differ from local-area networks by their limitations in terms of high latency, low reliability and vulnerability to partitions. These characteristics motivate the design of new services offering acceptable performance over these networks.

Most of today's large-scale networks are built of clusters of local-area networks interconnected by long-haul point-to-point links. To offer efficient group communication over this type of infra-structure, the dissertation suggests an hierarchical approach that combines protocols dedicated to local-area networks with protocols dedicated to large-scale networks. In the framework, the thesis extends and improves previous results in the area of local-area group communication. Additionally, new approaches for operation in large-scale are proposed, including algorithms to enforce causal and total order. Finally, to illustrate the use of these technologies, the thesis proposes a generic architecture to support remote invocation of replicated objects. The remote invocation protocol supports the use of different replication strategies for each object. It is believed that such a problem-oriented approach will be a key factor to offer the flexibility and efficiency required by future distributed applications.

Palavras Chave

Sistemas distribuídos de grande-escala

Comunicação em grupo

Tolerância a faltas

Gestão de dados replicados

Difusão fiável

Sincronização de relógios

Invocação remota

Keywords

Large-scale distributed systems

Group communication

Fault-tolerance

Replicated data management

Reliable broadcast

Clock synchronization

Remote invocation

Agradecimentos

Ao Professor Paulo Veríssimo, meu orientador. O seu exemplo, o seu interesse e a sua crítica exigente constituiram um incentivo insubstituível ao longo destes anos.

Ao Professor Nuno Guimarães, que co-orientou este trabalho de modo inexcedível, a partir do momento em que o Prof. Paulo Veríssimo assumiu funções na Faculdade de Ciências.

Para além do Prof. Paulo Veríssimo, vários investigadores colaboraram em diversas fases do trabalho que aqui se apresenta. O Eng. António Casimiro, Eng. Henrique Fonseca, Eng. José Rufino, Eng. Luís Silva e Eng. Werner Wogels participaram no desenvolvimento do xAMp e protocolos associados. O protocolo de ordenação total foi desenvolvido em colaboração com o Eng. Henrique Fonseca. O Protocolo de Invocação Remota foi desenvolvido em colaboração com a Dr.ª Ellen Siegel. No final de cada capítulo, será sempre feita referência aos investigadores que contribuiram para o trabalho descrito.

Devo também agradecer aos restantes elementos do Grupo de Sistemas Distribuídos e Automatização Industrial do INESC, Eng. Carlos Almeida, Eng. Jorge Frazão, Eng. Sérgio Melro e Eng. António Sargento pelo profissionalismo e espírito de camaradagem que sempre demonstraram. Um agradecimento especial ao meu colega de sala, Eng. François Cosquer, pela importância que sempre atribuíu ao meu trabalho.

Aos outros elementos do INESC, particularmente dos grupos de Sistemas Distribuídos, Técnicas de Interacção Multimédia e Engenharia de Software, muitos dos quais também estudantes de doutoramento, com os quais convivi durante estes anos e que contribuiram com a troca de ideias e sugestões para este trabalho: Pedro Antu-

nes, Luís Carriço, Miguel Castro, Adriano Couto, Paulo Ferreira, Paulo Guedes, David Matos, Nuno Neves, João Pereira, José Pereira, Manuel Sequeira, António Silva, Pedro Sousa, Pedro Trancoso e André Zuquete.

Diversos investigadores de outras instituições ofereceram também os seus comentários sobre diferentes partes desta tese. Nomeadamente, agradeço ao Prof. Özalp Babaoğlu, Prof. Ken Birman, Dr. Brad Glade, Dr. Marc Little, Prof. Michel Raynal, Dr. Robert v. Renesse e ao Prof. André Schiper.

A todos os que se prontificaram a ler uma primeira versão desta tese, contribuindo com sugestões e detectando diversas gralhas e omissões.

Um agradecimento adicional ao Eng. David Matos pelo apoio oferecido com o seu profundo conhecimento de LaTeX(e por ter amavelmente cedido os seus excelentes ficheiros de configuração).

Ao INESC, onde desde o final da licenciatura encontrei os meios técnicos e o enquadramento científico sem os quais este trabalho não seria possível. Em particular, um agradecimento ao Prof. Alves Marques que, no final da minha licenciatura, me incentivou a permanecer no INESC e a iniciar o mestrado.

Finalmente, mas não menos importante, ao meus amigos e, sobretudo, à minha mulher, aos meus pais e à minha irmã pelo encorajamento e apoio que sempre me deram.

Lisboa, Setembro de 1995

Luís Eduardo Teixeira Rodrigues

Ao Hugo.

Índice

Lista de Figuras

Lista de Tabelas

1

Introdução

As vantagens dos sistemas computacionais distribuídos são hoje bem conhecidas. A mais evidente motivação para a distribuição reside no carácter inerentemente distribuído das organizações humanas, quer do ponto de vista geográfico quer do ponto de vista organizacional. Outras vantagens importantes das arquitecturas distribuídas são a extensibilidade e escalabilidade. Para além disso, os sistemas distribuídos podem ser construídos de maneira a que os seus componentes tenham modos de falha independentes (num sistema centralizado, geralmente todo o sistema fica inacessível na presença de uma única falha). Esta última característica dá aos sistemas distribuídos o potencial para oferecer *tolerância a faltas*, isto é, a capacidade de continuar a fornecer um serviço correcto na presença de faltas. Infelizmente, esta mesma propriedade torna o desenvolvimento de programas distribuídos uma tarefa bastante complexa (Mullender, 1989).

1.1 Definição do problema e objectivos

O objectivo de fornecer um suporte adequado ao desenvolvimento de programas distribuídos tem sido perseguido pela investigação ao longo das últimas décadas. Existe um vasto número de publicações relatando trabalho em diversas facetas desta área, abarcando (entre outras) sistemas operativos distribuídos (Mullender *et al.*, 1990), sistemas transaccionais (Bernstein *et al.*, 1987), linguagens de programação suportando explicitamente a distribuição (Liskov & Scheifler, 1982), plataformas distribuídas orientadas aos objectos (Sousa *et al.*, 1993; Shrivastava *et al.*, 1991), e sistemas de memória partilhada distribuída (Nitzberg & Lo, 1991).

Tendo um campo restrito de aplicação nos sistemas distribuídos pioneiros, o suporte para tolerância a faltas, em particular usando técnicas de replicação[1], teem vindo a atrair a atenção dos investigadores nos últimos anos. Diversas soluções têm sido propostas para suportar eficazmente a replicação, nomeadamente: suporte para a replicação de objectos através de invocações remotas (e chamadas a procedimentos remotos), como os "bandos" (*troupes*) (Cooper, 1984), ou o DELTA-4 (Powell, 1991)); sistemas transaccionais, tais como o TABS (Spector *et al.*, 1985)/CAMELOT (Bloch, 1989) ou o ARGUS (Liskov & Scheifler, 1982)); ou sistemas operativos, como por exemplo o Clouds (Dasgupta *et al.*, 1988) ou o Amoeba (Mullender *et al.*, 1990); etc. Estes sistemas suportam replicação com o intuito de oferecer tolerância a faltas (aumentando a disponibilidade e a fiabilidade na presença de faltas) e aumentar o desempenho (explorando a localidade). Infelizmente, diminuir o tempo de acesso aos recursos, protegê-los contra falhas, e garantir a coerência da informação replicada são objectivos geralmente contraditórios, e cada aplicação poderá privilegiar em diferentes graus cada um destes aspectos.

A maioria dos sistemas que suportam replicação oferece um número limitado de protocolos para gestão da mesma. Frequentemente, apenas se encontra disponível um único protocolo, o que não permite uma aproximação orientada para o problema tal como defendida em (Cheriton, 1992) (a qual sugere que diferentes métodos de replicação devem ser aplicados a diferentes objectos, de acordo com as características particulares dos mesmos). Considera-se que uma aproximação orientada para o problema é um factor chave para oferecer a flexibilidade e a eficiência requeridas pelas aplicações distribuídas do futuro. Em muitos dos sistemas hoje disponíveis, o suporte à replicação encontra-se integrado no núcleo, dando poucas possibilidades ao programador para adaptar os protocolos às suas necessidades específicas ou para adicionar novos protocolos ao sistema. Apesar de existirem algumas excepções notáveis, nomeadamente o sistema ISIS (Birman & Joseph, 1987) ou o modelo FOG (Makpangou *et al.*, 1992), consideramos que existe uma lacuna no suporte ao refinamento da gestão da replicação. Isto motiva o trabalho apresentado:

[1] *Replicação* e *réplica* são termos usados nesta dissertação com o sentido de *reprodução* e não com o sentido de contestação. Termo usado em português também com este sentido noutras áreas, por exemplo, "réplica de um sismo".

A Tese estuda mecanismos de suporte ao desenvolvimento e uso de objectos replicados que permitam ao utilizador escolher — e aperfeiçoar — a solução mais adequada ao problema que pretende resolver.

A gestão da replicação coloca necessariamente diversos requisitos aos serviços de suporte, com especial ênfase ao suporte fornecido pela infra-estrutura de comunicação. Entre outros, alguns dos requisitos de um sistema que suporta replicação são: a necessidade de difundir informação para as diferentes réplicas de modo fiável, a necessidade de tolerar falhas de nós e partições na rede, a necessidade de manter informação coerente e precisa acerca da filiação no grupo de réplicas, etc. Recentemente, diversos paradigmas têm sido propostos com o objectivo de lidar com estes problemas. Entre estes, o uso de *grupos de difusão* como um mecanismo estruturante para a construção de aplicações distribuídas tem ganho particular aceitação. Esta aproximação, por vezes apelidada de "orientação para os grupos" (Birman *et al.*, 1991b; Veríssimo & Rodrigues, 1992a), reside no uso intensivo de "tecnologia de grupos", isto é, protocolos de filiação e difusão em grupo (em particular, no uso de difusão fiável em grupo, com diferentes propriedades de acordo e ordem). Chamam-se ao conjunto de maquinaria e de programas que fornecem estes serviços uma *plataforma de grupos*. Apesar da "orientação para os grupos" ter vindo a ganhar um interesse crescente, existe ainda um aceso debate acerca das suas vantagens relativamente a outras aproximações alternativas (Cheriton & Skeen, 1993; Birman, 1994). Este debate oferece uma segunda motivação para o trabalho aqui apresentado:

A Tese contribui para a avaliação da "orientação para os grupos" como paradigma estruturante para o desenvolvimento de sistemas de gestão da replicação. Para tal, a tese desenvolve soluções baseadas na utilização de plataformas de grupos.

Naturalmente, qualquer solução abrangente para o suporte à replicação deve ter a capacidade de operar em sistemas de grande escala geográfica. Actualmente, a grande maioria das plataformas de grupos foram concebidas para oferecerem o seu desempenho óptimo sobre redes locais. Existe pois a necessidade de desenvolver plataformas de grupos com a capacidade de operarem eficazmente em redes de grande escala. Deste modo:

A Tese contribui para a definição, concepção e desenvolvimento de uma plataforma de grupos capaz de operar sobre redes de grande escala geográfica.

1.2 Resultados

Esta tese enquadra-se num projecto de investigação levado a cabo pelo Grupo de Sistemas Distribuídos e Automatização Industrial do INESC, globalmente designado por NAVTECH /Romance. Por sua vez o NAVTECH /Romance enquadra-se em projectos internacionais de âmbito mais alargado, como os projectos Broadcast[2], GODC[3]. Naturalmente, os objectivos destes projectos são ambiciosos e almejados a longo prazo.

Esta dissertação contribui para este esforço, oferecendo soluções para alguns dos problemas encontrados do desenvolvimento das arquitecturas NAVTECH e Romance. Nomeadamente, esta dissertação apresenta os seguintes resultados:

- um conjunto de protocolos de filiação em grupo, comunicação em grupo e sincronização de relógios orientados à rede local;
- um conjunto de protocolos de comunicação em grupo orientados à rede de grande escala;
- um protocolo de invocação remota de componentes replicados.

1.3 Contribuição

As principais contribuições do trabalho apresentado nesta tese, e substanciado nos resultados anteriores, são as seguintes:

[2]Broadcast, Basic Research On Advanced Distributed Computing: from Algorithms to SysTems. European Strategic Programme for Research in Information and Technology (ESPRIT) Basic Research Project 6360. Consórcio constituido pelos seguintes parceiros: Ecole Polythecnique Fédérale de Lausanne (CH); INESC, Lisboa (P); INRIA, Rocquencourt (F), INRIA-IRISA, Rennes (F); Università di Bologna (I); Université Joseph Fourier (IMAG), Grenoble (F); The University of Newcastle upon Tyne (UK); Universiteit van Twente (NL).

[3]GODC, Group Oriented Distributed Computing. European Strategic Programme for Research in Information and Technology (ESPRIT) Basic Research Working Group 26. Consórcio constituido pelo INESC (P) e pela Cornell University (USA).

- *estende e aperfeiçoa resultados anteriores na área da comunicação em grupo sobre redes locais*, nomeadamente, demonstra que as propriedades destas redes podem ser utilizadas para fornecer primitivas de filiação e comunicação em grupo de baixa latência e serviços de sincronização de relógios de elevada precisão;

- *apresenta soluções inovadoras na área da comunicação em grupo sobre redes de grande escala*, nomeadamente, apresenta novas qualidades de serviço, um protocolo de ordenação causal que utiliza informação acerca da topologia da rede para reduzir informação de controlo agregada às mensagens, e um protocolo de ordenação total que em ambientes heterogéneos reduz a latência na entrega das mensagens;

- *sugere uma arquitectura genérica para suportar invocação remota, a qual é independente da estratégia de replicação*, em particular demonstra-se que através da composição dos serviços anteriormente referidos é possível suportar diferentes algoritmos de gestão da replicação.

1.4 Estrutura da tese

No Capítulo 2 o problema é enquadrado. Em primeiro lugar, faz-se uma panorâmica sobre os sistemas de gestão da replicação. Seguidamente, faz-se uma panorâmica do estado do conhecimento na área das plataformas de grupos. Finalmente, dedica-se alguma atenção à arquitectura Delta-4, a qual oferece uma solução integrada para o problema de suportar tolerância a faltas em sistemas distribuídos.

Com o intuito de fornecer suporte a computação fiável em redes de grande escala, o Grupo de Sistemas Distribuídos e Automatização Industrial encontra-se a desenvolver uma arquitectura orientada para os grupos, denominada NAVTECH. O Capítulo 3 descreve as características principais desta arquitectura. Como se verá, a arquitectura NAVTECH é uma arquitectura modular. A presente dissertação apresenta soluções para alguns dos módulos desta arquitectura. Os serviços fornecidos pela arquitectura NAVTECH podem ser expandidos através da utilização de um sistema de suporte à replicação. No âmbito desta tese, iniciou-se a concepção de uma arquitectura de

suporte à replicação seguindo uma aproximação orientada para o problema, a qual é também introduzida neste capítulo.

No Capítulo 4 descrevem-se os componentes orientados à rede local da arquitectura NAVTECH . São identificadas as características específicas das redes locais e apresentam-se diversos módulos orientados para este tipo de suporte. Destes módulos salientam-se: um protocolo de filiação de nós, denominado MGS; um protocolo multi-primitiva de filiação e comunicação em grupo, denominado xAMp; e um protocolo de sincronização de relógios. Estes módulos foram obtidos a partir da expansão e depuramento de versões anteriormente desenvolvidas.

Os componentes orientados às redes de grande escala da arquitectura NAVTECH são descritos no Capítulo 5. O capítulo faz uma análise das particularidades deste tipo de redes, identificando as diferenças em relação à comunicação em rede local. Seguidamente, descrevem-se os componentes totalmente desenvolvidos no âmbito desta Tese: um protocolo para ordenação causal de mensagens; dois mecanismos que dão ao sistema uma maior versatilidade, designados respectivamente por *mensagens transparentes* e por *mensagens retidas*; e finalmente, um protocolo para ordenação total. Estes serviços foram especialmente concebidos tendo em consideração as características das redes de grande escala.

No Capítulo 6 descreve-se um mecanismo de invocação remota fiável. O protocolo apresentado neste capítulo suporta de modo tolerante a faltas a invocação de serviços replicados, oferecendo não só a usual transparência à localização, mas também transparência aos mecanismos de replicação. O protocolo, sendo um componente do sistema Romance, faz uso dos serviços apresentados nos capítulos anteriores, nomeadamente dos serviços da arquitectura NAVTECH .

Finalmente, o Capítulo 7 conclui a tese e apresenta perspectivas futuras.

2

Panorâmica

2.1 Enquadramento

Nesta tese estudam-se modelos e mecanismos de suporte à computação distribuída e tolerante a faltas em sistemas de grande escala. Este capítulo motiva a aproximação tomada e refere soluções relacionadas. Atendendo ao seu carácter abrangente, e dada a complexidade dos problemas focados, a panorâmica não descreve as diferentes alternativas em pormenor. Optou-se antes por oferecer aqui uma vista de conjunto, relegando-se o estudo pormenorizado dos aspectos directamente relacionados com as soluções defendidas na tese para os capítulos em que estas últimas são apresentadas.

O capítulo está estruturado do seguinte modo: Com o objectivo de motivar a necessidade das soluções apresentadas, faz-se uma breve panorâmica sobre uma área que possui enorme relevância na computação distribuída e tolerante a faltas: a gestão de dados replicados. Seguidamente, introduzem-se os serviços de filiação e comunicação em grupo. Alguns destes serão retomados e estudados posteriormente. Finalmente, foca-se o sistema Delta-4. Embora tenha sido desenvolvido para operar sobre redes locais, este sistema constitui uma arquitectura de referência na área das soluções integradas para o suporte à computação distribuída e tolerante a faltas.

2.2 Gestão de dados replicados

Existem diversos dispositivos com o potencial para armazenar informação, os quais podem ser caracterizados por diferentes propriedades, tais como capacidade de arma-

zenamento, velocidade de acesso, volume, consumo, volatilidade, fiabilidade, etc.. Geralmente, um sistema possui diversos dispositivos de armazenamento de informação interligados entre si. Alguns exemplos: um processador pessoal possui diversos níveis de memória, incluindo disco, memória volátil e "cache" do processador; uma rede de estações de trabalho possui um conjunto de discos (e de memórias voláteis) interligados que podem ser partilhados; uma corporação possui diversas máquinas afastadas e interligadas por uma rede de grande escala. Nestes sistemas, é possível distribuir (e eventualmente replicar) a informação pelos diversos dispositivos interligados.

Diz-se que a informação está replicada quando existem diversas cópias do mesmo item em dispositivos diferentes. A replicação tem dois objectivos, a saber: melhorar o *desempenho* e fornecer *tolerância a faltas*. O desempenho pode ser melhorado quando se permite que os clientes acedam a dispositivos mais "próximos" (em tempo de acesso) ou menos sujeitos a congestão. A tolerância a faltas permite aumentar a disponibilidade (*availability*) e fiabilidade (*reliability*) dos dados na presença de faltas temporárias ou permanentes nos dispositivos (processador, memória, ou comunicação).

As vantagens mencionadas são contra-balançadas pela necessidade de maior capacidade de armazenagem e pelos custos inerentes à sincronização, necessários para manter a coerência da informação. De um modo informal, diz-se que um sistema de memória oferece um serviço coerente se, com o seu comportamento, oferece ao utilizador informação congruente com as expectativas deste. O comportamento de um sistema de memória é formalmente descrito por um modelo de coerência (*consistency model*). Num sistema suportando replicação, um algoritmo que permite satisfazer um modelo de coerência é designado por um *algoritmo de gestão de dados replicados*. Nas secções seguintes será feita uma breve panorâmica sobre alguns modelos de coerência e algoritmos para os satisfazer.

2.2.1 Modelos de coerência

Assuma-se que o sistema de memória é constituído por um conjunto de objectos caracterizados por um estado interno encapsulado por um conjunto de operações. Por simplicidade, a maioria dos modelos só consideram duas operações: *leitura*, que

devolve o estado do objecto; e *escrita*, que actualiza esse estado. Durante a sua execução, cada processo invoca uma sequência de operações sobre um ou mais objectos. Uma execução do sistema é representada por uma *história parcialmente ordenada* das operações invocadas por todos os processos. Um modelo de coerência de memória define quais a histórias que são consideradas correctas e como tal admissíveis.

2.2.1.1 Modelos genéricos

O modelo de memória mais intuitivo é o modelo de memória designado por *coerência atómica*, também denominado por linearizabilidade (*linearizability*) (Herlihy & Wing, 1990). De acordo com este modelo, as operações de escrita estão totalmente ordenadas respeitando a ordem (em tempo real) da execução das operações e as operações de leitura devolvem sempre o estado mais recente do objecto. Caso se enfraqueça este modelo, de modo a que não seja necessário respeitar a ordem temporal das invocações, obtém-se o modelo de memória originalmente proposto por Lamport (Lamport, 1979), designado por coerência sequencial (*sequencial consistency*).

Os modelos anteriores são também designados por modelos de coerência forte (*strong consistency*). Apesar de corresponderem às expectativas do programador (são intuitivos e fáceis de utilizar), estes modelos são de concretização dispendiosa uma vez que requerem o estabelecimento de uma ordem total sobre as operações de acesso à memória (o que, como se verá, não é trivial quando a memória se encontra distribuída e replicada). Com o objectivo de obter maior desempenho, foram propostos modelos alternativos.

O modelo de coerência causal (*causal consistency*) (Ahamad *et al.*, 1991), é um modelo de coerência que preserva as relações causais (Lamport, 1978) entre as operações. Operações que não são relacionadas causalmente podem ser observadas por processos diferentes em ordens diferentes. Este modelo é particularmente eficiente para aplicações em que os processos partilham informação mas não executam escritas concorrentes sobre os mesmos objectos.

Um modelo ainda mais fraco pode ser obtido respeitando apenas a ordem das actualizações executadas por um mesmo processo, mas não se ordenando escritas de

processos diferentes (mesmo que relacionadas causalmente). Este último modelo é denominado por *coerência por processo*[1] (Goodman, 1989). Este modelo possui a vantagem de permitir atrasar os efeitos de uma operação de escrita de modo a que esta seja agregada a outras operações de escrita posteriores, aumentando assim o desempenho do sistema.

Uma outra alternativa para definir modelos de coerência de concretização mais eficiente consiste em classificar os acessos à memória em diferentes categorias. O modelo de coerência híbrida (*hybrid consistency*) (Attiya & Friedman, 1992) distingue operações *fortes* e operações *fracas*. O modelo garante que as operações fortes são executadas numa ordem sequencial. Para além disso, se um mesmo processo executa duas operações e uma delas é forte, estas são ordenadas pela ordem de invocação. No entanto, os processos não necessitam de acordar a ordem das operações fracas que são executadas entre duas operações fortes. Geralmente, as operações fortes são associadas a mecanismos de sincronização. Isto permite que uma aplicação possa observar um comportamento sequencialmente coerente desde que utilize os mecanismos de sincronização de modo apropriado. Um refinamento deste modelo foi designado por coerência na saída (*release consistency*) (Gharachorloo *et al.*, 1990), em que as operações são classificadas em *aquisição*, *libertação* e *não-sincronizadas*. Um modelo aproximado, designado por coerência na entrada (*entry consistency*) (Bershad & Zekauskas, 1991), requer que todas as variáveis partilhadas sejam associadas a uma variável de sincronização.

2.2.1.2 Modelos semânticos

Os modelos anteriores definem coerência de memória de um modo genérico e independente de pormenores de concretização. Em particular, o utilizador não necessita de estar ciente da eventual distribuição ou replicação da informação. É possível definir modelos de coerência que tornam estes atributos visíveis. De um modo geral, estes modelos são específicos de determinadas aplicações e dependentes da semântica destas, pelo que não são de aplicação genérica. A vantagem destes modelos é que

[1] Originalmente definido no âmbito de arquitecturas multi-processador e designado por coerência por processador (*processor consistency*).

permitem por vezes significativos ganhos no desempenho, sobretudo nas aplicações em que os utilizadores estão preparados para lidar com informação (controladamente) incoerente.

O mais comum destes modelos é talvez o modelo *mestre-cópias*. De acordo com este modelo, cada objecto possui várias cópias e uma delas é designada como cópia *mestre*. Esta cópia mantém o estado correcto do objecto. Todas as actualizações são realizadas na cópia mestre. Estas actualizações não necessitam de ser imediatamente propagadas para as restantes cópias, as quais podem armazenar informação desactualizada. Usualmente, as diferenças entre a cópia mestre e as restantes estão limitadas por um dado critério, dependente da aplicação. Por exemplo, um dado critério pode garantir que as cópias são actualizadas pelo menos uma vez por cada intervalo x de tempo real (Rusinkiewick *et al.*, 1991).

Uma generalização deste modelo, permite que actualizações possam ser feitas de modo independente em cópias diferentes. Estas actualizações podem ser coordenadas de modo a garantir, tal como anteriormente, que as diferenças entre as cópias são limitadas. Alternativamente, é possível permitir actualizações sem qualquer coordenação, e deste modo, permitir que as cópias possuam conteúdos divergentes. Estes modelos permitem uma maior disponibilidade da informação em situações em que a coordenação é impossível (por exemplo, quando as cópias estão mutuamente inacessíveis). Assim que possível, as cópias divergentes devem ser reconciliadas, isto é, colocadas num estado simultaneamente coerente e congruente com as operações efectuadas em cada cópia. Infelizmente, não existe solução genérica para o problema da reconciliação e é necessário recorrer a algoritmos que dependem da semântica das operações. Em muitos casos, não existem soluções determinísticas para o problema sendo necessário recorrer à reconciliação manual (isto é, controlada pelo utilizador final).

2.2.1.3 Transacções e controlo da concorrência

Um requisito habitual das aplicações ainda não coberto pelos modelos anteriores é a necessidade de agrupar uma sequência de operações e especificar que essa sequência se deve executar de um modo atómico. Este aspecto é coberto pelo conceito de *transacções*

(*transactions*) (Bernstein *et al.*, 1987). Muitos dos programas que acedem a informação distribuída estão hoje estruturados usando modelos transaccionais. Nos próximos parágrafos abordam-se de modo conciso os aspectos relacionados com a gestão das transacções (Dietzen & Spector, 1992).

Transacções De um modo geral, transacções são sequências de operações sobre dados, usualmente distinguidas entre *escritas* e *leituras*, conjuntamente com passos de computação. Estas sequências são geralmente delimitadas por instruções tais como `inicio-de-transacção` e `fim-de-transacção`. As transacções são caracterizadas pelas seguintes propriedades (designadas na literatura anglo-saxónica por propriedades ACID[2]): atomicidade (*atomicity*), coerência (*consistency*), isolamento (*isolation*), e durabilidade (*durability*).

- A *atomicidade* refere-se ao facto de uma transacção ser tratada como uma unidade operacional. Deste modo, ou todas as operações são executadas com sucesso ou nenhuma operação deve ser executada. Se uma transacção é interrompida por uma falta, todas as operações executadas até então devem ser anuladas (a transacção pode ser posteriormente resubmetida para execução). Uma transacção cujas operações são executadas com sucesso diz-se confirmada (*committed*). Uma transacção cujas operações são anuladas, diz-se cancelada (*aborted*).

- A *coerência* e o *isolamento* referem-se à correcção dos resultados da transacção, em particular quando esta é executada em paralelo com outras transacções. A coerência define-se em relação à especificação semântica dos dados acedidos pela transacção. O isolamento define que uma transacção não observa os resultados de outras transacções concorrentes, a não ser que estas venham a ser confirmadas e serializadas no seu passado.

- A *durabilidade* garante que os resultados de uma transacção confirmada são permanentes e não podem ser apagados (excepto na eventualidade de acontecimentos catastróficos). Os sistemas podem ser concebidos de modo a reduzir ao mínimo o

[2]Do Inglês, "**A**tomicity, **C**onsistency, **I**solation, and **D**urability".

risco de catástrofes. De facto, o recurso à replicação é uma das técnicas disponíveis para atingir este objectivo.

Para concretizar um sistema transaccional é necessário suportar um conjunto de funcionalidades. Por exemplo, para poder anular as operações executadas por uma transacção é necessário manter um histórico (*log*) das operações efectuadas (existem diversas técnicas para este efeito que não serão aqui focadas). Outra funcionalidade fundamental é um serviço que permita aos participantes de uma transacção acordar acerca da sua confirmação ou cancelamento. Este serviço pode ser concretizado por um protocolo de confirmação atómica (*atomic commit*).

Transacções distribuídas Uma vez que tanto os participantes como os dados se podem encontrar distribuídos, o protocolo de confirmação atómica deve executar um algoritmo distribuído e, desejavelmente, tolerante a faltas. O protocolo usado com mais frequência para este fim é um protocolo de confirmação em duas-fases (*two-phase commit*). De acordo com este protocolo, um processo age como coordenador e, numa primeira fase, recolhe informação de todos os participantes e toma uma decisão acerca da confirmação ou cancelamento da transacção, a qual dissemina numa segunda fase. Este protocolo bloqueia no caso de falha do coordenador. Infelizmente, prova-se que num sistema assíncrono sujeito a falhas na comunicação, não existem soluções não bloqueantes para o problema do confirmação atómica (Skeen, 1982).

É no entanto possível arranjar soluções que diminuem a probabilidade de bloqueio (por exemplo, que garantem o progresso desde que uma maioria dos processos se encontre mutuamente acessível). Este tipo de protocolos requerem geralmente uma fase adicional e são, por isso, designados por protocolos de confirmação em três-fases (*three-phase commit*) (Skeen & Stonebraker, 1983). Esta fase executa-se após a tomada de decisão, e atrasa a confirmação da transacção até que pelo menos uma maioria dos participantes declare ter conhecimento da decisão.

Controlo da concorrência Naturalmente, para ter aplicabilidade prática, um sistema transaccional deve permitir a execução concorrente de várias transacções, garantindo

que o resultado dessa execução satisfaz um determinado modelo de coerência. O modelo de coerência mais usado nos sistemas transaccionais é a serializabilidade (*serializability*), o qual pode ser definido do seguinte modo: a execução concorrente de várias transacções é *serializável* se produz os mesmos resultados que uma execução em série dessas transacções. Este modelo permite que os acessos à memória sejam efectuados em *qualquer* ordem, desde que o seu efeito seja o mesmo que *uma dada* ordem série.

Este modelo de coerência pode ser concretizado através de mecanismos de controlo da concorrência tais como: grafos de serialização, trincos (*locks*), ou estampilhas temporais (*timestamps*). Estes mecanismos podem ser aplicados através de políticas *optimistas* ou *pessimistas*. Uma política pessimista assume que a maioria das transacções vão concorrer sobre os mesmos dados e tenta sincronizá-las o mais cedo possível durante a sua execução. Uma política optimista assume que os conflitos são raros e só estabelece a sincronização no final da execução das transacções. Um estudo aprofundado do controlo da concorrência está fora do âmbito desta tese (o leitor interessado pode consultar, por exemplo, Bernstein *et al.* (1987)).

Outros critérios Tal como a coerência sequencial, no caso de operações não agrupadas, a serializabilidade é um modelo de coerência intuitivo e simples de compreender. Infelizmente, as mesmas propriedades que a tornam atractiva também a tornam limitativa para aplicações com os seguintes requisitos (Barghouti & Kaiser, 1991):

- *Transacções de longa duração*. Quando as transacções possuem elevados tempos de execução, a garantia da propriedade de isolamento impede o progresso eficiente do sistema. Por exemplo, uma transacção pode deter o trinco sobre um determinado objecto durante toda a sua execução, impedindo o accesso ao mesmo por parte das restantes transacções.

- *Cooperação*. Por vezes é útil tornar resultados intermédios visíveis a outras transacções. Por exemplo, duas transacções podem necessitar de trocar informação. Neste caso, é a própria propriedade de isolamento que não se adequa aos requisitos da aplicação.

Para satisfazer estes requisitos é necessário procurar modelos de coerência mais fracos e mecanismos alternativos para os concretizar. Para uma panorâmica, veja-se Barghouti e Kaiser (1991).

2.2.2 Algoritmos

Nesta secção descrevem-se diversos algoritmos que podem ser usados para satisfazer modelos de coerência de memória. Focam-se principalmente algoritmos que podem ser usados em sistemas transaccionais usando a serializabilidade como modelo de coerência.

2.2.2.1 Factores de influência

Antes de se referirem os algoritmos, enumeram-se de modo abreviado alguns dos mais importantes factores que necessitam de ser tomados em conta ao conceber um algoritmo para gestão de dados replicados. Distinguem-se dois tipos de factores: *factores operacionais* e *critérios de desempenho*.

Dentro dos factores operacionais, sublinham-se os seguintes:

- *Tipos de faltas*. Um aspecto crucial na concepção de um algoritmo é o tipo de faltas que este deve tolerar. Nalgumas áreas de aplicação, em que as faltas são infrequentes, a tolerância a faltas é um aspecto secundário. Noutras, a tolerância a faltas é o único propósito da replicação.

  Neste contexto, um tipo de faltas particularmente pernicioso é uma *partição* na rede (para uma taxonomia dos tipos de faltas veja-se, por exemplo, Veríssimo (1989)). Uma partição ocorre quando o sistema fica dividido em grupos de processos isolados. Os processos numa partição podem comunicar entre si mas não conseguem comunicar com nenhum nó noutra partição. Como não é possível garantir a coerência dos objectos através de partições, geralmente permite-se o progresso apenas numa partição, designada por *partição seleccionada*. Para tal, é necessário identificar esta partição. Infelizmente, como se verá mais adiante, num

sistema assíncrono é impossível distinguir uma partição de um elo (ou processo) lento, ou mesmo de um processo falhado. Um algoritmo que fornece aos participantes informação acerca de quais os nós que se encontram acessíveis designa-se por um algoritmo de *filiação*. Este é um serviço chave que deve ser oferecido pelos ambientes de suporte.

- *Atrasos na comunicação*. O custo da comunicação (sobretudo em termos de latência) é fundamental para o desempenho dos algoritmos. Na presença de uma rede lenta deve-se tentar minimizar a comunicação entre processos.

- *Padrões de acesso*. A razão entre a frequência das operações de leitura e escrita é também um factor importante. Alguns algoritmos privilegiam as operações de escrita, outros operações de leitura. De um modo geral, quanto maior for esta razão mais eficiente será a replicação da informação uma vez que menos actualizações (proporcionalmente) necessitam de ser feitas.

O algoritmo de gestão da replicação deve atender às condições atrás enunciadas de modo a satisfazer um determinado critério de desempenho. Listam-se alguns dos critérios mais relevantes:

- *Latência e débito.* Em muitas aplicações a latência no acesso à informação é um aspecto crítico. Em sistemas com características de tempo-real, mesmo que no sentido lato, este factor é ainda mais importante. No entanto, a latência é por vezes preterida para beneficiar o débito uma vez que estes dois requisitos são frequentemente contraditórios. Por este motivo, *é importante que o ambiente de suporte ofereça primitivas de comunicação, nomeadamente primitivas de difusão, que minimizem a latência no acesso à informação*.

- *Confiança no funcionamento.* Do ponto de vista de tolerância a faltas, o desempenho de algoritmos de gestão de dados replicados é medido considerando a confiança no funcionamento por estes oferecida. Medidas comuns de confiança no funcionamento são a fiabilidade, definida como a probabilidade de um determinado objecto permanecer continuamente acessível por um período de tempo,

e a disponibilidade, definida como a probabilidade de um objecto se encontrar acessível num determinado instante.

- *Custos de armazenamento.* A maioria dos algoritmos de gestão da replicação necessitam de armazenar informação de controlo conjuntamente com os dados. A sobrecarga imposta por esta informação adicional deve ser minimizada.

- *Complexidade*. Alguns algoritmos são bastante simples de concretizar. Outros requerem serviços auxiliares, que podem ser de concretização mais complexa. As possíveis vantagens obtidas de acordo com os restantes critérios devem ser ponderadas em relação ao custo da concretização. Por este motivo, *é importante que o sistema de suporte ofereça — de um modo modular e eficiente — os serviços auxiliares necessários para satisfazer os critérios anteriores*.

2.2.2.2 Componentes

Um algoritmo de gestão da replicação possui dois componentes inter-relacionados (Agrawal & Abbadi, 1990): *controlo da concorrência* e *controlo de cópias*. O controlo da concorrência é responsável por satisfazer o modelo de coerência na presença de acessos concorrentes. Na presença de replicação, o acesso aos dados pode ter de ser executado em mais que uma réplica. Se dois acessos concorrentes são feitos a dois conjuntos de réplicas cuja intersepção é não nula, diz-se que os acessos se *interseptam*. Usualmente, a intersepção ocorre em pelo menos uma cópia, de modo a que essa cópia possa verificar a compatibilidade das operações através de um mecanismo de controlo da concorrência local. Quando se sabe de antemão que duas operações são compatíveis, este requisito pode ser enfraquecido. Por exemplo, apesar de operações de leitura se deverem interseptar com operações de escrita, não é necessário que duas operações de leitura se interseptem. O controlo de cópias é responsável por garantir que os resultados da última operação de escrita podem ser identificados e observados. Nem todas as cópias necessitam de manter os valores actualizados desde que os utilizadores obtenham os valores desejados. Geralmente, o controlo de cópias é feito numerando versões. Deve-se sublinhar que estes componentes são complementares, uma vez que não basta manter as cópias com

o mesmo valor para que a informação seja congruente (por exemplo, permitir dois levantamentos do mesmo valor em duas cópias diferentes da mesma conta bancária, deixa as cópias mutuamente coerentes mas incongruentes).

Nas secções seguintes serão descritos alguns algoritmos de gestão da replicação. Será dado ênfase a algoritmos que podem ser utilizados em sistemas transaccionais seguindo a serializabilidade como modelo de coerência. Estes algoritmos são designados por algoritmos que mantêm equivalência a uma cópia (*1-copy equivalence*), uma vez que tornam a replicação transparente. Estes algoritmos podem ser classificados em duas categorias principais, a saber: algoritmos *não-adaptativos* (ou *estáticos*) e algoritmos *adaptativos* (ou *dinâmicos)*. Os algoritmos estáticos são configurados durante a iniciação do sistema e não alteram o seu funcionamento no caso de ocorrência de faltas. Estes algoritmos são simples e pouco exigentes em termos de ambiente de suporte, mas são geralmente pouco eficientes. Os algoritmos dinâmicos são capazes de se adaptar a mudanças no ambiente, tais como falhas ou partições, sendo significativamente mais eficientes mas exigindo um suporte sofisticado (Agrawal & Abbadi, 1990), tal como a capacidade de estabelecer acordo acerca de quais os componentes mutuamente acessíveis.

2.2.2.3 Algorimos estáticos

Uma maneira de garantir que as operações se interceptam em pelo menos uma cópia é associar a cada operação um número mínimo de cópias, ou *quorum*, em que a operação deve ser executada. Para garantir a intersepção, a atribuição de *quorums* deve assegurar que a soma de dois *quorums* associados a operações incompatíveis é superior ao número total de cópias. O exemplo mais simples de um algoritmo de *quorum* é o algoritmo em que as operações de escrita devem ser feitas em todas as réplicas e as operações de leitura apenas numa réplica. Este último algoritmo permite que se executem operações de leitura caracterizadas por elevada disponibilidade e baixo custo. Por outro lado, este algoritmo restringe a disponibilidade das operações de escrita, que deixam de poder ser executadas assim que uma cópia fica inacessível.

Votação Uma extensão ao esquema anteriormente descrito consiste em atribuir a cada cópia um valor designado por *voto*. Os *quorums* são definidos tendo por base o número de votos em vez do número de cópias e a condição para assegurar a intersepção consiste em requerer que a soma dos *quorums* de operações incompatíveis exceda o número de votos no sistema. Esta técnica constitui a base dos algoritmos designado por votação maioritária (*majority voting*) (Thomas, 1979) e votação ponderada (*weighted voting*) (Gifford, 1979). Uma atribuição criteriosa dos votos, tendo em consideração as propriedades de cada cópia (por exemplo uma cópia pode residir numa máquina mais fiável ou com maior conexidade) permite aumentar o desempenho do sistema.

Coutadas Uma outra alternativa consiste em indicar explicitamente quais os grupos de cópias, ou *grupo de quorum*, em que cada operação se deve executar para poder ser confirmada. Uma lista explícita destes grupos de cópias designa-se por *conjunto de quorum*. Para garantir a intersepção, cada grupo de *quorum* (do conjunto associado a uma dada operação) deve-se interseptar com todos os grupos de *quorum* nos conjuntos associados a operações incompatíveis. Conjuntos de *quorum* com estas características são designados por coutadas (*coteries*) (Garcia-Molina & Barbara, 1985). A razão para indicar conjuntos de *quorum* explicitamente deve-se ao facto de existirem coutadas que não podem ser descritas através de atribuição de votos. A utilização de conjuntos de *quorum* possui algumas limitações. Em primeiro lugar, a manipulação e comparação dos grupos de *quorum* é dispendiosa, quer em termos de armazenagem, quer em termos de processamento. Por outro lado, o número de coutadas que podem ser definidas para um determinado problema pode ser bastante vasto, tornando-se extremamente complexo encontrar a coutada mais adequada à resolução de um determinado problema. Por este motivo foram procuradas representações alternativas.

Representações estruturais Um dos métodos para representar conjuntos de *quorum* é recorrer a estruturas lógicas, tais como árvores, grelhas (*grids*), etc. Citam-se alguns exemplos. O algoritmo de *quorum em árvore* (Agrawal & Abbadi, 1991) organiza logicamente as cópias numa árvore binária. O algoritmo tenta obter um *quorum* seleccionando um caminho a partir da raiz até uma das folhas. Se nenhum destes caminhos for encon-

trado devido à inacessibilidade de uma dada cópia c, essa cópia é substituida no *quorum* por dois caminhos, cada qual começando num nó filho do nó correspondente à cópia inacessível (o algoritmo é recursivo). Um dos problemas com este algoritmo é que cada cópia, desde que esteja acessível, participa em todos os *quorums* que passam pelo seu ramo, pelo que pode ficar sobrecarregada (em particular, a cópia associada à raiz da árvore participa em todos os *quorums*). Outro exemplo é o *quorum em grelha* (Cheung *et al.*, 1990), que organiza logicamente as cópias numa grelha rectangular: um *quorum* para leitura deve conter uma cópia de cada coluna e um *quorum* para escrita deve conter, para além de um *quorum* de leitura, todos os elementos de uma coluna da grelha. Em muitos destes algoritmos a disponibilidade dos dados para escrita tende para zero à medida que o número de cópias aumenta (por este motivo, dizem-se com disponibilidade assimptótica para escrita). No entanto, existem variantes que não possuem esta desvantagem (Kumar & Cheung, 1991).

Existem representações alternativas de conjuntos de *quorums*, sendo talvez a mais conhecida a representação designada por *votação multidimensional* (Ahamad & Ammar, 1991), mas uma descrição pormenorizada de todas estas técnicas cai fora do âmbito da tese.

2.2.2.4 Algoritmos dinâmicos pessimistas

Nos algoritmos descritos anteriormente, os conjuntos de *quorum* são escolhidos durante a iniciação do sistema. Em tempo de execução, cada processo progride assim que consegue obter resposta do *quorum* associado à operação e bloqueia caso esse *quorum* não seja atingido. Um algoritmo dinâmico tenta garantir o progresso recolhendo informação acerca do estado do sistema e reconfigurando-se em tempo de execução. Tipicamente, os sistemas dinâmicos tentam trabalhar com pequenos *quorums* para leitura, uma vez que na maioria dos sistemas esta é a operação dominante. Para garantir a intersepção, são requeridos elevados *quorums* para escrita que podem não ser atingidos quando algumas cópias ficam inacessíveis. Neste caso, o sistema muda os conjuntos de *quorum*, escolhendo *quorums* que favoreçam o progresso do sistema. A grande maioria dos algoritmos referidos na secção anterior possui variantes dinâmicas.

Por exemplo, o algoritmo de votação ponderada dinâmica (Davcev, 1989) altera o critério de maioria quando suspeita da existência de uma partição. Sempre que este critério é alterado, memoriza-se a versão que cada uma das cópias possuia nesse momento (esta informação designa-se *vector de partição*). Isto permite saber quais são as cópias que participaram na maioria mais recente, só se permitindo que o critério seja alterado quando se consegue aceder a uma maioria das cópias actualizadas. Quando se detecta que a comunicação é reestabelecida, os vectores de partição são utilizados para saber quais as cópias a actualizar. Outras variantes dinâmicas de algoritmos de votação baseiam-se no mesmo princípio: identificam-se de algum modo as partições e assegura-se que uma dada operação é efectuada em cópias que se encontram na mesma partição. Os mecanismos transaccionais usados no acesso às cópias permitem garantir que as partições são ordenadas em relações às restantes operações. Geralmente, para não degradar o desempenho das operações de leitura, uma nova partição só é instalada durante operações de escrita (Agrawal & Abbadi, 1990).

Tal como os algoritmos de votação, os algoritmos baseados em representações estruturais também possuem variantes dinâmicas. Por exemplo, o algoritmo descrito por Paris e Sloope (1992) é baseado no algoritmo em grelha mas permite que a grelha se reorganize em função de informação de acessibilidade das cópias.

Primário-secundário Uma classe particular de algoritmos dinâmicos, conhecida por algoritmos *primário-secundário* (Alsberg & Day, 1976; Stonebraker, 1979), caracteriza-se por seleccionar uma cópia como responsável por estabelecer a sincronização das operações. Todas as operações incompatíveis são realizadas sobre essa cópia, designada por *primário*, que por sua vez é responsável por propagar o seus efeitos para as restantes cópias, designadas por *secundários*. Em tempo de execução, as cópias podem trocar de modo para permitir o progresso do sistema na presença de falhas (geralmente, quando o primário falha, um dos secundários é promovido a novo primário) ou simplesmente para aumentar o desempenho do sistema (movendo o primário para junto dos clientes que estão a aceder aos dados com maior frequência). Neste último caso, o modo primário pode estar associado à posse de um *testemunho*, que pode ser circulado entre as cópias (Minoura & Wiederhold, 1982).

A principal vantagem desta classe de algoritmos é a de simplificar o controlo da concorrência (uma vez que todas as operações incompatíveis são realizadas numa única cópia). A segunda vantagem é que os clientes só necessitam de contactar com uma única cópia (de cada vez), o que simplifica o processamento a efectuar nos mesmos. Este último aspecto não é desprezável, uma vez que é geralmente muito mais fácil adaptar aplicações existentes de modo a introduzir replicação usando esta técnica do que usando técnicas que requeiram a interacção com diversas cópias em cada acesso. Talvez por este motivo, esta técnica tem sido bastante utilizada, nomeadamente para construir sistemas de ficheiros tolerantes a faltas (por exemplo, HARP (Liskov *et al.*, 1990) ou Echo (Hisgen *et al.*, 1990)). A principal dificuldade associada a este método reside em garantir que, num determinado instante, apenas uma cópia assume o modo primário. Uma vez mais, identifica-se a necessidade de manter informação actualizada acerca de quais as cópias acessíveis ou, por outras palavras, identificam-se requisitos de *manutenção da filiação*.

2.2.2.5 Cópias disponíveis (algoritmos optimistas)

Existem sistemas em que a ocorrência de partições é extremamente improvável, pelo que é relativamente seguro assumir que todas as cópias inacessíveis se encontram falhadas. Face a esta hipótese, francamente optimista (uma vez que não garante a consistência das cópias caso as partições ocorram), é possível permitir o progresso desde que exista uma cópia disponível. Esta classe de algoritmos é designada por algoritmos de *cópias disponíveis* (Bernstein & Goodman, 1984; Pu *et al.*, 1988). Geralmente, os algoritmos realizam as operações de escrita em todas as cópias acessíveis e as operações de leitura na cópia mais próxima do cliente. Quando uma cópia falha e depois recupera, deve actualizar o seu estado a partir de uma cópia activa. Nestes algoritmos, a situação mais complexa ocorre quando todas as cópias falham (por exemplo, devido a uma falha de corrente eléctrica). Neste caso, é necessário garantir que o sistema recupera o seu estado actualizado, ou seja, que só se progride após a recuperação da última cópia a falhar (Skeen, 1985).

2.2.2.6 Algoritmos para mestre-cópias

Como se acabou de descrever, garantir modelos de coerência forte é uma tarefa dispendiosa que não maximiza a disponibilidade da informação. Por exemplo, clientes numa partição minoritária não podem aceder a informação actualizada como requerido pelo modelo. Por outro lado, certas aplicações podem funcionar com base em informação (ligeiramente) desactualizada (Barbara & Garcia-Molina, 1990). Isto motiva o uso de modelos de coerência fraca, como o de mestre-cópias, os quais podem ser concretizados com algoritmos menos exigentes.

Os algoritmos que concretizam o modelo mestre-cópias podem ser vistos como variantes degeneradas de algoritmos primário-secundário. Tal como nestes últimos, todas as actualizações são realizadas na cópia mestre, só que os resultados não são propagados de imediato para as cópias. Em geral, a sincronização entre o mestre e as restantes cópias é limitada ao mínimo para satisfazer o modelo de coerência. Esta sincronização pode tomar uma de duas formas: o mestre pode ficar responsável por actualizar as cópias sempre que o critério estiver em risco de ser violado; ou as cópias podem ficar responsáveis por contactar o mestre periodicamente. Por vezes, em vez de se actualizar uma dada cópia, pode-se optar por invalidá-la.

2.2.2.7 Múltiplo-mestre

Um algoritmo do tipo múltiplo-mestre permite actualizações em cópias distintas mesmo quando não é possível verificar a sua incompatibilidade. Deste modo, oferece-se uma maior disponibilidade da informação correndo o risco de obter cópias irreconciliáveis. Tal como em todos os algoritmos dinâmicos, um factor chave na aplicação destes algoritmos é a manutenção de informação actualizada acerca da acessibilidade entre réplicas. Sempre que a comunicação entre duas réplicas é reestabelecida, após um período de inacessibilidade, o algoritmo deve tentar reconciliar as réplicas. Como se referiu anteriormente, não existem soluções genéricas para este problema e os algoritmos existentes dependem da semântica das aplicações. Quando operações incompatíveis são detectadas é necessário recorrer à reconciliação manual (isto é, realizada pelo uti-

lizador final) ou cancelar algumas das operações e iniciar acções que compensem os efeitos desse cancelamento.

2.2.3 Disseminação de informação

Todos os algoritmos atrás referidos requerem a disseminação de informação, quer entre os clientes e as cópias, quer entre as diversas cópias. Faz-se aqui uma breve referência às principais alternativas existentes para realizar esta disseminação.

Em primeiro lugar, é possível classificar as alternativas no que diz respeito ao momento em que é feita a disseminação: assim que os dados são actualizados (ou propagação *pronta*) ou em períodos de baixa carga na rede ou nos processadores (ou propagação *diferida*).

Em segundo lugar, é necessário decidir qual o melhor caminho a percorrer pelos dados para assegurar a sua disseminação. Apesar deste problema poder ser resolvido ao nível da aplicação, usando conhecimento semântico acerca desta (Alonso *et al.*, 1990), a alternativa mais modular consiste em delegar esta tarefa num protocolo de encaminhamento, sobretudo se este suportar a *difusão* de dados.

Finalmente, é necessário decidir qual a informação a disseminar. Existem duas alternativas principais: disseminar o estado completo da cópia ou disseminar simplesmente as últimas alterações (também designado por *refrescamento diferencial*). A primeira alternativa é naturalmente mais simples de concretizar, embora só seja eficaz se o tamanho da cópia for pequeno.

2.3 Filiação e comunicação em grupo

Na secção anterior mostrou-se que na gestão de dados replicados existe frequentemente a necessidade de saber quais os participantes que estão mutuamente acessíveis. Existe também a necessidade de difundir informação por um conjunto de participantes. Os paradigmas, protocolos e sistemas orientados para os grupos pretendem fornecer uma solução integrada para este tipo de problemas.

De um modo genérico, designa-se por um *serviço de filiação em grupo* um serviço que permite aos participantes organizarem-se em grupos e receber informação actualizada de quais os elementos acessíveis (esta informação designa-se por *vista* de grupo). Do mesmo modo, um serviço de *comunicação em grupo* permite aos elementos do grupo trocarem informação, sendo-lhes oferecidas garantias de entrega e de ordenação que variam de serviço para serviço.

A utilidade destes serviços não se limita naturalmente a aplicações de suporte à replicação, sendo extensiva a todas as áreas que utilizam explicita ou implicitamente a noção de grupos de participantes. Em particular, aplicações na área do trabalho cooperativo, as quais utilizam explicitamente a noção de grupo, podem também beneficiar com a utilização de serviços orientados aos grupos (Cosquer & Veríssimo, 1994).

Tal como no caso dos protocolos de gestão da replicação anteriormente referidos, existem diversos serviços orientados aos grupos que se distinguem, por vezes subtilmente, por oferecerem diferentes propriedades. Por outro lado, a concretização de um dado serviço é também extremamente dependente das propriedades do sistema, nomeadamente do tipo de faltas que se pretende tolerar. Nesta secção apresentam-se as principais serviços oferecidos nesta área, assim como os principais algoritmos para os concretizar.

2.3.1 Filiação em grupo

A função de um serviço de filiação em grupo é fornecer aos participantes *vistas de grupo*, isto é, listas de identificadores de processos que se encontram mutuamente acessíveis (pressupõe-se que os processos são univocamente identificados e que pode ser estabelecida uma relação de ordem total entre os identificadores). A maioria dos serviços de filiação em grupo oferecem primitivas que permitem a um processo inscrever-se (ou cancelar a inscrição), entregando uma nova vista a todos os membros sempre que ocorrem alterações à filiação. Do mesmo modo, estes serviços são geralmente responsáveis por chegar a acordo acerca da ocorrência de falhas, sejam estas falhas nos nós ou falhas no meio de comunicação. Os serviços de filiação distinguem-se sobretudo por oferecerem diferentes garantias de ordenação na entrega das vistas, quer

entre si, quer em relação ao fluxo de mensagens.

Existem duas propriedades fundamentais que um serviço de filiação deve tentar satisfazer (Hiltunen & Schlichting, 1994b):

- *Exactidão:* A informação entregue reflecte a actual situação física.
- *Coerência*: A informação é idêntica em todos os processos na mesma vista.

2.3.1.1 Obtenção da exactidão

A obtenção destas propriedades está limitada pelas caraterísticas do sistema. Neste momento, convém distinguir duas grandes classes de sistemas, a saber:

- *sistemas síncronos:* caracterizados por atrasos limitados no processamento, na troca das mensagens e na velocidade dos relógios.
- *sistemas assíncronos:* em que não é possível definir limites aos atrasos na troca de mensagens ou no processamento (mais precisamente, sempre que estes limites, mesmo quando existem, são demasiado elevados para poderem ser utilizados na prática).

Um sistema síncrono possui duas características fundamentais. Em primeiro lugar, torna possível oferecer uma referência de tempo global, geralmente obtida através de um *protocolo de sincronização de relógios* (Schneider, 1987). Em segundo lugar, torna possível detectar de um modo fiável, usando a passagem do tempo, uma falha de um nó (ou uma partição na rede, quando estas são possíveis). Por outro lado, num sistema assíncrono é geralmente impossível obter uma referência de tempo global, a não ser que se usem canais dedicados (por exemplo, veja-se Veríssimo *et al.* (1995)). Mais importante ainda, num sistema assíncrono não é possível distinguir a falha de um processo (ou de um elo) de um processo (ou um elo) lento. Por este motivo, num sistema assíncrono o objectivo de *exactidão* só pode ser aproximado.

Estas limitações dos sistemas assíncronos são bem conhecidas, e formalizadas em diversos resultados fundamentais, tais como o da não existência de soluções

determinísticas para o problema do consenso[3] (*consensus*) distribuído em sistemas assíncronos sujeitos a faltas (Fischer *et al.*, 1985). No entanto, estas limitações não impedem o desenvolvimento de serviços distribuídos seguindo modelos de coerência mais fracos. Um destes modelos é o modelo da *sincronia virtual* (Birman & Joseph, 1987) de que se voltará a falar adiante.

Num sistema assíncrono, os resultados de impossibilidade podem ser evitados recorrendo a um componente auxiliar designado por *detector de falhas*. Este é encarregue de indicar quais os processos falhados no sistema. O modo mais simples de concretizar este componente é usar temporizadores (*timers*), solução que, num sistema assíncrono, pode gerar falsas detecções (isto é, indicar a falha de processadores correctos). Este problema pode ser minimizado através de soluções de engenharia e, sobretudo, de uma afinação cuidada dos seus parâmetros em função das características da rede (Frazão, 1995).

Recentemente, tem vindo a ser realizado bastante trabalho na formalização das propriedades dos detectores de falhas e na classificação do seu potencial para a satisfação para diversos tipos de requisitos. Um dos trabalhos mais influentes nesta área consiste na classificação apresentada por Chandra e Toueg (1991). Segundo estes autores, os detectores de falhas possuem duas propriedades elementares:

- a plenitude (*completeness*), um requisito de animação (*liveness*), que especifica que uma falha deve eventualmente ser detectada por pelo menos um processo correcto (*plenitude fraca*) ou por todos os processos correctos (*plenitude forte*);
- e a exactidão (*accuracy*), um requisito de segurança (*safety*), que indica que nenhum processo correcto é considerado suspeito (*exactidão forte*) ou que pelo menos um processo correcto nunca é considerado suspeito (*exactidão fraca*).

[3] Tal como formulado por Fischer *et al.* (1985), o problema do consenso consiste em fazer com que todos os processos correctos (de um conjunto inicial de processos, cada qual detendo um valor prévio para propor) cheguem a acordo acerca de um único valor que depende dos valores previamente detidos.

Usando estas propriedades podem definir-se três classes de detectores de falhas, a saber[4]:

- detectores de falhas *perfeitos*, denotados pelo símbolo $\mathcal{P}$, satisfazendo a plenitude forte e exactidão forte.
- detectores de falhas *fortes*, denotados pelo símbolo $\mathcal{S}$, satisfazendo a plenitude forte e exactidão fraca.
- detectores de falhas *fracos*, denotados pelo símbolo $\mathcal{W}$, satisfazendo a plenitude fraca e exactidão fraca.

A propriedade de exactidão pode ser enfraquecida, exigindo-se apenas que seja satisfeita alguma vez (*eventually*), o que dá origem a um novo conjunto de detectores de falhas, designados por *alguma vez perfeitos*, *alguma vez fortes* e *alguma vez fracos*, e denotados respectivamente por $\diamond\mathcal{P}$, $\diamond\mathcal{S}$, $\diamond\mathcal{W}$. Chandra e Toueg (1991) demonstram como se pode converter um detector do tipo $\diamond\mathcal{W}$ num detector do tipo $\diamond\mathcal{S}$ e que com estes detectores é possível resolver o consenso distribuído desde que só falhe uma minoria dos nós.

Infelizmente, mesmo o mais fraco destes detectores não pode ser obtido num sistema assíncrono, podendo apenas ser aproximado. Ricciardi *et al.* (1993) mostram que o comportamento de um detector $\diamond\mathcal{W}$ pressupõe que o sistema nunca é sujeito a partições. Existe pois um compromisso entre a *exactidão* e a *animação* de um detector de falhas (Hiltunen & Schlichting, 1994b):

- Por exemplo, um processo pode avisar os restantes processos que falhou após a sua recuperação. Neste caso, a informação é exacta mas pode ser indefinidamente atrasada.
- Por outro lado, a indicação de uma falha pode ser gerada com base em testes não fiáveis, tais como temporizadores, os quais permitem o progresso mas podem gerar falsas suspeitas.

[4] A notação deriva das palavras do Inglês, "Perfect", "Strong" e "Weak", mas está de tal modo divulgada que não se considerou justificável a sua "tradução".

De um modo geral, compromete-se a exactidão na detecção de falhas mas deixa-se o ónus da detecção de recuperações aos processos falhados.

2.3.1.2 Obtenção da coerência

Do ponto de vista da coerência, o serviço de filiação distingue-se pela ordem relativa da entrega de vistas (a entrega de uma mensagem contendo uma vista designa-se geralmente por *instalação da vista*). Geralmente, deseja-se que os processos recebam a mesma informação pela mesma ordem. Por exemplo, considere-se um grupo $\mathcal{G}$, e duas vistas de grupo $V^1(\mathcal{G})$ e $V^2(\mathcal{G})$. Se um processo $p \in V^1(\mathcal{G}) \cap V^2(\mathcal{G})$ instala V^1 antes de V^2, então todos os processos $q \in V^1(\mathcal{G}) \cap V^2(\mathcal{G})$ que permaneçam acessíveis devem também instalar V^1 antes de V^2.

O serviço mais simples ordena as vistas de um modo parcial, permitindo que vistas concorrentes se interseptem (designando-se por *serviço de filiação parcial fraca* (Jahanian *et al.*, 1993)). Este serviço pode ser concretizado por protocolos pouco dispendiosos que tentam garantir a convergência das vistas mas não impedem os processos de receber informação incoerente. É também possível, embora mais complexo, garantir que vistas concorrentes não se interseptam (este serviço designa-se por *serviço de filiação parcial forte* (Jahanian *et al.*, 1993)). Os serviços de filiação do tipo parcial possuem a vantagem de, em caso de falhas na rede, garantirem o progresso em todas as partições. Em contrapartida, a sua utilização pressupõe a capacidade da aplicação executar procedimentos de reconciliação quando, após a recuperação da rede, duas ou mais partições se voltam a fundir. Como se referiu anteriormente, a reconciliação nem sempre é possível, pelo que é frequente fornecer um serviço de filiação que ordena as vistas de um modo total e que, na presença de partições, preserva a animação numa única partição do sistema (este serviço designa-se por *serviço de filiação linear* (Ricciardi & Birman, 1991; Schiper & Sandoz, 1994)).

Os serviços de filiação não estão dissociados dos serviços de comunicação. Geralmente, o valor acrescentado destes serviços consiste na ordenação das vistas em relação ao fluxo de mensagens. A maioria dos serviços hoje existentes baseia-se no conceito de *sincronia virtual*, originalmente proposto no sistema ISIS (Birman & Joseph, 1987). Uma

possível definição de sincronia virtual é a seguinte (Schiper & Ricciardi, 1993):

Difusão virtualmente síncrona: Considere-se um grupo $\mathcal{G}$, uma vista $V^i(\mathcal{G})$, e uma mensagem m difundida para $\mathcal{G}$. A difusão m é virtualmente síncrona na vista $V^i(\mathcal{G})$ se e só se: para todo o p pertencente a $V^i(\mathcal{G})$ tal que p tenha entregue m na vista $V^i(\mathcal{G})$ e tenha instalado a vista $V^{i+1}(\mathcal{G})$, então todos os processos $q \in V^i(\mathcal{G})$ que tenham instalado $V^{i+1}(\mathcal{G})$, também entregaram m antes de instalar $V^{i+1}(\mathcal{G})$.

Diversas variantes da sincronia virtual podem ser obtidas, não só em função do modo como as vistas são ordenadas entre si, mas também em função do modo como as vistas são ordenadas em relação ao fluxo das mensagens. Por exemplo, Babaoğlu *et al.* (1995) descrevem um serviço parcial *quasi-forte*, o qual permite a intersepção de vistas concorrentes desde que estas sejam subconjuntos das restantes. Friedman e Renesse (1995) distinguem a *sincronia virtual forte* (em que as mensagens são entregues na vista em que foram enviadas) da *sincronia virtual fraca* (em que estas podem ser entregues numa outra vista)[5].

2.3.2 Comunicação em grupo

Designam-se por serviços de comunicação em grupo, serviços que permitem difundir mensagens para todos (ou para um subconjunto) de filiados num grupo de comunicação. Do ponto de vista do serviço fornecido, importa distinguir dois aspectos fundamentais:

- *Aspectos de fiabilidade* respeitantes às garantias oferecidas na entrega das mensagens.

- *Aspectos de ordenação* respeitantes às garantias oferecidas na ordenação das mensagens entre si.

[5]O leitor deve notar que os termos *fraco* e *forte* não são usados aqui para caracterizar a composição relativa de vistas concorrentes, mas sim para associar a transmissão de mensagens com a instalação de vistas.

2.3.2.1 Fiabilidade na difusão

Entende-se por fiabilidade da difusão em grupo a garantia de que uma mensagem é entregue a todos os destinatários acessíveis, geralmente todos os filiados no grupo. Uma vez que a filiação do grupo pode variar no tempo, a fiabilidade da comunicação tem necessariamente que ser definida em relação a uma vista de grupo, e não pode ser dissociada do serviço de filiação. Uma possível definição de difusão fiável é pois a definição de sincronia virtual apresentada anteriormente.

Esta definição não oferece nenhumas garantias no que diz respeito a mensagens entregues a processos que abandonem uma vista. Por exemplo, se um processo receber uma mensagem e abandonar a vista imediatamente a seguir, não chegando a instalar a vista seguinte, não se tem a garantia que os restantes processos que se mantenham activos (e venham a instalar uma nova vista) venham a receber essa mensagem. Isto pode levar a que um processo que fique isolado (por exemplo devido a uma partição) realize acções incompatíveis com as executadas na partição seleccionada. É possível reforçar as garantias oferecidas pela sincronia virtual, oferecendo uma primitiva de comunicação uniforme (Schiper & Sandoz, 1993) (também designada por difusão *segura* (Amir *et al.*, 1993)):

Difusão uniforme virtualmente síncrona: Considere-se um grupo $\mathcal{G}$, uma vista $V^i(\mathcal{G})$, e uma mensagem m difundida para $\mathcal{G}$. A difusão m é virtualmente síncrona na vista $V^i(\mathcal{G})$ se e só se: para todo o p pertencente a $V^i(\mathcal{G})$ tal que p tenha entregue m na vista $V^i(\mathcal{G})$ (mesmo que não tenha instalado a vista $V^{i+1}(\mathcal{G})$), então todos os processos $q \in V^i(\mathcal{G})$ que tenham instalado $V^{i+1}(\mathcal{G})$, também entregaram m antes de instalar $V^{i+1}(\mathcal{G})$.

Note-se que esta definição não obriga a que mensagens sejam entregues a processos falhados (algo claramente impossível) mas obriga a que mensagens entregues a processos falhados (ou isolados) sejam necessariamente entregues a todos os processos correctos. Esta primitiva é particularmente útil para a concretização de protocolos de confirmação atómica (Schiper & Sandoz, 1993). Infelizmente, a concretização deste serviço requer pelo menos duas fases de troca de informação antes de uma mensagem poder ser entregue (numa primeira fase obtém-se a garantia que a mensagem poderá

ser entregue a todos os participantes correctos, e só após essa garantia ter sido obtida se entrega a mensagem).

2.3.2.2 Ordenação de mensagens

Os protocolos de filiação em grupo oferecem já garantias de ordenação das mensagens em relação às indicações de alteração da filiação. Todavia, demonstra-se que a construção de algumas aplicações distribuídas pode ser simplificada caso se imponham restrições adicionais em relação à ordem de entrega das mensagens (Birman & Joseph, 1989; Schneider, 1990; Renesse, 1993). Por exemplo, na secção 2.2.1, referiu-se que para oferecer um modelo de coerência de memória forte é necessário estabelecer uma ordem total entre as operações de actualização dos dados.

Por esta razão, a grande maioria dos serviços de comunicação em grupo oferecem primitivas que concretizam diferentes políticas de ordenação, sendo as mais utilizadas as seguintes:

- *Ordem FIFO*[6]: Duas mensagens enviadas pelo mesmo emissor são entregues pela ordem pela qual forma enviadas.
- *Ordem Causal:* As mensagens são entregues respeitando as relações de *causalidade potencial*, definida do seguinte modo: num sistema distribuído no qual a informação é trocada exclusivamente através da troca de mensagens, diz-se que uma mensagem m_1 *precede*, ou está *potencialmente relacionada de modo causal* com uma outra mensagem m_2, denotando-se $m_1 \rightarrow m_2$, se e apenas se:
  - m_1 e m_2 foram emitidas pelo mesmo processo e m_2 foi enviada após m_1 ou;
  - m_1 foi entregue ao emissor de m_2 antes de m_2 ter sido emitida ou;
  - existe m_3 tal que $m_1 \rightarrow m_3$ e $m_3 \rightarrow m_2$.
- *Ordem total:* Duas mensagens entregues a um par de processos são entregues pela mesma ordem a ambos.

[6] Do Inglês, "First-In-First-Out".

Um serviço que satisfaz a propriedade anterior pode ser designado por serviço de *ordenação total forte* uma vez que garante a ordem total entre mensagens entregues a qualquer par de processos. Em particular, se um processo entregar um par de mensagens por uma determinada ordem e posteriormente falhar, qualquer processo correcto que venha a entregar esse par de mensagens deve entregá-las na mesma ordem. Este serviço pode ser enfraquecido, por exemplo, oferecendo uma *ordenação total fraca*, a qual garante a ordem de entrega apenas para processos que permanecem correctos (para uma possível hierarquia, veja-se o trabalho de Wilhelm e Schiper (1995)).

2.3.3 Algoritmos

Existem diversas técnicas para concretizar os serviços de filiação e comunicação em grupo mencionados. Como se referiu no início deste capítulo, optou-se por não apresentar aqui uma descrição extensiva de todas as técnicas descritas na vasta bibliografia disponível sobre o assunto. Nos parágrafos que se seguem, de modo a dar ao leitor uma ideia da complexidade do problema, referem-se apenas — e de modo conciso — alguns dos algoritmos que se julgam mais representativos. Dada a sua especificidade, panorâmicas sobre o trabalho directamente relacionado com as soluções apresentadas na tese são apresentadas no Capítulo 5.

2.3.3.1 Factores de influência

Tal como no caso dos algoritmos para gestão de dados replicados, as técnicas para concretizar serviços de filiação e comunicação em grupo dependem das características do sistema alvo e dos critérios de desempenho que se pretendem satisfazer. Os factores enunciados para esses algoritmos são pois também relevantes neste contexto.

No que se refere a factores operacionais, referem-se os seguintes:

- *Tipos de faltas.* Têm sido desenvolvidos algoritmos de comunicação e filiação em grupo para tolerar os mais diversos tipos de faltas. Naturalmente, a complexidade

do algoritmo aumenta com o número e com a malignidade das faltas toleradas. No extremo da complexidade, encontram-se os algoritmos que toleram faltas arbitrárias. Infelizmente, demonstra-se que este tipo de faltas só pode ser tolerado em ambientes restritos (nomeadamente, com um comportamento síncrono) e com um custo proibitivo para a grande maioria das aplicações (Lamport *et al.*, 1982). Nesta dissertação, estudam-se algoritmos que toleram os seguintes tipos de falhas: falhas por paragem nos processos, omissões no meio e partições na rede.

- *Características do meio de comunicação.* Existem diversas propriedades dos meios de comunicação que devem ser tidas em conta no desenvolvimento de protocolos orientados para grupos. Salientam-se as seguintes: *latência*, tanto mais importante quanto maior for o número de passos do tipo pedido-resposta necessários para executar o serviço; *débito*, do qual depende a quantidade relativa de informação de controlo que se pode usar e que condiciona a utilização de políticas de agregação de mensagens; *capacidade de difusão ao nível físico*, a qual deve ser aproveitada sempre que possível; e *topologia*, que pode ser usada para optimizar o desempenho dos algoritmos.

- *Padrões de tráfego.* O padrão de acesso aos serviços, nomeadamente a distribuição do tráfego submetido para transmissão (quer a distribuição dos pedidos no tempo, quer a distribuição do tamanho das mensagens) são factores dos quais depende o desempenho óptimo do sistema. Infelizmente, estes factores variam de aplicação para aplicação. Por outro lado, o esforço necessário para desenvolver protocolos orientados aos grupos aconselha a que as concretizações sejam úteis para um largo leque de aplicações. Por este motivo, é importante que as concretizações sejam configuráveis, se possível utilizando algoritmos com a capacidade de se adaptarem em tempo-de-execução a variações destes factores.

Listam-se também os critérios de desempenho mais relevantes nesta área:

- *Complexidade.* Tal como qualquer programa, uma concretização de um algoritmo de filiação ou difusão em grupo necessita de ser testada, optimizada, adaptada a vários ambientes de execução e, com o tempo, ajustada para satisfazer novos

requisitos ou novas condições operacionais. O recurso a algoritmos pouco complexos não só simplifica as tarefas atrás mencionadas como facilita o processo de avaliação da sua correcção.

- *Custos de armazenamento.* Os protocolos de difusão fiável exigem, de um modo geral, a manutenção de cópias das mensagens trocadas no sistema até que a entrega destas esteja garantida para todos os destinatários. Estes custos são comuns aos protocolos ponto-a-ponto, embora amplificados pelo elevado número de destinatários, e são controlados por técnicas de controlo de fluxo. Protocolos de ordenação, requerem também a troca de informação de controlo cujo peso pode não ser desprezável. Este aspecto será focado com especial atenção no Capítulo 5.

- *Latência e débito.* Como se referiu anteriormente, estes dois requisitos são difíceis de conciliar. A obtenção de baixas latências requer geralmente políticas de tolerância a faltas agressivas, tais como a transmissão redundante no espaço (em múltiplos canais (Babaoğlu & Drummond, 1985) ou em sobre múltiplos percursos (Garcia-Molina *et al.*, 1991)) ou no tempo (Rufino, 1993). Por outro lado, estas políticas consomem recursos necessários para obter elevados débitos. O aspecto a privilegiar depende do tipo de aplicações que se pretende suportar.

- *Confiança no funcionamento.* Medida como a capacidade do algoritmo oferecer o serviço especificado apesar da ocorrência de faltas. Geralmente medida em termos do número máximo de faltas de cada tipo que o algoritmo tolera (embora também possa ser especificada de modo probabilístico (Babaoğlu, 1987)).

2.3.3.2 Algoritmos de difusão

Podem-se identificar diversos componentes dos algoritmos de difusão de mensagens, nomeadamente:

- *encaminhamento*, responsável por escolher o percurso adequado para fazer chegar a mensagem aos seus destinatários;

- *recuperação de omissões da rede*, responsável por retransmitir mensagens que sejam descartadas ou adulteradas pela infra-estrutura física;
- *controlo de fluxo*, responsável por minimizar a perda de informação causada pela falta de memória nos destinatários no momento da recepção;
- *recuperação da falha do remetente e ordenação em relação às vistas de grupo*, responsável por assegurar um determinado critério de fiabilidade;
- *ordenação das mensagens entre si*, de acordo com um determinado requisito de ordenação.

Os três primeiros aspectos são da responsabilidade de um *serviço de transporte em difusão*. Garantias de entrega a todos os destinatários em caso de falha do remetente, e garantias de ordenação em relação a vistas de grupo são geralmente oferecidas por algoritmos de filiação. Garantias de ordenação relativa de mensagens são oferecidas por algoritmos de ordenação. Os dois últimos aspectos serão referidos nas secções subsequentes.

O processo de encaminhamento consiste em escolher um percurso que tente minimizar o número de mensagens trocadas e ao mesmo tempo, tente minimizar a latência da difusão. Para satisfazer o primeiro requisito, deve-se tentar aproveitar as capacidades de difusão física sempre que a topologia da rede assim o permita. Para satisfazer o segundo requisito, deve-se tentar difundir a mensagem de acordo com uma árvore de menor custo. Apesar de existirem protocolos de difusão fiável que tentam abordar este aspecto directamente (Schneider *et al.*, 1984; Garcia-Molina & Spauster, 1991), a tendência actual é para o encaminhamento em difusão ser da responsabilidade dos protocolos normalizados (Deering, 1989).

A recuperação de omissões no meio é feita à custa de troca de confirmações de recepção e posterior retransmissão das mensagens perdidas. Estas confirmações podem ser geradas sempre que uma mensagem é recebida (*confirmação positiva*) ou apenas quando a ausência de uma mensagem é detectada (*confirmação negativa*). O primeiro método privilegia a rapidez na detecção da omissão (mesmo com tráfego esporádico) e o segundo minimiza o número de confirmações geradas.

Finalmente, existem diversas técnicas para concretizar o controlo de fluxo (Macedo *et al.*, 1995), incluindo a utilização de créditos (Powell, 1991), janelas-deslizantes (Tanenbaum, 1989), controlo da taxa de transmissão (Sanders, 1990), etc.

2.3.3.3 Algoritmos de filiação

Os algoritmos de filiação possuem pelo menos três componentes macroscópicos: um detector de falhas, um componente responsável por gerar vistas, e um componente responsável por garantir a entrega das mensagens (para uma possível decomposição mais fina veja-se, por exemplo, Hiltunen e Schlichting (1994a)). Estes componentes não são completamente independentes, devendo interagir de modo a fornecer o serviço desejado.

Como se referiu anteriormente, o método mais simples para efectuar a detecção de falhas consiste em exigir que os processos enviem periodicamente mensagens para a rede (quer voluntariamente, quer por resposta a uma mensagem de teste), e assumir a sua falha quando este período é excedido (através de uma medição local do tempo, geralmente recorrendo a um temporizador). A exactidão deste mecanismo pode ser melhorada, fazendo com que os processos troquem entre si os valores obtidos localmente e tomem uma decisão em função dos valores recolhidos. É também possível utilizar critérios de decisão mais próximos dos requisitos específicos das aplicações (por exemplo, considerando falhado um processo que se mostre incapaz de manter de modo sustentado um débito pré-determinado (Cosquer *et al.*, 1995b)). O componente responsável por gerar vistas executa um algoritmo de acordo, o qual pode ser totalmente distribuído ou, alternativamente, basear-se num mecanismo de eleição que selecciona um coordenador responsável por instalar, numa segunda fase, a próxima vista. O componente responsável por entregar mensagens é executado antes da instalação de uma nova vista, e garante que todas as mensagens referentes à vista anterior são entregues a todos os destinatários correctos.

A título de exemplo considere-se o algoritmo descrito por Dolev *et al.* (1993a)[7], cu-

[7] A descrição é extremamente concisa e, necessariamente, pouco precisa. Julga-se, no entanto, que

jos princípios básicos foram incorporados em diversos sistemas, incluindo o sistema Transis (Amir *et al.*, 1992), o Totem (Amir *et al.*, 1993) e o sistema Horus (Renesse *et al.*, 1992). Este algoritmo assume a existência de um serviço de transporte capaz de ordenar as mensagens de modo causal. O algoritmo usa, entre outras, as seguintes estruturas auxiliares (mantidas em cada processo): o conjunto de processos membros do grupo de acordo com a última vista instalada (PM); um contador (N), designado por *contexto*, que é incrementado sempre que uma nova vista é instalada; o conjunto de processos correctos que poderão fazer parte da próxima vista (PC); o conjunto de processos falhados (PF); um conjunto de mensagens em espera (ME), correspondendo às mensagens que podem ser descartadas em caso de falha do remetente; e um conjunto de mensagens pendentes (MP), que devem necessariamente ser entregues. O protocolo distingue também as mensagens de dados das mensagens de filiação; estas últimas disseminam o conteúdo das variáveis N, PC e PF do seu remetente.

O funcionamento do algoritmo é resumido nas próximas linhas. O conjunto PC é actualizado quando é recebida uma mensagem de um processo que não se encontra nem em PC nem em PF. O conjunto PF é actualizado sempre que se detecta localmente a falha de outro processo. Estes conjuntos são também actualizados sempre que se recebe uma mensagem de filiação, fazendo a sua reunião com os conjuntos incluídos na mensagem ou, alternativamente, assumindo a falha do remetente caso este considere o processo local falhado (para garantir a simetria da detecção de falhas). Sempre que, por uma das razões atrás enunciadas, um processo altera PC ou PF, é disseminada uma nova mensagem de filiação. Quando se recebe um conjunto de mensagens de filiação equivalentes (isto é, com os mesmos valores de N, PC e PF) de todos os processos em PC $\setminus$ PF, instala-se uma nova vista. Antes porém, entregam-se todas as mensagens que dependem de modo causal das mensagens de filiação que deram origem a essa vista. Para além disso, mensagens temporariamente armazenadas em ME e emitidas por processos que se tenham filiado são agora movidas para MP. Finalmente, mensagens provenientes dos processos falhados que não tenham sido entregues a nenhum processo na nova vista são descartadas. Ao instalar-se de uma nova vista atribuem-se novos valores a PM, N, PC e PF.

ilustra os aspectos fundamentais deste tipo de algoritmos.

2.3.3.4 Algoritmos de ordenação

Os algoritmos de ordenação dependem bastante do tipo de ordem que se deseja obter. Apresentam-se as principais alternativas para a ordenação FIFO, causal e total.

Ordem FIFO A ordem FIFO pode ser obtida marcando as mensagem com um número de sequência no momento da sua transmissão e entregando-as de acordo com essa numeração.

Ordem causal De modo a preservar a ordem causal, cada mensagem deve transportar informação que permita ao receptor identificar quais as mensagens que a precedem. Esta informação pode ser de natureza diversa: estampilhas com o tempo real em que a mensagem foi emitida (em sistemas síncronos) (Veríssimo, 1995); estampilhas lógicas (sob diversas formas, nomeadamente, relógios lógicos (Lamport, 1978), relógios vectoriais (Peterson *et al.*, 1989; Ladin *et al.*, 1990), e relógios matriciais (Raynal *et al.*, 1991)); identificação explícita das mensagens precedentes (Rodrigues & Veríssimo, 1995b); ou mesmo as próprias mensagens precedentes (Birman & Joseph, 1987). No Capítulo 5 apresenta-se uma panorâmica mais pormenorizada das alternativas disponíveis.

Ordem total Em sistemas síncronos, o método mais simples de ordenar as mensagens de um modo total consiste em estampilhar a mensagem com o tempo real em que a mensagem foi emitida e em entregar as mensagens por ordem do instante de transmissão (sendo mensagens enviadas no mesmo instante ordenadas de acordo com um critério pré definido, por exemplo, usando a ordem dos identificadores dos processos como critério de seriação). Esta classe de algoritmos (Cristian *et al.*, 1985) pode implicar um atraso sistemático na entrega das mensagens, na ordem do pior tempo de propagação na rede (uma vez que existem execuções em que, só após este intervalo, se tem a certeza de que já não chegará nenhuma mensagem concorrente com a que se pretende ordenar). Por este motivo, esta aproximação é impraticável em redes com elevada latência.

Uma segunda alternativa consiste em negociar, entre todos os destinatários, um

número de ordem para cada mensagem que é recebida. De modo a diminuir o número de mensagens em difusão, esta negociação é geralmente coordenada pelo próprio emissor. De acordo com esta aproximação (Birman & Joseph, 1987; Schneider, 1990), cada destinatário propõe um número de ordem com base nas mensagens por si já recebidas, o qual é utilizado para calcular o número de ordem definitivo. Esta técnica é poderosa pois permite, por exemplo, calcular o número de ordem em função da prioridade da mensagem (Rodrigues *et al.*, 1995a). Infelizmente, a necessidade de trocar informação entre todos os destinatários a cada transmissão restringe a sua aplicabilidade a sistemas com pequeno número de participantes.

Uma terceira alternativa consiste em delegar num processo a responsabilidade de atribuir uma ordenação para as mensagens. Este processo trabalha como um sequenciador e é geralmente designado por *processo com testemunho*. Foram publicados diversos algoritmos baseados neste princípio, oferecendo diferentes graus de tolerância a faltas. A família de protocolos de Chang e Maxemchuk (1984) faz circular o testemunho entre um conjunto de processos (que são simultaneamente emissores e receptores). O protocolo do sistema Amoeba (Kaashoek & Tanenbaum, 1991) é semelhante ao protocolo anterior sem os mecanismos de tolerância a faltas. O sistema ISIS (Birman *et al.*, 1991a) também usa uma aproximação baseada em testemunho mas tenta migrar o testemunho para o processo que transmite com maior taxa. O protocolo Totem (Amir *et al.*, 1993) organiza os processos num anel lógico no qual o testemunho é passado. No Totem, o testemunho tem diversas finalidades: para ordenação (as mensagens são ordenadas de acordo com a sua "inserção" no testemunho) e para controlo de fluxo. Os protocolos baseados em testemunho são atraentes pela sua simplicidade e bom desempenho em redes com baixa latência.

Finalmente, uma quarta alternativa consiste a recorrer a *relógios lógicos* (Lamport, 1978), ou *relógios vectoriais* (Peterson *et al.*, 1989; Ladin *et al.*, 1990), os quais podem também ser usadas para assegurar a entrega causal de mensagens. De acordo com esta técnica, as mensagens são entregues pela sua ordem parcial e mensagens concorrentes são totalmente ordenadas usando uma função determinística. Estes protocolos são também conhecidos por protocolos *simétricos*, uma vez que todos os processos executam o mesmo algoritmo. Nestes protocolos, uma mensagem só pode ser entregue quando

há a garantia de que não vão ser recebidas mais mensagens concorrentes ou, por outras palavras, quando uma mensagem com estampilha igual ou superior já foi recebida de *todos* os restantes processos (Schneider, 1990) (isto significa que todos os processos devem periodicamente enviar mensagens). Quando existe um elevado número de processos no sistema, este esquema torna-se difícil de gerir uma vez que pode ser necessário enviar mensagens supérfluas para evitar elevados atrasos na entrega das mensagens. O protocolo ToTo (Dolev *et al.*, 1993b) minimiza este efeito ao usar um teste de estabilidade que requer a recepção de mensagens com estampilha igual ou superior de apenas uma maioria de processos no sistema, fornecendo deste modo *entrega antecipada*, pelo menos a alguns processos. No entanto, a latência no pior caso é ainda limitada pela taxa de transmissão do processo mais lento na maioria. Este aspecto será retomado no Capítulo 5.

2.4 O sistema Delta-4

A arquitectura Delta-4 (Powell, 1991) é uma arquitectura que permite a construção de sistemas distribuídos e tolerantes-a-faltas, disponibilizando um conjunto de serviços construídos segundo uma orientação para os grupos. A arquitectura, foi concebida por um consórcio de diversas companhias, institutos e universidades europeias, ao abrigo do programa ESPRIT. O nome da arquitectura deriva da designação deste projecto, em Inglês, "**D**efinition and **D**esign of an Open **D**ependable **D**istributed Architecture".

O Delta-4 foi, tanto quanto é do conhecimento do autor, uma das primeiras arquitecturas a absorver de um modo integrado, e a todos os níveis da mesma, as tecnologias emergentes de comunicação e filiação em grupo. Sendo uma arquitectura de referência, no desenvolvimento da qual o autor esteve envolvido (e que esteve, em diversos aspectos, na génese de algum do trabalho apresentado nesta dissertação), justifica-se uma breve apresentação das suas características principais. Esta apresentação, necessariamente sumária, é baseada na óptima perspectiva oferecida por Powell (1994).

Um sistema Delta-4 consiste num número de computadores (possivelmente heterogéneos) ligados por um sistema de comunicação fiável. Os programas de aplicação

estão estruturados como componentes de *software*, distribuídos pelos vários nós do sistema. Um dado componente pode encontrar-se replicado, sendo as cópias executadas em diferentes máquinas do sistema. Cada máquina compõe-se de um *computador* hospedeiro e de um *Controlador de Ligação ao Meio* (CLM). Os CLMs são processadores de comunicações dedicados, executando o sistema de comunicação fiável, o qual permite comunicação multi-ponto entre entidades computacionais. Os CLMs constituem a única maquinaria específica da arquitectura Delta-4; enquanto o computador hospedeiro poderá não ser um componente com modo de falha controlada, o CLM deverá justificar a hipótese de falha silenciosa. A combinação do computador hospedeiro com o CLM forma uma *estação* da rede.

Os Executivos Locais[8] (LEXs) dos computadores hospedeiros podem ser heterogéneos. Cada CLM possui o seu executivo local na forma de um monitor de tempo-real. O *software* distribuído correndo em cada nó pode ser classificado do seguinte modo:

- O *software* de comunicações executado nos CLMs; o sistema de comunicações na arquitectura aberta é designado por *Sistema de Comunicação em Difusão (SCD)*, fornecendo comunicação fiável baseado em ligações multi-ponto.

- O *software* de administração que gere os elementos computacionais, nomeadamente os de comunicação do sistema e que é executado parcialmente no computador hospedeiro e parcialmente no CLM.

- O *Ambiente de Suporte às Aplicações* (DELTASE) que fornece uma base para a construção de aplicações distribuídas. O DELTASE foi concebido segundo normas emergentes na altura, tal como o modelo ODP[9] (ODP, 1987) e acompanhou de perto o trabalho de outras organizações que assistem este esforço de normalização, nomeadamente a arquitectura ANSA[10] (1987).

[8] Designação para o ambiente de execução local, geralmente constituído por um *sistema operativo* ou *núcleo de tempo-real*.

[9] Open Distributed Processing.

[10] Advanced Networked Systems Architecture.

- O *software* do utilizador, o qual consiste de componentes de *software*, potencialmente escritos em linguagens diferentes, os quais interagem através de *Pedidos de Execução Remota* (Remote Service Request (Powell, 1991)).

Particular relevância para esta tese possui o Sistema de Comunicação em Difusão, desenvolvido como uma arquitectura de camadas, semelhante à OSI[11]. Os principais componentes desta arquitectura são (Powell, 1991):

- Um componente de *Filiação e Difusão em Rede Local*. Uma versão deste componente foi desenvolvida no INESC pelo Grupo de Sistemas Distribuídos e Automatização Industrial e posteriormente aperfeiçoado pelo autor. Este componente será descrito no Capítulo 4.
- Um *Serviço de Transporte em Difusão*, suportando ligações multiponto fiáveis.
- Um componente de *Gestão da Replicação*, elemento chave na concretização das técnicas de tolerância a faltas da arquitectura. Este nível coordena a comunicação entre pontos de acesso replicados, garantindo a entrega das mensagens a todos os pontos de acesso e arbitrando os eventos de transmissão de modo a que os pontos de acesso de destino recebam apenas uma cópia de cada mensagem. Este componente pode executar protocolos de detecção e compensação de erros quando necessário.
- Serviços de *Sessão* e *Apresentação* multiponto.

Os serviços do Sistema de Comunicação em Difusão são usados para fornecer três técnicas de tolerância a faltas.

A primeira técnica envolve a execução paralela de várias cópias idênticas de um mesmo componente, sendo as suas saídas comparadas e votadas. Este paradigma é conhecido pelo modelo de Réplicas Activas (*Active Replication*). A comparação de assinaturas é usada para validar as mensagens. Se dois componentes produzem assinaturas idênticas, o Sistema de Comunicação em Difusão selecciona uma das mensagens

[11] Open Systems Interconnection.

emitidas e entrega-a aos seus destinatários (possivelmente componentes, também eles, replicados). Este modelo detecta a falha de um componente quando uma mensagem produzida por este está em desacordo com as restantes, ou quando esta nem sequer chega a ser produzida. Isto permite ao sistema tolerar hospedeiros de falha não controlada, sendo o seu funcionamento vigiado pelos CLMs. Se componentes de *software* são executados em hospedeiros de falha silenciosa, então os replicados activos podem ser usados sem recorrer a votação, com o grau de replicação suficiente para manter a disponibilidade do componente.

A segunda técnica usada no sistema Delta-4 consiste no modelo das Réplicas Passivas (*Passive Replication*). Neste modelo, cada componente pode ser replicado, mas apenas uma cópia está activa, enviando periodicamente uma representação do seu estado para as restantes. Por este motivo, os modos de falha estão limitados à hipótese de falha silenciosa. Se o nó hospedeiro de um componente activo falha, uma das cópias é activada. Não existe, no entanto, nenhum mecanismo implícito para detectar a falha de um nó hospedeiro.

A terceira técnica, designada por Replicação Semi-Activa (*Semi-Active Replication*), possui facetas de ambas as técnicas anteriores. Nesta técnica, apenas uma réplica (designada por *líder*) processa as mensagens e gera respostas. Na ausência de faltas, as restantes réplicas (designadas por *seguidores*) não produzem mensagens, sendo o seu estado actualizado através do processamento das mensagens (por uma ordem especificada pelo líder) ou após receber notificações contendo alterações ao estado provenientes do líder. Esta técnica pretende manter os seguidores sincronizados com o líder, de modo a permitir uma rápida recuperação de falhas, sem obrigar que todas as operações sejam completamente determinísticas.

Na arquitectura Delta-4, a utilização destas técnicas é complementada por um potente sistema de administração, responsável por diagnosticar as faltas no sistema e por realizar as necessárias reconfigurações de sistema quando estas ocorrem. Por outro lado, a complexidade destes mecanismos é escondida ao programador de aplicações pelo ambiente de desenvolvimento (DELTASE).

2.5 Discussão

A replicação de dados pode ser utilizada, como técnica para fornecer tolerância a faltas ou para aumentar o desempenho do sistema, num largo leque de aplicações, incluindo, entre outras, bases de dados (Stonebraker, 1979; Borr, 1981), sistemas de ficheiros (Marzullo & Schmuck, 1988; Liskov *et al.*, 1990), sistemas transaccionais orientados aos objectos (Little, 1991), sistemas operativos (Tanenbaum *et al.*, 1992), sistemas de memória partilhada distribuída (Guedes & Castro, 1993), e aplicações de trabalho cooperativo (Cosquer & Veríssimo, 1994). Naturalmente, cada aplicação possui diferentes requisitos, impondo não só diferentes modelos de coerência de memória mas também diferentes critérios de desempenho. Mesmo para uma dada aplicação, o algoritmo de gestão de replicação adequado depende de factores operacionais, tais como o tipo de faltas que se deseja tolerar, os atrasos na comunicação e os padrões de acesso aos dados.

Apesar desta (necessária) diversidade, existem múltiplos aspectos comuns às várias alternativas. Isto sugere que, caso se consiga identificar um conjunto significativo destes requisitos comuns, e caso se consiga satisfazê-los através de um conjunto de serviços concretizados de forma modular e configurável, pode ser possível obter uma arquitectura com a capacidade de suportar eficientemente um leque alargado de algoritmos de gestão da replicação.

Identificaram-se dois destes requisitos macroscópicos: filiação e difusão fiável em grupo. Estes requisitos não se projectam num único serviço, mas sim num alargado conjunto de serviços, com diferentes propriedades de ordenação, coerência e sincronismo. Por sua vez, estes serviços podem ser obtidos através de composição de sub-serviços tais como detectores de falhas, serviços de transporte em difusão, serviços de ordenação de mensagens, etc. Uma plataforma que oferece este tipo de serviços designa-se, devido às suas óbvias características multi-participante, por uma plataforma de grupos.

As vantagens das plataformas de grupos para suportar a construção de aplicações distribuídas têm sido demonstradas por vários projectos. Foi referida a experiência obtida com o projecto DELTA-4. Outros projectos obtiveram resultados semelhan-

tes (Mishra *et al.*, 1993; Birman & Renesse, 1994). Infelizmente, ainda existem poucas plataformas de grupos com a capacidade de operar em sistemas de grande escala geográfica (sistemas como os agora referidos foram concebidos para oferecerem o seu desempenho óptimo sobre redes locais). Identifica-se pois a necessidade de desenvolver algoritmos adequados à operação em grande escala que permitam recolher a experiência necessária para avaliar a "orientação para os grupos" sobre este tipo de redes.

2.6 Sumário

Neste capítulo ilustrou-se a necessidade de desenvolver um sistema de suporte à construção de aplicações distribuídas fazendo uma breve descrição de uma área da maior relevância, nomeadamente a replicação de dados. Muitos dos requisitos impostos pelas aplicações que usam replicação, tais como serviços de filiação em grupo e de difusão fiável, têm vindo a ser fornecidos por plataformas globalmente designadas por *plataformas de grupos*. O Capítulo fez também uma resenha dos principais serviços oferecidos por estas plataformas. Finalmente, focou-se um caso particular de integração destes serviços numa arquitectura capaz de oferecer tolerância a faltas de um modo eficiente.

3

Arquitectura

3.1 Enquadramento

Esta tese enquadra-se num projecto de investigação levado a cabo pelo Grupo de Sistemas Distribuídos e Automatização Industrial do INESC, globalmente designado por NAVTECH /Romance. A arquitectura NAVTECH pretende oferecer uma plataforma orientada aos grupos com a capacidade de operar em redes de grande escala. O sistema Romance pretende fornecer suporte para o desenvolvimento e uso de objectos replicados.

Tanto o NAVTECH como o Romance encontram-se ainda num estado embrionário do seu desenvolvimento. À medida que os seus componentes vão sendo desenvolvidos — e esta tese contribui com vários componentes para estas arquitecturas — é recolhida experiência que permite refinar os modelos. Apesar disso, julga-se relevante descrever aqui sumariamente o seu estado actual. Isto porque, apesar dos módulos que se apresentam nos capítulos seguintes possuirem um valor intrínseco, é interessante ilustrar como estes se podem integrar e como é possível obter valor acrescentado através da sua combinação.

3.2 NavTech

3.2.1 Motivação

No capítulo anterior foram apresentados diversos serviços orientados aos grupos e motivou-se a sua utilidade para a construção de aplicações distribuídas. No passado, foram desenvolvidos diversos sistemas concretizando estes serviços (Cristian *et al.*, 1990; Birman *et al.*, 1991a; Powell, 1991; Kaashoek & Tanenbaum, 1991; Mishra *et al.*, 1993). A maioria destes sistemas exibe um conjunto de limitações no funcionamento em grande escala, nomeadamente uma severa degradação do desempenho ou mesmo funcionamento incorrecto na presença de impedimentos como atrasos elevados ou partições na rede. A arquitectura NAVTECH pertence a uma nova geração de sistemas que tentam resolver os problemas colocados pelas redes de grande escala (Amir *et al.*, 1992; Amir *et al.*, 1993; Babaoğlu & Toueg, 1993; Schiper & Ricciardi, 1993; Schiper & Sandoz, 1993). O desenvolvimento de uma nova arquitectura é motivado pela convicção — e o trabalho apresentado nos capítulos seguintes confirma esta convicção — que para oferecer serviços orientados aos grupos em redes de grande escala é necessário tomar explicitamente em consideração as características destas redes.

3.2.2 Características principais

A arquitectura possui um conjunto de características cuja combinação a torna uma solução englobante para a construção e execução de aplicações fiáveis em grande escala. Nomeadamente, a arquitectura NAVTECH (Veríssimo & Rodrigues, 1995) é:

- *Orientada ao problema.* A arquitectura concretiza um conjunto de *perfis* de utilização, cada um destinado a uma classe de aplicações. A oferta de diversos perfis tem como objectivo permitir ao utilizador escolher o compromisso mais adequado entre desempenho e qualidade de serviço, com a garantia de que o conjunto de serviços oferecidos em cada perfil é compatível. Os perfis previstos nesta data são os seguintes: *melhor-esforço*, oferecendo primitivas eficientes mas pouco fiáveis; *partição-primária*, que garante a animação numa única partição

do sistema; e *múltipla-partição*, para aplicações onde é fundamental garantir o progresso (mesmo que isso signifique enfraquecer a coerência e ter de executar procedimentos de reconciliação).

- *Modular e componível.* Os serviços são concretizados por um conjunto de módulos, que podem ser combinados para suportar um vasto leque de estilos de interacção. Por exemplo, o serviço de filiação suporta dois tipos de entidades, *membros de grupos*, e *remetentes para grupos* a partir dos quais é possível suportar estilos de interacção tais como *conversação*, *cliente-servidor*, ou *disseminação*.

- *Pragmática.* A arquitectura está a ser desenvolvida tendo em atenção o estado actual da tecnologia de redes. O modelo seguido torna visíveis as propriedades das redes que são relevantes para o desenvolvimento de sistemas em grande escala. Por exemplo, a arquitectura explora o carácter hierárquico das redes existentes e a sua organização em forma de aglomerado de redes locais (rápidas e fiáveis) interligadas por redes de longo alcance (lentas e não fiáveis). Por outro lado, a arquitectura distingue os *processos* dos *nós que os albergam*. Esta dualidade é utilizada para minimizar o tráfego na rede e para simplificar os protocolos de filiação.

- *Configurável.* Os módulos constituintes da arquitectura podem ser configurados para se ajustarem a requisitos das aplicações que suportam. Um exemplo de configuração pode ser encontrado por Cosquer *et al.* (1995b).

- *Orientada-aos-grupos.* A arquitectura NAVTECH suporta, mas não força, um estilo de programação orientado aos grupos. Apesar de disponibilizar primitivas de comunicação ponto-a-ponto, a arquitectura encoraja o programador a usar os serviços de filiação e comunicação em grupo.

3.2.3 Ambiente alvo

Um factor crucial para obter um bom desempenho da arquitectura é uma correcta modelação do ambiente em que se vai executar. Nesta secção caracterizam-se os ambientes para os quais a arquitectura NAVTECH está concebida.

3.2.3.1 Infra-estrutura de comunicação

A identificação das propriedades da infra-estrutura de comunicação é fundamental para obter um desempenho adequado dos protocolos a desenvolver. Uma aproximação possível consiste em assumir que todas as máquinas se encontram mutuamente ligadas por um canal lógico que pode, eventualmente, falhar (e posteriormente recuperar). Esta abstracção, pela sua simplicidade, pode ser facilmente concretizada mas fornece pouca informação acerca das propriedades dos componentes físicos que a concretizam. Como se verá na sequência da tese, existem diversos atributos que são específicos de certas classes de redes (por exemplo, a capacidade de difusão com sobrecarga desprezável nas redes locais) e que podem ser explorados para optimizar o desempenho de diversos protocolos. De facto, a perspectiva de uma rede completamente ligada é uma abstracção que descreve o serviço da rede mas não a sua estrutura física. Neste aspecto, a estrutura da rede global[1] possui características relevantes:

- não é homogénea, possuindo, para além de uma rede de elos ponto-a-ponto de longa distância, um conjunto de redes de diferentes características tais como redes locais, redes metropolitanas, entre outras;
- a grande maioria dos nós estão localizados dentro de domínios privados, associados a organizações, que estão separados do resto da rede por conversores de protocolos (*gateways*) ou encaminhadores (*routers*);
- em grandes organizações, esta separação encontra-se por vezes ao nível do departamento;
- em muitos sistemas, existe em cada nó um processo servidor (Birman & Joseph, 1987; Peterson *et al.*, 1989), um gestor de periférico (*device driver*) dedicado (Vogels *et al.*, 1992), ou uma carta de comunicações (Powell, 1991) responsável por todas as interacções com a rede.

Esta estrutura irá permanecer em utilização por tempo suficiente para merecer uma sistematização que potencie uma arquitectura de comunicação mais eficiente (Veríssimo

[1] Por exemplo, a Internet no caso de redes baseadas em TCP/IP.

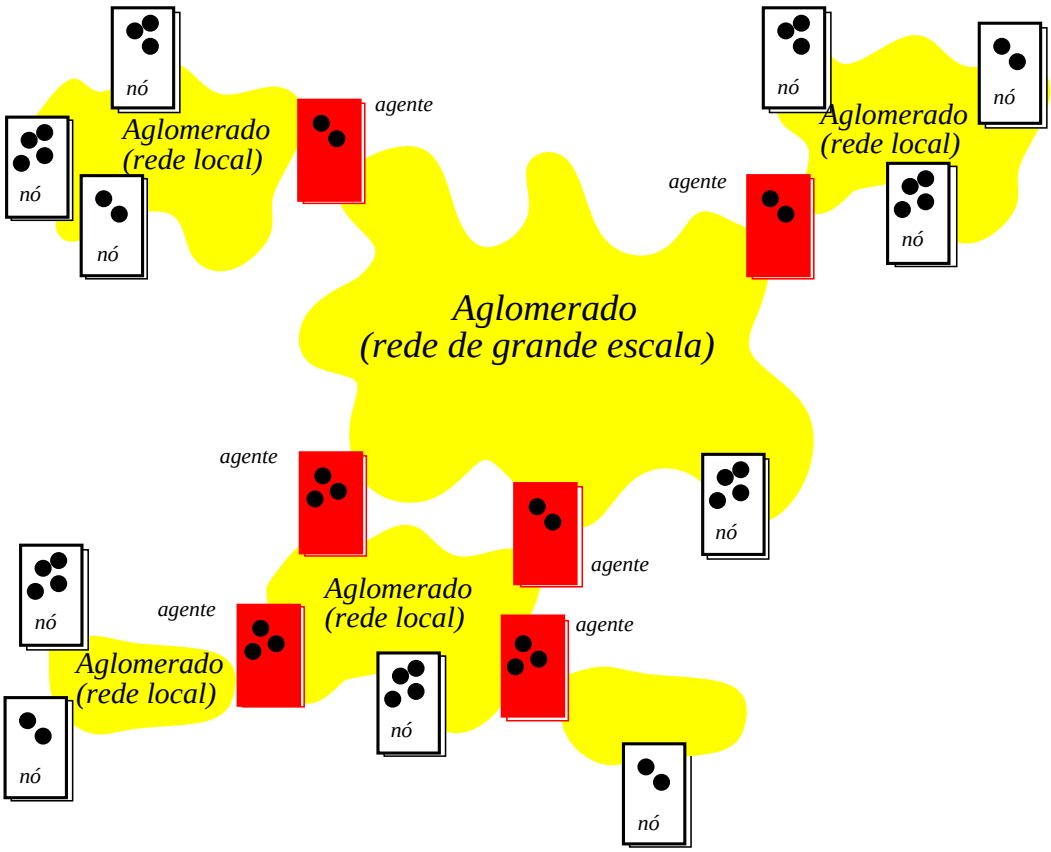


Figura 3.1: Modelo de rede na arquitectura NAVTECH.

& Vogels, 1993). Sendo assim, no NAVTECH utiliza-se a seguinte modelação das redes alvo, ilustrada na Figura 3.1:

- A rede alvo é constituída pela interligação hierárquica de *aglomerados* de *nós*.
- Cada aglomerado é constituído por um conjunto de nós interligados por uma *rede*. Alguns destes nós podem ser *agentes* que representam outros aglomerados.
- No NAVTECH consideram-se duas classes de redes, designadas genericamente por *redes locais*, e *redes de grande escala*. Caso venha a ser necessário, poderão vir a criar-se mais classes no futuro.
- Todo o tráfego de e para o aglomerado é realizado através de um conjunto bem identificado de *encaminhadores*, os quais podem executar protocolos dedicados. Estes encaminhadores dizem-se os *agentes* do aglomerado.
- Em cada *nó* executam-se um ou mais *processos participantes* na computação.

3.2.3.2 Propriedades dos nós

Na arquitectura NAVTECH assume-se que cada nó é constituído por uma máquina com modo de falha silenciosa (Powell, 1991) na qual se executam um ou mais processos participantes na computação. Assume-se também que a falha de um processo pode ser detectada de modo fiável na máquina em que este se executa. Esta distinção entre nós e participantes é um aspecto do modelo com importância tanto teórica como prática.

É teoricamente relevante porque separa o domínio onde os resultados de impossibilidade (Fischer *et al.*, 1985) são aplicáveis e o domínio onde as falhas podem ser detectadas com exactidão. Para além disso, também a comunicação entre processos de um mesmo nó pode ser executada de um modo fiável. Deste modo, restringe-se o problema de garantir a fiabilidade à comunicação entre nós.

É de relevância prática porque existe geralmente um factor multiplicativo entre o número de nós e o número de participantes envolvidos na comunicação. Este factor aumenta com a capacidade dos nós e com a complexidade das aplicações. Sempre

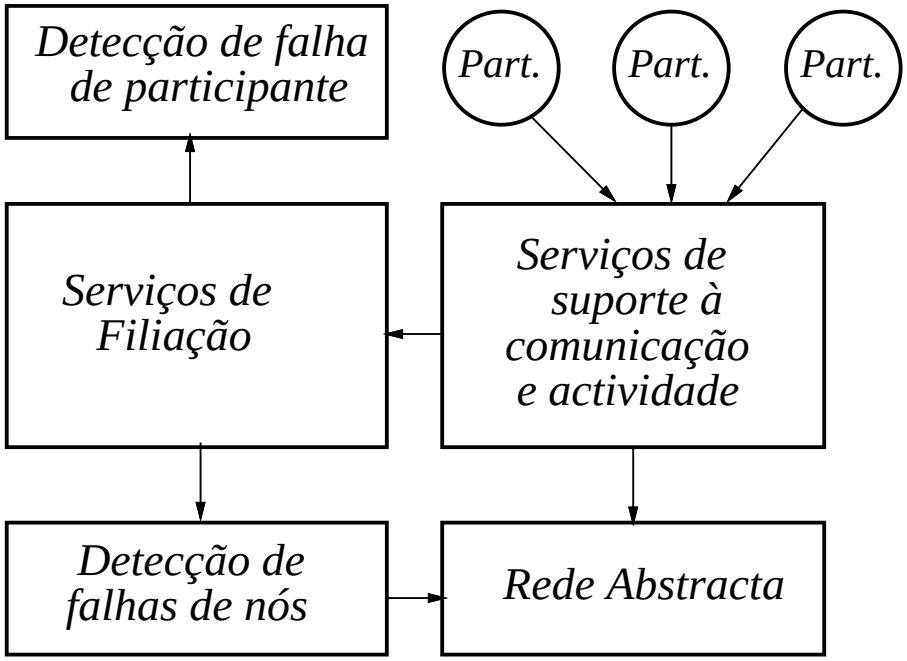


Figura 3.2: Blocos principais da arquitectura NAVTECH.

que este factor é significativo, a distinção entre participantes e nós permite aumentar o desempenho e a escalabilidade do sistema. Por exemplo, para endereçar vários participantes numa mesma máquina basta enviar uma única mensagem para o nó onde estes residem. Do mesmo modo, se o nó falhar é possível indicar que todos esses participantes ficaram simultaneamente inacessíveis.

3.2.4 Blocos básicos da arquitectura

Os principais blocos da arquitectura NAVTECH, tal como ilustrado na Figura 3.2 (na qual as setas representam relações "depende de"), são os seguintes:

Rede Abstracta Construída sobre a infra-estrutura física, oferece uma rede virtual que suporta difusão, endereçamento lógico, e tem uma capacidade moderada de corrigir perturbações na rede.

Detector de Falhas dos Nós Responsável por aferir a conectividade entre os nós.

Detector de Falha de Participantes Responsável por detectar a falha de processos locais ao nó.

Serviços de Filiação Responsável por manter a filiação de grupos de *participantes*. Este bloco usa a informação fornecida pelo Detector de Falhas dos Nós e o Detector de falha de Participantes para manter a informação de filiação actualizada. É também responsável por satisfazer critérios de animação impostos pelas aplicações, tais como *maioria*, *quorum*, etc.

Serviços de Suporte à Comunicação e Actividade Um conjunto de serviços concretizado por protocolos tais como: comunicação em grupo, protocolos de ordenação, sincronização de relógios, invocação remota, gestão da replicação, sincronização de processos.

Uma vez que a arquitectura NAVTECH reconhece diversas classes de redes, existirão diversas concretizações das funcionalidades fornecidas por estes blocos. Neste momento existem concretizações parciais de alguns blocos para as duas classes inicialmente previstas, nomeadamente as redes locais e as redes de grande escala.

Uma determinada configuração da arquitectura pode usar exclusivamente os protocolos de uma classe de rede (neste caso, os protocolos destinados às redes de grande escala são os mais genéricos) ou usar ambos os protocolos de um modo hierárquico, necessitando para tal de uma camada de integração, tal como se ilustra na Figura 3.3.

3.2.5 Concretização

Cada um dos blocos anteriores será concretizado por um ou mais *módulos*, cumprindo funções específicas, os quais podem ser interligados para oferecer os serviços desejados. Apesar das vantagens do desenvolvimento modular serem bem conhecidas, podem levantar-se dúvidas acerca do impacto desta aproximação no desempenho final do sistema. No entanto, a experiência obtida em projectos como o x-Kernel (Hutchinson & Peterson, 1988), Consul (Mishra *et al.*, 1993), Horus (Renesse *et al.*, 1992), e no próprio desenvolvimento do xAMp (Fonseca, 1994), demonstram que este aspecto é desprezável quando comparado com os benefícios derivados da modularidade tais como, entre outros: configurabilidade, portabilidade, e facilidade de

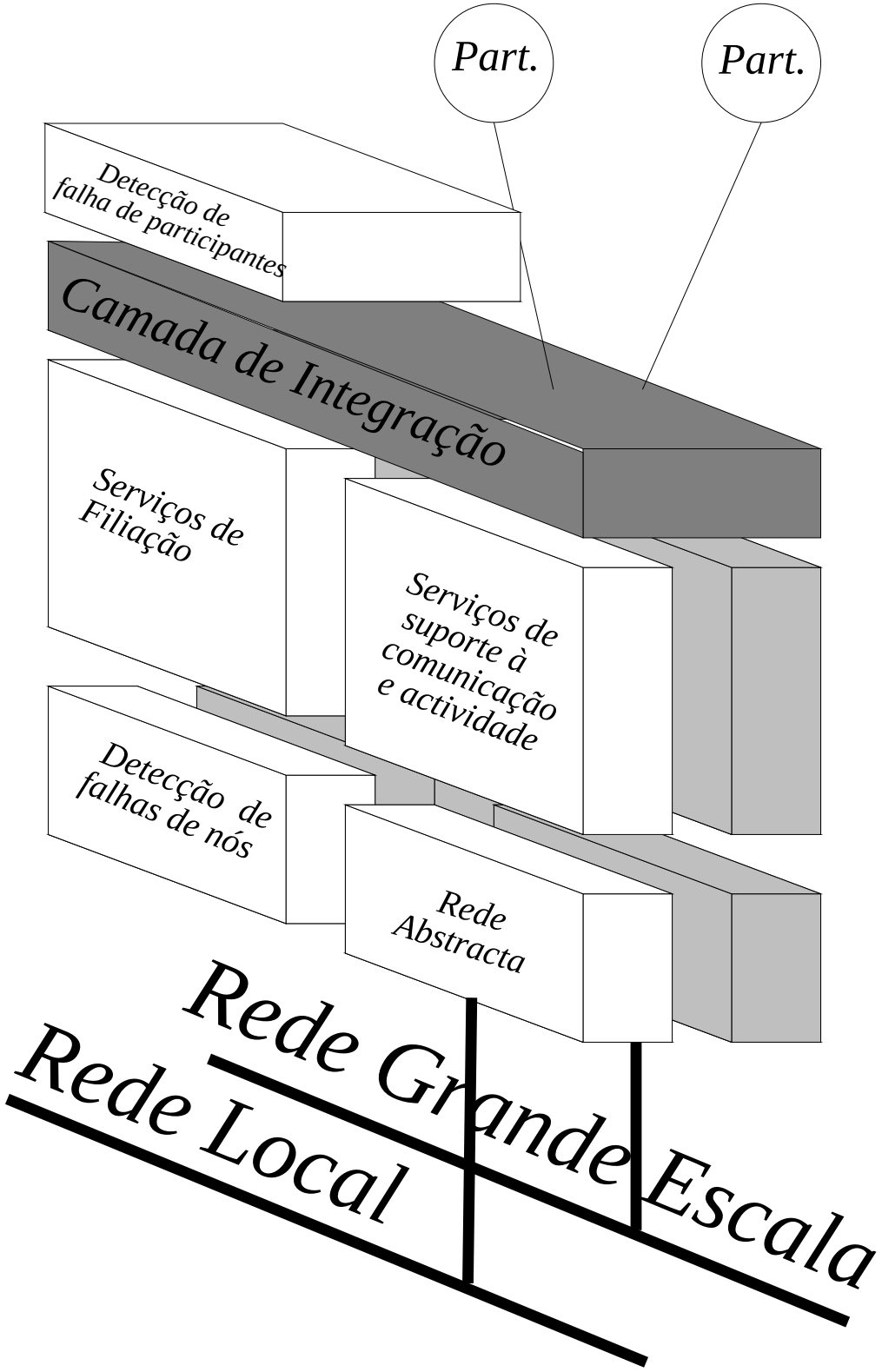


Figura 3.3: Especialização por classe de rede NAVTECH .

manutenção. O desenvolvimento de sistemas de comunicação modulares é simplificado pelo uso de um *ambiente de suporte local* adequado; para o desenvolvimento do NAVTECH pensa-se usar uma combinação de serviços desenvolvidos localmente com serviços disponibilizados pelo Horus. A justificação desta escolha encontra-se fora do âmbito desta tese (para uma abordagem aos ambientes de suporte locais veja-se o trabalho de Fonseca (1994)).

3.3 Romance

3.3.1 Motivação

Os serviços fornecidos pela arquitectura NAVTECH podem ser expandidos através de um leque de ferramentas de carácter mais especializado. A vantagem principal de fornecer este tipo de ferramentas consiste em oferecer abstracções mais poderosas, escondendo a utilização dos mecanismos elementares. Deste modo, o utilizador pode beneficiar dos serviços de filiação e comunicação em grupo sem ter de abarcar todas as diferenças — frequentemente subtis — entre as diversas qualidades de serviço disponíveis.

Nesta secção faz-se referência ao sistema Romance (Rodrigues & Veríssimo, 1993a; Rodrigues & Veríssimo, 1993b) (do Inglês, "Replicated Object MANagement Configurable Environment"), o qual pretende fornecer um conjunto de serviços que facilite ao programador a utilização de componentes replicados na sua aplicação.

A replicação de objectos em sistemas distribuídos tem sido usada para melhorar o desempenho ou para aumentar a tolerância a faltas em diversas áreas, tais como: gestão de dados persistentes (Bernstein *et al.*, 1987), gestão de computações replicadas (Powell, 1991) ou memória partilhada distribuída (Nitzberg & Lo, 1991). Estas áreas são complementares e nenhuma cobre totalmente o espectro de aplicações que pode usufruir da gestão da replicação de dados. A título de exemplo podemos referir que a maioria dos sistemas de gestão de objectos persistentes não oferece mecanismos eficientes de partilha de objectos voláteis e, por outro lado, sistemas de memória virtual distribuída

raramente oferecem tolerância a faltas (no entanto, veja-se, por exemplo, o trabalho de Neves *et al.* (1994)). O Romance pretende investigar modelos que conciliem estas diferentes aproximações.

Um outro objectivo fundamental do sistema Romance é exercitar a arquitectura NAVTECH de modo a avaliar o seu grau de adequação ao suporte à replicação. Assim, espera-se vir a fazer uso extensivo dos serviços de filiação e comunicação em grupo no desenvolvimento dos componentes do sistema Romance, aproveitando-se as características da arquitectura NAVTECH para suportar diversos algoritmos de gestão da replicação sobre redes de grande escala.

3.3.2 Aspectos chave

Como se referiu, o principal objectivo do Romance é fornecer suporte à programação e utilização de componentes replicados. Para tal, o Romance pretende disponibilizar uma biblioteca com diversos protocolos de gestão da replicação e protocolos de invocação remota capazes de interagir com componentes replicados. Para facilitar a integração dos mecanismos fornecidos na biblioteca, propõe-se que esta esteja construída segundo uma aproximação orientada aos objectos. Assim, pressupõe-se que os objectos a replicar são constituídos por uma zona de dados, encapsulada por um conjunto de operações ou métodos designados pela *interface do objecto*. Este modelo é semelhante ao utilizado por sistemas como Argus (Liskov, 1985), Emerald (Black *et al.*, 1986), Clouds (Dasgupta *et al.*, 1988), SOS (Shapiro, 1989), e Arjuna (Shrivastava *et al.*, 1991), Comandos (Sousa *et al.*, 1993).

As vantagens deste tipo de aproximação são bem conhecidas: é possível disponibilizar diversas concretizações para uma mesma interface e, através do recurso a serviços de suporte, enriquecer os objectos com atributos tais como, distribuição transparente, persistência, controlo de concorrência, etc (Matos, 1994). Naturalmente, o sistema Romance usa técnicas semelhantes para, através do recurso à replicação, enriquecer os objectos com os atributos de tolerância a faltas e elevado desempenho.

Tal como se referiu, de acordo com a aproximação seguida, os objectos só podem ser

acedidos através de uma interface. De modo a suportar a replicação dos objectos de uma maneira versátil, considera-se que uma concretização dessa interface deve permitir pelo menos dois métodos distintos de acesso ao objecto: (i) acesso através de um mecanismo de invocação remota que contacta um ou mais processos onde residem cópias do objecto; (ii) manutenção de uma cópia local do objecto, a qual é mantida coerente em relação às restantes cópias através de um mecanismo de gestão da replicação. Ambos os métodos de acesso devem ser permitidos e, eventualmente, usados em simultâneo por diferentes processos. Cada um destes métodos será analisado individualmente e seguidamente mostra-se como estes coexistem no sistema Romance.

3.3.2.1 Acesso remoto

O acesso a objectos remotos é executado através invocações remotas, tal como ilustrado na Figura 3.4. Este método é suportado por rotinas de adaptação no lado do cliente e pelos correspondentes embaixadores (*ambassador*) (Coplien, 1992) nos processos que possuem cópias do objecto, designados neste contexto por processos servidor. As rotinas de adaptação encapsulam os pedidos dos clientes em mensagens que são enviadas para um ou mais embaixadores. Estes, por sua vez, são responsáveis por desencapsular a mensagem, invocar o método associado ao pedido e devolver o resultado ao cliente.

As rotinas de adaptação interagem com os embaixadores associados de acordo com um protocolo pré-determinado. Por sua vez, os embaixadores podem também trocar informação entre si. Estes protocolos poderão variar com o tipo de objecto e deverá ser possível suportar diferentes métodos de acesso remoto para cada objecto. Na literatura podem-se encontrar diversos protocolos que suportam a invocação remota de objectos replicados (Cooper, 1984; Ladin *et al.*, 1990). No entanto, a grande maioria encontra-se optimizada para um determinado protocolo de gestão da replicação. Geralmente, é o próprio protocolo de invocação remota que mantém a coerência das réplicas do objecto.

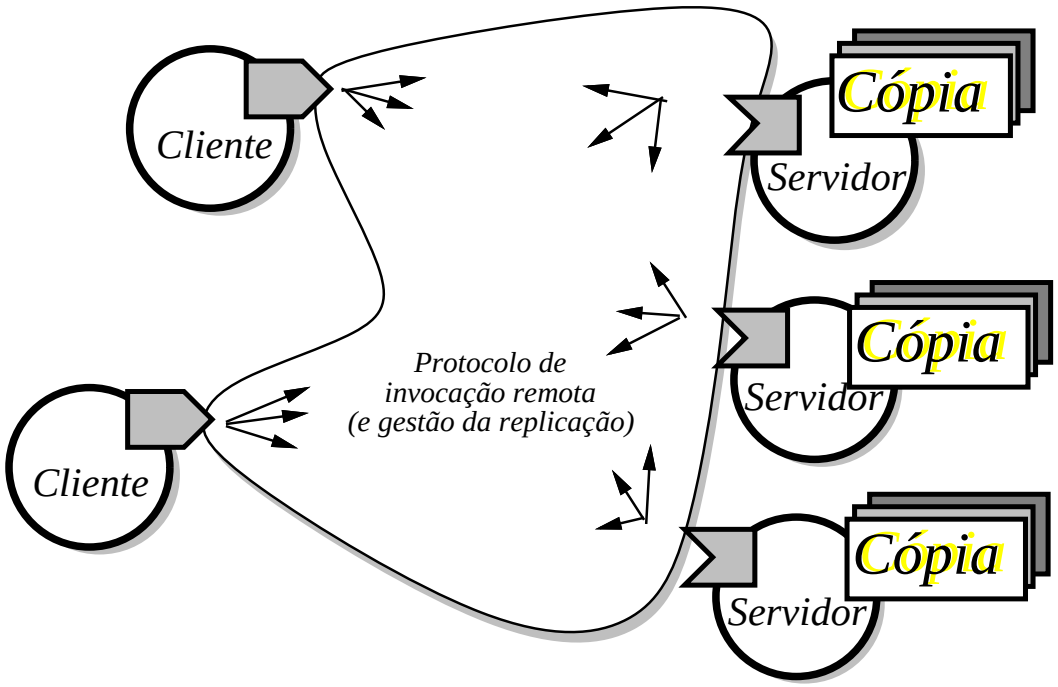


Figura 3.4: Acesso via invocação remota.

3.3.2.2 Cópia local

Um outro método de aceder a objectos no sistema Romance consiste em manter uma cópia local do objecto. Isto pode aumentar o desempenho no acesso, sobretudo se o objecto for acedido maioritariamente em modo de leitura. Ao se criar uma cópia local é necessário activar um mecanismo de gestão da replicação que mantenha as diversas cópias coerentes de acordo com um determinado critério de coerência de memória. No sistema Romance possibilita-se que cada objecto utilize o mecanismo de gestão da replicação mais apropriado para a aplicação a que se destina.

De modo a manter a coerência entre as cópias de um mesmo objecto, os processos que as detêm devem comunicar entre si. Neste modelo, ilustrado na Figura 3.5, o cliente acede à cópia local através de um serviço de gestão da replicação que, em função do modelo de coerência escolhido e dos mecanismos utilizados para concretizar esse modelo, decide quando deve interagir com as restantes cópias. Deste modo, os pormenores da coordenação entre réplicas não são observados pelos clientes. Note-se que nem todos

Figura 3.5: Acesso via cópia local.

os acessos à cópia local obrigarão a sincronização com as restantes cópias. Tipicamente, algumas operações poderão ser executadas localmente (geralmente, operações de leitura).

3.3.2.3 Acesso misto

Os dois tipos de acesso são conciliados no sistema Romance, de modo a poderem coexistir e a poderem ser utilizados simultaneamente, tal como se ilustra na Figura 3.6. De acordo com este modelo, diversas cópias de um objecto podem existir no sistema. Estas cópias são mantidas coerentes usando um determinado protocolo de gestão da replicação. Este protocolo exporta uma interface que só pode ser acedida pelos processos que detêm uma cópia do objecto.

Algumas destas cópias podem ser acedidas por clientes locais. Por outro lado, clientes remotos são também suportados, permitindo que alguns destes processos (eventualmente todos) possam receber invocações através de um canal logicamente diferente, suportado por rotinas de adaptação e embaixadores distintos dos utilizados para manter a coerência das réplicas. Tipicamente, este acesso remoto será suportado pelo protocolo de invocação remota descrito na Capítulo 6, o qual é constituido por um sub-protocolo de acesso remoto e por um sub-protocolo de detecção de reinvocações. O acesso misto permite separar os aspectos de invocação remota dos aspectos de gestão da replicação e permite suportar simultaneamente clientes com propriedades distintas (em termos de acessibilidade, segurança, etc).

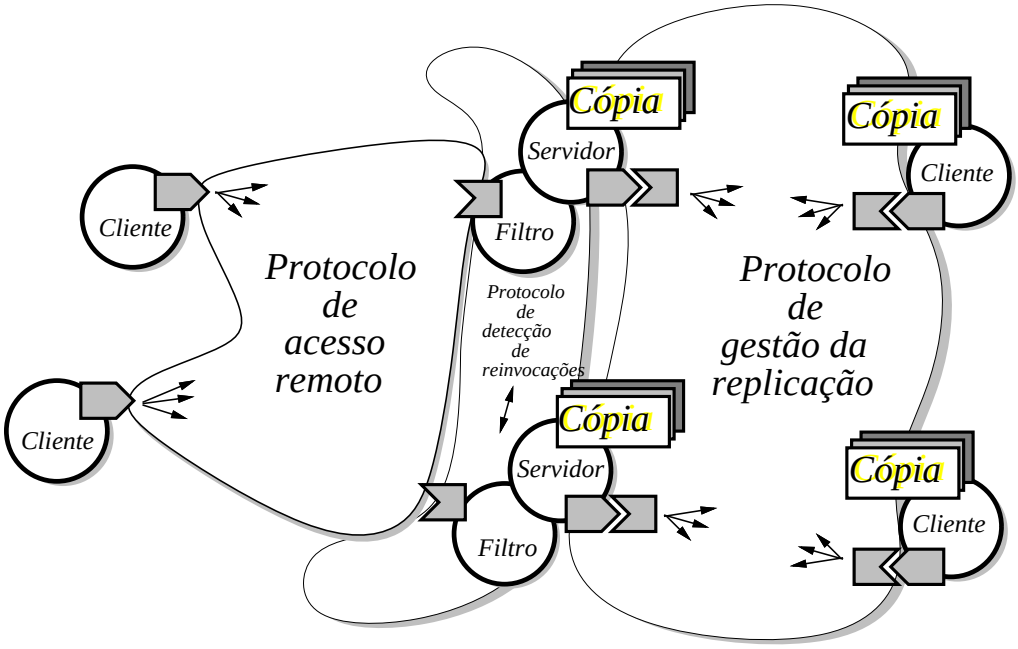


Figura 3.6: Acesso misto.

3.3.3 Concretização

Pretende-se que o Romance venha a constituir um ambiente que encorage o programador a *usar* e *programar* objectos que podem ser replicados. Estes dois objectivos são suportados através de mecanismos distintos. O uso de objectos replicados é encorajado fazendo com que estes sejam acedidos do mesmo modo que os objectos não replicados. A programação e refinamento de mecanismos de gestão da replicação é encorajada oferecendo ao programador uma biblioteca de mecanismos pré-programados assim como uma metodologia que permita ao programador acrescentar facilmente novos mecanismos a esta biblioteca.

Para testar a exequibilidade desta aproximação, realizou-se um protótipo do sistema Romance que permite utilizar e desenvolver objectos replicados usando a linguagem C++ (Ellis & Stroustrup, 1990). Para permitir a transparência à distribuição e replicação, usou-se um idioma (Coplien, 1992) de C++ que permite separar facilmente as interfaces das concretizações. Segundo este idioma, cada classe dá origem a um conjunto de classes que inclui os seguintes elementos: uma classe *envelope*, utilizada pelo cliente que mais não faz que reencaminhar as invocações para uma classe concretização; uma classe

assinatura, que descreve a interface da classe, a qual deve ser suportada pelo envelope e pelas concretizações; uma ou mais classes de concretização, também designadas por *cartas*, que concretizam a interface. As cartas podem concretizar a interface de modos distintos, por exemplo, fazendo um acesso remoto, mantendo uma cópia local do objecto, etc.

Para facilitar a programação, o sistema permite que as interfaces sejam descritas em CORBA[2] IDL[3] (OMG, 1991). Um tradutor de CORBA IDL para C++ gera todas as classes necessárias para suportar o idioma envelope-cartas descrito atrás. Este tradutor foi desenvolvido usando o "IDL Compiler Front End"[4] fornecido pela Sun[5] para o grupo OMG[6], o qual permite refinar a componente geradora de código sem alterar o interpretador, o que garante a conformidade das especificações com a sintaxe CORBA.

Este protótipo executa-se sobre os componentes para rede local do NAVTECH e usa um mecanismo de invocação remota bastante simplificado. Foram desenvolvidas concretizações rudimentares de alguns mecanismos simples de gestão de replicação.

3.4 Discussão

O desenvolvimento de um protótipo do Romance reforçou a convicção de que um sistema com as suas características acrescenta um valor significativo aos serviços oferecidos pela arquitectura NAVTECH . Experiências complementares (Little, 1991; Hagsand *et al.*, 1992; RDO, 1993; Maffeis, 1995) demonstram a validade desta aproximação. Para além disso, a experiência obtida com este protótipo permitiu identificar alguns aspectos críticos para o sucesso de um sistema deste tipo:

- *Necessidade de um sólido modelo de programação.* O desenvolvimento do protótipo deu ênfase aos aspectos relacionados com a replicação. No entanto, tornou-se

[2]Do Inglês, "Common Object Request Broker Architecture".
[3]Do Inglês, "Interface Description Language".
[4]Mais precisamente, o tradutor foi baseado numa versão já modificada desenvolvida no INESC (Trancoso & Sequeira, 1993).
[5]Sun Microsystems, Inc.
[6]Do Inglês, "Object Management Group".

claro que estes aspectos devem ser oferecidos ao programador integrados num modelo de programação coerente, o que pressupõe um esforço mais abrangente e multi-disciplinar. Paralelamente, desenvolveram-se no INESC um conjunto de esforços semelhantes, usando o mesmo tipo de aproximação, mas focando aspectos complementares como seja a persistência (Matos, 1994), o controlo de concorrência (Pereira, 1994; Silva *et al.*, 1995a), ou o processo de desenvolvimento (Silva *et al.*, 1995b). Neste momento, decorre um esforço de integração destas aproximações.

- *Necessidade de um mecanismo de invocação remota versátil e independente da gestão da replicação.* Esta observação motivou o desenvolvimento do protocolo de invocação genérica descrito no Capítulo 6.

- *Necessidade de um mecanismo de filiação de baixo-custo.* Sistemas do tipo do Romance tendem a promover um estilo de programação baseado na composição de um elevado número de componentes desenvolvidos de forma relativamente independente. A cada componente está associado um conjunto de rotinas de adaptação e embaixadores que requerem, em tempo de execução, a criação de diversos grupos de comunicação. Consequentemente, uma aplicação distribuída usando esta aproximação pode criar várias dezenas (ou mesmo centenas) de grupos que, na prática, possuem a mesma filiação. Soluções que permitem optimizar o desempenho de aplicações deste tipo estão a ser estudadas.

Um aspecto particularmente relevante para o desenvolvimento deste tipo de sistemas, identificado por todos os trabalhos atrás mencionados, é a necessidade de dispor de concretizações eficientes de serviços de filiação e comunicação em grupo. A arquitectura NAVTECH pretende dar resposta a este problema. Nos dois próximos capítulos, serão apresentados diversos componentes desta arquitectura, os quais seguem os princípios estruturantes por esta definidos:

- Consideram-se diferentes classes de redes, nomeadamente, redes locais e redes de grande escala. No Capítulo 4 apresentam-se componentes orientados à rede local e no Capítulo 5 componentes orientados à rede de grande escala.

- Usa-se a topologia das redes para optimizar o desempenho dos protocolos. Este princípio é aplicado em ambos os tipos de redes. Nas redes locais usa-se o facto da sua topologia facilitar a difusão física da informação. Nas redes de grande escala usa-se o facto da rede não ser completamente ligada para diminuir a informação de controlo trocada na rede.

- Toma-se em consideração o carácter hierárquico das redes de grande escala. Em particular, prevê-se a existência de padrões de tráfego exibindo elevado grau de localidade dentro dos aglomerados correspondentes a organizações. Reconhece-se também que a latência da comunicação entre dois participantes é extremamente dependente da sua localização na hierarquia.

- Segue-se uma aproximação modular, desenvolvendo diversos componentes que se podem compor para fornecer uma qualidade de serviço acrescentada.

3.5 Sumário

Neste capítulo apresentou-se, de um modo abreviado, a arquitectura NAVTECH /Romance. Esta apresentação restringiu-se a alguns aspectos chave: motivação, objectivos e decomposição funcional em blocos macroscópicos. Pretendeu-se assim enquadrar os três próximos capítulos, os quais apresentarão componentes desenvolvidos para esta arquitectura.

Notas

A arquitectura NAVTECH *resulta de um esforço liderado pelo Prof. Paulo Veríssimo, para o qual têm contribuído, para além do autor, diversos elementos do Grupo de Sistemas Distribuídos e Automatização Industrial, nomeadamente, F. Cosquer, H. Fonseca, J. Frazão, A. Sargento e W. Vogels. Por sua vez, o autor é o principal responsável pela concepção da grande maioria dos componentes da arquitectura (os quais serão apresentados nos capítulos seguintes). O sistema Romance foi proposto em "Replicated Object Management Using Group Technology",*

L. Rodrigues e P. Veríssimo, actas do 4º "Workshop on Future Trends of Distributed Computing Systems", Lisboa, Portugal, Setembro, 1993.

4

Comunicação em rede local

4.1 Enquadramento

No Capítulo 3 foi descrita a arquitectura NAVTECH para suporte à computação distribuída. Esta arquitectura contempla a existência de duas classes de protocolos, cada uma especializada para um determinado tipo de redes, nomeadamente, redes locais e redes de grande escala. Neste capítulo referem-se os componentes desenvolvidos para operarem sobre redes locais.

A grande maioria destes componentes foi obtida através da depuração e extensão do trabalho apresentado pelo autor na sua Tese de Mestrado (Rodrigues, 1990). Dado estes componentes não terem sido desenvolvidos exclusivamente no âmbito do trabalho de doutoramento, não será feita uma descrição pormenorizada dos mesmos. Existem no entanto razões que justificam uma descrição sumária destes componentes e a sua inclusão na dissertação:

- Estes componentes, na sua forma actual, são versões consideravelmente melhoradas das apresentadas anteriormente (Rodrigues, 1990). Este esforço de depuração e extensão, quer do ponto de vista da engenharia, quer do ponto de vista algorítmico, representou uma percentagem não desprezável do trabalho de doutoramento.

- O esforço de desenvolvimento destes componentes permitiu aos elementos do Grupo de Sistemas Distribuídos e Automatização Industrial do INESC adquirir um conjunto de lições que se reflectiram na génese da arquitectura NAVTECH. Este aspecto será retomado no final do capítulo.

- As opções tomadas tendo em vista as propriedades das redes locais são, em certa medida, o contra ponto das impostas pelas redes de grande escala. Uma breve panorâmica sobre estes componentes permite avaliar mais claramente as diferenças entre este tipo de soluções e as soluções que são descritas no Capítulo 5.

Os componentes desenvolvidos para redes locais foram os seguintes: um serviço de filiação de nós; um serviço multi-primitiva de comunicação e filiação em grupo; e um serviço de sincronização de relógios. Antes de descrevermos sumariamente estes serviços fazemos uma resenha das propriedades das redes locais relevantes neste contexto.

4.2 Particularidades das redes locais

Um dos principais objectivos da concepção dos componentes destinados a operar sobre redes locais foi o de garantir a sua portabilidade. Dada a multiplicidade de normas existentes (por exemplo, Ethernet (CSMA/CD, 1985), Token-Ring (Token-Ring, 1985), e Token-Bus (Token-Bus, 1985)) definiu-se uma *Rede Local Abstracta*, caracterizada por um conjunto de propriedades comuns à grande maioria das redes locais. Algumas destas propriedades foram utilizadas para simplificar os protocolos desenvolvidos, como se indicará adiante. Esta aproximação permitiu conciliar os requisitos de portabilidade com os requisitos de desempenho.

Uma justificação pormenorizada da interface escolhida para a rede abstracta e uma descrição da concretização deste interface sobre as diversas normas encontra-se fora do âmbito da tese. O leitor interessado poderá consultar a bibliografia (Veríssimo, 1989; Rufino, 1993). Na Tabela 4.1, enumeram-se de modo sumário as propriedades da rede abstracta.

- **RA1** - *Difusão*: Os destinatários que recebem uma trama não adulterada, recebem a mesma trama.
- **RA2** - *Detecção de erros*: Os destinatários conseguem detectar a eventual adulteração de uma trama pelo meio de comunicação.
- **RA3** - *Grau de omissão limitado*: O número de omissões na rede é limitado por uma constante k.
- **RA4** - *Bidirecionalidade*: Uma trama pode ser também entregue ao seu emissor.
- **RA5** - *Ordem*: Duas tramas entregues a dois destinatários são entregues pela mesma ordem a ambos.
- **RA6** - *Atraso limitado*: Os atrasos na rede são limitados por duas constantes, $[\Gamma^{min}, \Gamma^{max}]$. A variação dos atrasos na rede, $\Delta\Gamma$, é dada por: $\Delta\Gamma = \Gamma^{max} - \Gamma^{min}$.
- **RA7** - *Rigidez*: O intervalo de tempo real que separa a recepção de uma mesma trama em dois destinatários diferentes é limitado por uma constante $\Delta\Gamma_{rígido}$.

Tabela 4.1: Sumário das propriedades da rede abstracta.

4.3 Filiação de nós

Um dos componentes basilares do sub-sistema de comunicação em grupo é o protocolo de filiação de nós designado por MGS (Rodrigues *et al.*, 1993a) (do Inglês, Multicast Group of Stations). Este protocolo tem duas funções fundamentais:

- Mantém informação actualizada acerca da filiação de um conjunto seleccionado de nós da rede que participa na comunicação em difusão. O protocolo MGS assegura que alterações na filiação são indicadas de modo coerente a todos os nós na presença de chegada, partida ou falha de membros.

- Concretiza uma função que emparelha os identificadores únicos de nó em *endereços-abreviados*. Um endereço-abreviado identifica unicamente um nó dentro do grupo e necessita de um conjunto significativamente menor de dígitos binários para ser armazenado (tipicamente, é um inteiro no intervalo $[1, N]$ em que N é o número máximo de nós participando na difusão). O emparelhamento não é estático nem pré-definido, podendo-se atribuir um endereço-abreviado a novas

estações em qualquer momento.

Na arquitectura, o protocolo MGS facilita o desenvolvimento dos protocolos de filiação de processos oferecido pelas camadas superiores. A atribuição de endereços-abreviados permite optimizar os mecanismos de endereçamento em difusão, reduzindo o número de dígitos binários necessários para endereçar um conjunto de nós e facilitando o reconhecimento destes endereços.

O protocolo completo necessitaria de bastante espaço para ser descrito com um mínimo de precisão. Devido a esse facto, assinalam-se apenas as suas características principais.

O protocolo baseia-se numa primitiva de *transmissão-com-resposta*, que envia uma trama e espera uma confirmação positiva de cada destinatário durante um período de tempo pré-determinado. Omissões no canal são detectadas pela ausência de uma ou mais respostas e mascaradas através da retransmissão da trama. Falhas de nós são detectadas quando um destinatário persiste em não confirmar a recepção de uma trama após um número também pré-determinado de retransmissões. A detecção de falhas assume pois um comportamento síncrono do sistema, derivado das propriedades da rede abstracta e de uma correcta configuração dos nós (em termos de carga e escalonamento de actividades).

Os nós usam a primitiva de transmissão-com-resposta para, de um modo distribuído, trocarem intenções de alteração da filiação no grupo e para serializarem a confirmação dessas alterações. A filiação do grupo é mantida numa tabela replicada em todos os nós, a qual é também utilizada para concretizar o emparelhamento entre identificadores únicos e endereços-abreviados.

O uso de um protocolo baseado em confirmações positivas propicia uma rápida detecção de falhas e garante tempos de recuperação limitados. Em contrapartida, a latência da primitiva de transmissão-com-resposta aumenta com o número de participantes. A título de exemplo, uma concretização do MGS executando-se como um gestor de periférico UNIX num agregado de SUN Sparc-Stations apresenta uma latência para cada operação que varia entre 7.3 ms com dois nós até 20.3 ms com 13 nós, o que

representa um incremento de cerca de 1ms por cada nó adicional[1].

4.4 O xAMp

O xAMp[2] (Rodrigues & Veríssimo, 1992b; Vogels *et al.*, 1992; Rodrigues *et al.*, 1995a) é um serviço de comunicação em grupo versátil que oferece um leque de serviços de difusão abrangendo desde a difusão não fiável e não ordenada até à difusão atómica. Os protocolos interagem com o MGS e, tal como este, são baseados na primitiva de transmissão-com-resposta, privilegiando a garantia de tempos de terminação limitados e baseando-se no comportamento síncrono do sistema. Para além disso, a actual concretização não tolera partições na rede (a rede abstracta pode ser construída usando técnicas que reduzem a probabilidade de ocorrência de partições; por exemplo, usando duplicação do meio físico (Rufino, 1993)). A maioria dos serviços prestados são obtidos pela composição e parametrização desta primitiva elementar, pelo que a concretização resultante é significativamente homogénea e integrada. Os serviços do MGS são utilizados para concretizar um mecanismo de endereçamento em grupo extremamente eficiente baseado em *máscaras de dígitos binários* e para oferecer um serviço de filiação em dois níveis: *filiação de nós* (suportado pelo MGS) e *filiação de processos* (oferecido pelo xAMp). Nos parágrafos seguintes, resumem-se os serviços prestados pelo xAMp:

Em primeiro lugar, o xAMp presta serviços de *filiação*, permitindo a criação (e reconfiguração) em tempo de execução de grupos de processos. Durante o tempo de vida de um grupo, novos processos podem inscrever-se no grupo, abandonar o grupo ou falhar. Alterações na filiação são indicadas a todos os membros através de uma mensagem contendo a filiação do grupo, também denominada *vista* do grupo.

Em segundo lugar, o xAMp fornece um serviço de endereçamento em grupo que se pretende versátil. Todas as primitivas de difusão aceitam como endereço de des-

[1] Naturalmente, a latência da primitiva de transmissão-com-resposta depende tanto do tamanho da mensagem como das respectivas confirmações. No MGS, as confirmações são relativamente grandes (várias centenas de octetos, dependendo do tamanho máximo da tabela). Como se verá adiante, no xAMp (que possui confirmações de menor tamanho), obtêm-se valores mais favoráveis.

[2] O xAMp foi desenvolvido a partir de um outro serviço de comunicação em grupo também desenvolvido no INESC, o AMp (Veríssimo *et al.*, 1989).

Vistas de grupo coerentes

- **Px1** - Cada alteração na filiação de um grupo é indicada aos membros por uma mensagem ordenada de modo total.

Endereçamento

- **Px2** - Endereçamento selectivo: Os destinatários de uma mensagem são identificados por um par (g, ld), onde g identifica um grupo e ld é uma lista de membros desse grupo.
- **Px3** - Endereçamento lógico: Para cada grupo g existe um endereço lógico E_g, tal que E_g permite endereçar todos os membros de g sem saber o seu número ou identificação.

Segurança

- **Px4** - Não trivialidade: Todas as mensagens entregues foram enviadas por um participante correcto.
- **Px5** - Acessibilidade: Todas as mensagens entregues são entregues a um participante acessível para essa mensagem.
- **Px6** - Entrega: Uma mensagem é sempre entregue excepto no caso em que pelo menos um destinatário está inacessível ou no caso em que o emissor falha.

Sincronismo

- **Px7** - O intervalo de tempo entre a invocação do serviço e a consequente indicação de entrega a um destinatário, (T_e), assim como o intervalo entre duas destas indicações em destinatários diferentes (T_i), são:
  - Sincronismo fraco: ΔT_e e ΔT_i podem não ser desprezáveis em relação a $max\ T_e$.
  - Sincronismo forte: ΔT_e e ΔT_i são desprezáveis em relação a $max\ T_e$.

Acordo

- **Px8** - Unanimidade: Qualquer mensagem entregue a um destinatário é entregue a todos os destinatários correctos.
- **Px9** - Pelo-menos-N: Qualquer mensagem entregue a um destinatário é entregue a pelo menos N dos destinatários correctos.
- **Px9.1** - Pelo-Menos-Para: Dado um subconjunto D_{para} dos destinatários, qualquer mensagem entregue a um destinatário é entregue a pelo menos todos os elementos correctos de D_{para}.
- **Px10** - Melhor-Esforço-N: Qualquer mensagem entregue a um destinatário é entregue a pelo menos N dos destinatários correctos desde que o emissor não falhe.
- **Px10.1** - Melhor-Esforço-Para: Dado um subconjunto D_{para} dos destinatários, qualquer mensagem entregue a um destinatário é entregue a pelo menos todos os elementos correctos de D_{para} desde que o emissor não falhe.

Ordem

- **Px11** - Ordem total: Duas mensagens entregues a dois destinatários são entregues pela mesma ordem em ambos.
- **Px12** - Ordem causal: Duas mensagens entregues a um determinado membro de um grupo são entregues de acordo com a relação de causalidade potencial.
- **Px13** - Ordem FIFO: Se duas mensagens do mesmo emissor são entregues a um mesmo destinatário, estas são entregues por ordem de emissão.

Tabela 4.2: Propriedades do serviço xAMp

primitiva	acordo	ordem total	causal	reordenação
melhorEsforçoN	não (melhor esforço N)	não	FIFO	sem fila
melhorEsforçoPara	não (melhor esforço Para)	não	FIFO	sem fila
peloMenosN	não (garantido N)	não	FIFO	sem fila
peloMenosPara	não (garantido Para)	não	FIFO	sem fila
fiável	todos	não	FIFO	sem fila
causal	todos	não	sim	sem fila
atómica	todos ou nenhum	sim	sim	não
rígida	todos ou nenhum	sim	sim	sim
delta	todos ou nenhum	estampilha	estampilha	estampilha

Tabela 4.3: As primitivas de difusão do xAMp

tino a identificação de um grupo (*endereçamento lógico*) e, opcionalmente, uma lista de membros desse grupo (*endereçamento selectivo)*. O endereçamento lógico permite que todos os membros de um grupo sejam endereçados sem que o utilizador indique explicitamente o seu número ou identificação.

Finalmente, o xAMp fornece um leque de primitivas de comunicação que satisfazem um conjunto de propriedades identificadas como úteis para o desenvolvimento de algoritmos distribuídos. Estas propriedades são apresentadas de modo sumário na Tabela 4.2. As propriedades de segurança e correcção temporal (Px4, Px5, Px6 e Px7) são desejáveis na maioria dos sistemas de comunicação. Estas indicam que o utilizador pode confiar no sistema no sentido em que as mensagens não são adulteradas, perdidas de modo arbitrário ou geradas de modo espontâneo. As propriedades de correcção temporal asseguram que o serviço é fornecido dentro de prazos pré-estabelecidos. O comportamento temporal dos protocolos, sendo fundamental em sistemas de tempo-real, é importante na maioria dos sistemas. As propriedades de acordo descrevem quando, e a quem, uma mensagem em difusão deve ser entregue. A propriedade mais forte neste conjunto é a *unanimidade* (Px8) que indica que se uma mensagem é entregue a um participante correcto, então é entregue a todos os participantes correctos, mesmo na presença de faltas. Propriedades de acordo mais fracas, e como tal passíveis de serem concretizadas com maior desempenho, são também oferecidas (Px9 e Px10). Fi-

nalmente, as propriedades de ordem especificam quais as disciplinas de ordenação que podem ser impostas na entrega das mensagens. A propriedade mais forte, *ordem total*, assegura que as mensagens são entregues a participantes diferentes pela mesma ordem em ambos. Disciplinas de ordenação mais fracas, como causal (Px12) e FIFO (Px13) são também oferecidas. Claramente, todas estas propriedades distintas não podem ser satisfeitas por uma única primitiva de comunicação. Por este motivo, o xAMp fornece diversas primitivas, enumeradas na Tabela 4.3 (para uma descrição pormenorizada veja-se Rodrigues e Veríssimo (1992b)).

A concretização do xAMp consiste de um conjunto de módulos independentes do sistema que interagem com o sistema operativo através de uma interface designada por LSE (do Inglês, Local Support Environment (Fonseca *et al.*, 1990)). O protocolo foi utilizado sobre diversos sistemas operativos e diferentes redes locais e, naturalmente, o seu desempenho é função da arquitectura em que se executa. Na Figura 4.1 mostram-se a título de exemplo resultados obtidos para as primitivas *fiável* e *atómica* numa concretização executando-se como um gestor de periférico UNIX em máquinas SUN Sparc-Stations 1. Como se pode observar, tal como o protocolo MGS, e devido ao facto de se basear no procedimento de transmissão-com-resposta, o desempenho da qualidade de serviço atómica diminui com o número de participantes (a primitiva fiável, sendo não ordenada, possui uma menor latência mas o seu tempo de execução — não indicado na figura — também aumenta com o número de participantes).

4.5 Sincronização de relógios

Para além dos serviços anteriores, foi desenvolvido um serviço de sincronização de relógios baseado num algoritmo inovador, denominado de *acordo à posteriori* (Veríssimo & Rodrigues, 1992b). O algoritmo usa as propriedades das redes locais enumeradas na Tabela 4.1 para, sem sacrificar a exactidão, atenuar drasticamente o efeito da variação dos tempos de propagação na precisão dos relógios.

Para esclarecer como as propriedades das redes locais podem ser aproveitadas para a sincronização de relógios recorre-se a um estudo da influência das propriedades

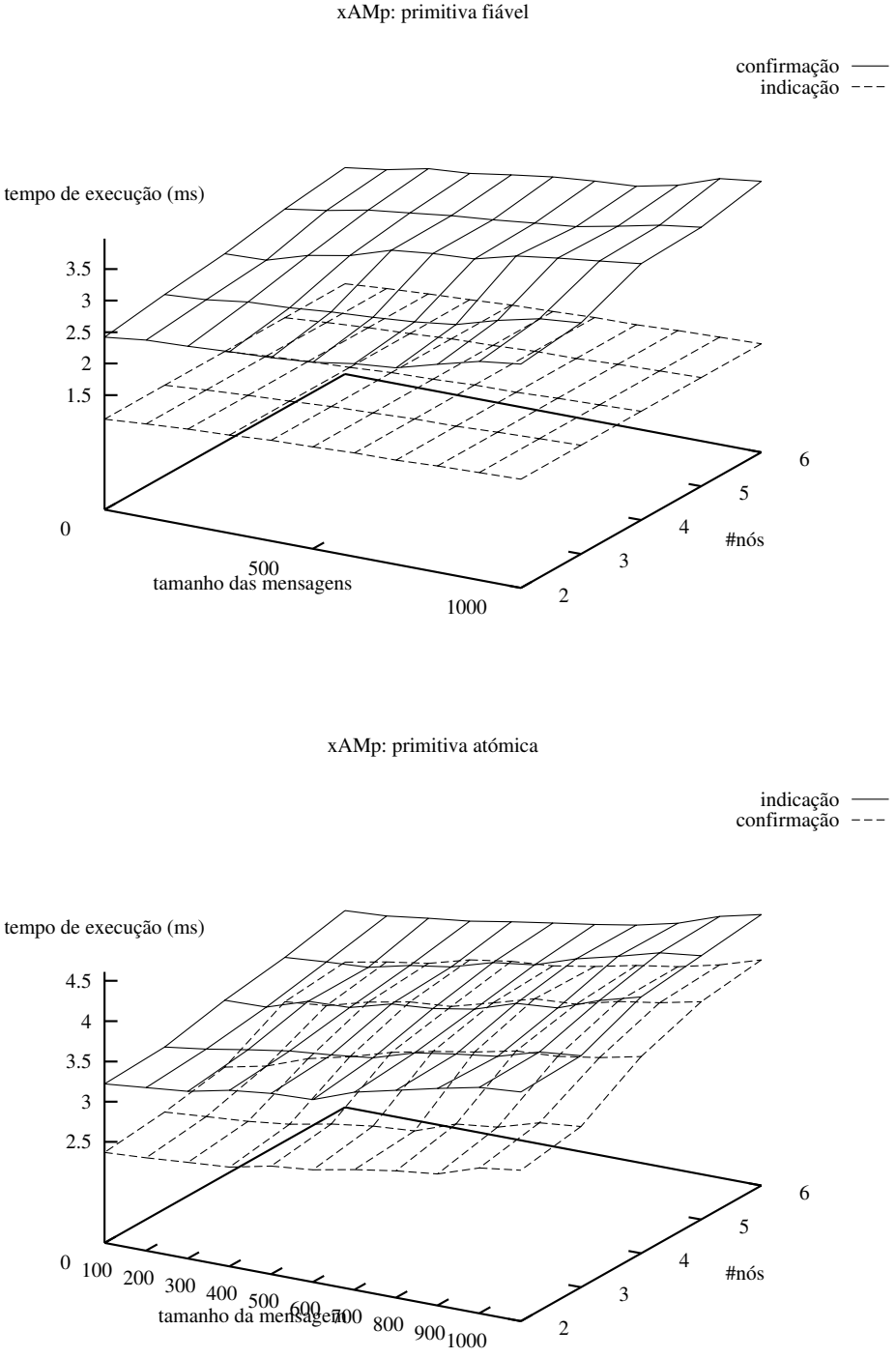


Figura 4.1: Desempenho do xAMp

temporais das redes neste tipo de protocolos (Kopetz & Schwabl, 1989). Neste estudo, decompõe-se o tempo de atraso envolvido na transmissão de uma mensagem nos seguintes termos: *tempo de transmissão*, Γ_{trans}, necessário para empacotar uma mensagem e emitir o pedido de transmissão; *tempo de acesso*, Γ_{acesso}, necessário para o emissor aceder ao canal; *tempo de propagação*, Γ_{prop}, necessário para o canal entregar a mensagem a todos os destinatários (o qual depende da localização física dos mesmos); *tempo de recepção*, Γ_{rec}, necessário para o destinatário processar a mensagem.

A precisão do algoritmo de sincronização é influenciada pela variação do tempo de transmissão, a qual é a soma da variação de cada um destes termos. O estudo mostra que, numa rede local, a variação do tempo de propagação é quase desprezável (geralmente, menor que $20\ \mu s$). O tempo de recepção é relativamente baixo, dependendo da concretização (apresentam-se valores adiante). O tempo de transmissão é quase constante. Pelo contrário, a variação do tempo de acesso, $\Delta\Gamma_{acesso}$, é significativa uma vez que é influenciada por um elevado número de factores operacionais de difícil controlo (por exemplo, colisões na Ethernet, tempo de rotação do testemunho no Token-Bus, FDDI e Token-Ring, carga na rede, etc). Deste modo, pode-se afirmar que a variação do tempo de acesso é o termo dominante na variação do tempo de transmissão e estabelecer a seguinte desigualdade (vejam-se as definições RA6 e RA7 na Tabela 4.1):

$$\Delta\Gamma_{rígido} = \Delta\Gamma_{prop} + \Delta\Gamma_{rec} \ll \Delta\Gamma$$

O algoritmo de sincronização por *acordo à posteriori* explora o facto de, numa rede local, em resposta a uma difusão, os instantes de tempo em que quaisquer dois destinatários processam a mesma mensagem ocorrerem num intervalo de amplitude menor que $\Delta\Gamma_{rígido}$. O algoritmo baseia-se no princípio de que, sendo o grau de omissão limitado, é possível num intervalo de tempo limitado gerar uma *mensagem de sincronização* que é recebida por todos os processos correctos. Esta difusão é utilizada para iniciar um novo relógio virtual que tem uma precisão de aproximadamente $\Delta\Gamma_{rígido}$. Como as mensagens de sincronização podem sofrer omissões, pode ser necessário a sua retransmissão e, consequentemente, pode ser necessário iniciar vários relógios virtuais em cada período de sincronização. Por este motivo, estes relógios designam-se por

Cenário	típico	pior-caso
$\Delta\Gamma_{rígido}$		
(A) SUN	$100\mu s$	$200\mu s$
(B) SPART	$40\mu s$	$100\mu s$
Γ_{acordo}		
SUN (x10^{-6})	$2ms(< 0.5\mu s)$	$< 100ms(< 1\mu s)$

Tabela 4.4: $\Delta\Gamma_{rígido}$ e Γ_{acordo}

relógios *candidatos*. Posteriormente, o algoritmo selecciona um dos relógios candidatos para ser utilizado no próximo período de sincronização. Durante esta segunda fase, ajusta-se também o valor absoluto do relógio seleccionado para preservar a sua exactidão. Designe-se a taxa de desvio dos relógios físicos por ρ_p e o tempo máximo necessário para seleccionar o relógio por Γ^{max}_{acordo}. Demonstra-se (Veríssimo & Rodrigues, 1992b) que a precisão δ_v obtida no final da sincronização (também designada por precisão da função de convergência concretizada por este algoritmo (Schneider, 1987)) é limitada por:

$$\delta_v \geq (1+\rho_p)\Delta\Gamma_{rígido} + 2\rho_p\Gamma^{max}_{acordo}$$

A geração de relógios candidatos requer somente a existência de suporte à difusão e o algoritmo é independente do protocolo utilizado para acordar qual o relógio a seleccionar desde que este satisfaça requisitos de tolerância a faltas e correcção temporal. Em particular, o xAMp oferece um leque de serviços que satisfazem plenamente os requisitos impostos pelo algoritmo de acordo à posteriori tendo sido utilizado para realizar um protótipo deste serviço. Uma descrição completa desta concretização, que envolve um conjunto de optimizações relativamente complexas, pode ser encontrada em Rodrigues *et al.* (1993b). Valores que permitem calcular δ_v para uma versão executando-se como um gestor de periférico UNIX em máquinas SUN Sparc-Stations 1 interligadas por Ethernet (linha A) e em cartas de comunicação dedicadas interligadas por Token-Bus (linha B) encontram-se na Tabela 4.4. Como se pode ver, o algoritmo permite obter precisões significativas sem recorrer a maquinaria especializada.

Nome	Tipo	Parâmetros
Comunicação		
l_c_melhorEsforçoN	pedido	Grupo g, PidLista dest, Inteiro N, Msg m
l_c_melhorEsforçoPara	pedido	Grupo g, PidLista dest, PidLista para, Msg m
l_c_peloMenosN	pedido	PidLista dest, Inteiro N, Msg m
l_c_peloMenosPara	pedido	Grupo g, PidLista dest, PidLista para, Msg m
l_c_fiável	pedido	Grupo g, PidLista dest, Msg m
l_c_causal	pedido	Grupo g, PidLista dest, Msg m
l_c_atómica	pedido	Grupo g, PidLista dest, Msg m
l_c_rígida	pedido	Grupo g, PidLista dest, Msg m
l_c_delta	pedido	Grupo g, PidLista dest
l_c_entrega	indicação	Grupo g, Pid remetente, Msg m
Filiação		
l_f_entra	pedido	Grupo g
l_f_sai	pedido	Grupo g
l_f_vista	indicação	Grupo g, PidLista vista
Sincronização de relógios		
l_sr_começa	pedido	
l_sr_acaba	pedido	
l_sr_tempo	indicação (a pedido)	UnidadeDeTempo valor

Tabela 4.5: Interface do NAVTECH para rede local

4.6 Interface NAVTECH para rede local

Na Tabela 4.5 resume-se a interface NAVTECH para rede local. Esta é apresentada sem mais comentários, uma vez que se pretende apenas agregar a interface das diversas primitivas referidas anteriormente numa única tabela, de modo a facilitar a sua referência.

4.7 Acerca da correcção dos protocolos e sua concretização

Um aspecto crucial no desenvolvimento e concretização de protocolos de comunicação (como aliás em muitas outras áreas) é a validação da sua correcção. Existem diversas técnicas que permitem aumentar a confiança na correcção de uma concretização, incluindo a simulação, teste, injecção de faltas, inspecção de código, desenvolvimento de provas informais, e métodos formais de especificação e verificação (Ghezzi *et al.*, 1991). Estes últimos revelam-se particularmente úteis, pois

para além de permitirem detectar erros nos protocolos, dão indicações precisas acerca da sua localização (Baptista, 1991).

Uma parte substancial do protocolo que deu origem ao xAMp (o AMp) foi especificado formalmente usando a linguagem ESTELLE e posteriormente verificado usando uma ferramenta dedicada (Baptista *et al.*, 1991). Como alguns dos mecanismos do AMp foram integrados no xAMp, esta validação aumenta a confiança nos protocolos utilizados apesar de, naturalmente, ser impossível garantir que as diversas optimizações entretanto efectuadas não comprometeram a sua correcção. Assim, e mesmo considerando que o xAMp foi extensivamente testado, nomeadamente no sistema Delta-4, julga-se importante realizar no futuro um novo esforço de verificação formal. O mesmo seria interessante fazer em relação ao protocolo de sincronização de relógios, para o qual foi desenvolvida uma prova informal (Rodrigues & Veríssimo, 1992a). No entanto, note-se que o investimento na especificação e verificação formal de protocolos complexos, para os quais ainda não é possível gerar automaticamente código executável a partir da especificação, é sempre comprometido por potenciais erros introduzidos durante a codificação na linguagem de programação alvo (Baptista, 1991) (o que já não acontece para protocolos simples, como os utilizados em sistemas críticos (Melliar-Smith & Schwartz, 1982)).

4.8 Discussão

Os componentes da arquitectura NAVTECH descritos neste capítulo foram obtidos através do aperfeiçoamento de serviços desenvolvidos para a arquitectura Delta-4. Os restantes componentes da arquitectura NAVTECH surgiram da necessidade de suportar o mesmo tipo de serviços, ou semelhantes, sobre redes de grande escala. Podem-se enumerar as principais razões que impedem a utilização destes protocolos sobre este tipo de redes:

- Muitos dos protocolos baseiam-se na propriedade de ordenação da rede abstracta (RA5) para estabelecer ordens totais entre acontecimentos (serialização de mensagens, exclusão mútua, etc). As redes de grande escala não possuem esta

propriedade (aliás, assim como as redes locais interligadas) pelo que outro mecanismo de serialização necessita de ser desenvolvido.

- Muitos dos protocolos baseiam-se num mecanismo de transmissão com resposta. Este mecanismo é intrinsecamente não escalável em relação ao número de participantes. Mesmo para um reduzido número de participantes, a latência de comunicação em grande escala penaliza este tipo de aproximação.

Naturalmente, a experiência adquirida com o desenvolvimento destes protocolos influenciou profundamente a concepção dos restantes componentes da arquitectura NAVTECH. Listam-se de seguida os aspectos mais importantes.

- O desenvolvimento do protocolo MGS reforçou a convicção de que um sistema de gestão da filiação em dois níveis é uma solução adequada para sistemas em que diversos processos partilham o mesmo nó.
- Os resultados obtidos demonstraram que uma cuidada modelização das redes alvo, identificando as propriedades relevantes para a construção de protocolos, permite obter mais valias sem comprometer demasiado a portabilidade. Daí a importância que se deu no NAVTECH à modelização da rede de grande escala.
- Alguns dos resultados obtidos são de tal modo favoráveis (por exemplo, a precisão obtida na sincronização de relógios) que aconselham a sua utilização, mesmo que restrita às redes locais. Isto reforçou a convicção de que interessava preservar um conjunto de protocolos optimizados para este tipo de redes e combiná-los, de forma hierárquica, com protocolos mais adequados à grande escala.
- O suporte de um elevado número de qualidades de serviço mostrou vantagens e desvantagens. Por um lado, surgiu da necessidade concreta, expressa por diversos utilizadores, de emparelhar o serviço fornecido com os requisitos da aplicação. Por outro lado, tornou mais difícil a utilização e manutenção dos componentes uma vez que existem interdependências não triviais entre os diversos serviços. Por este motivo, optou-se na arquitectura NAVTECH por suportar um conjunto de *perfis*, aos quais se associam um conjunto de serviços compatíveis.

- O recurso à especificação e verificação formal revelou-se extremamente útil para aumentar a confiança na correcção dos protocolos. Infelizmente, nem sempre é possível verificar um protocolo complexo na sua totalidade e, adicionalmente, nem sempre é possível criar automaticamente um executável eficiente a partir de uma especificação formal. Assim, ao invés de estar completamente integrada no ciclo de desenvolvimento, a tarefa de especificação e verificação formal acaba por se tornar uma tarefa quase independente, e onerosa em termos de recursos (Baptista, 1991). Deste modo, e para não invalidar o esforço de validação com posteriores alterações ao código, sugere-se uma aproximação pragmática que consiste em recorrer a estas técnicas apenas quando já se têm versões estáveis dos protocolos (após obter significativa experiência através de simulações e testes).

4.9 Sumário

Neste capítulo apresentaram-se, de um modo abreviado e informal, os componentes da arquitectura NAVTECH desenvolvidos para operarem sobre rede local. Historicamente, estes componentes foram desenvolvidos através do aperfeiçoamento de serviços cuja génese é anterior à concepção desta arquitectura. A experiência obtida no seu desenvolvimento inspirou a concepção da arquitectura NAVTECH e a análise das suas limitações motivou o desenvolvimento dos componentes orientados às redes de grande escala, que serão o tema do próximo capítulo.

Notas

O trabalho desenvolvido na área da comunicação em grupo orientada para rede local foi realizado em colaboração com os Engs. Mário Baptista, António Casimiro, Henrique Fonseca, José Rufino, Luís Silva e Werner Vogels. Parte deste trabalho foi baseado na depuração e extensão de resultados apresentados na Tese de Mestrado do autor, "Mecanismos de comunicação eficientes para sistemas de tempo-real e tolerantes a faltas", Instituto Superior Técnico, 1990. O protocolo de filiação de nós foi apresentado em "A low-level processor group membership protocol for LANS",

L. Rodrigues, P. Veríssimo, e J. Rufino, actas da 13ª "IEEE International Conference on Distributed Computing Systems", Pittsburgh, Pennsylvania, USA, Maio 1993. O xAMp foi apresentado em "xAMp: a Multi-primitive Group Communications Service", L. Rodrigues e P. Veríssimo, actas do 11º "IEEE Symposium on Reliable Distributed Systems", Houston, Texas, Outubro 1992. A qualidade de serviço rígida *deste pacote foi descrita em "Priority-based totally ordered multicast", L. Rodrigues, A. Casimiro, e P. Veríssimo, actas do 3º "IFAC/IFIP workshop on Algorithms and Architectures for Real-Time Control (AARTC'95)", Maio 1995. O protocolo de sincronização de relógios foi apresentado em "A posteriori Agreement for Fault-tolerant Clock Synchronization on Broadcast Networks", L. Rodrigues, e P. Veríssimo, actas do 22º "IEEE International Symposium on Fault-Tolerant Computing", Boston, USA, Julho 1992. Uma concretização deste protocolo sobre o xAMp foi apresentada em "Using atomic broadcast to implement* a posteriori *agreement for clock synchronization", L. Rodrigues, P. Veríssimo, e A. Casimiro, actas do 12º "IEEE Symposium on Reliable Distributed Systems", Princeton, New Jersey, USA, Outubro 1993.*

5

Comunicação em grande escala

5.1 Enquadramento

Este capítulo apresenta os módulos da arquitectura NAVTECH, concebidos para operarem sobre grande escala, que foram desenvolvidos no âmbito desta tese. Nomeadamente, apresenta-se um protocolo de ordenação causal (assim como duas variantes deste serviço, designadas por *mensagens transparentes* e por *mensagens retidas*) e um protocolo de ordenação total. Módulos relacionados desenvolvidos — ou em desenvolvimento — por outros elementos do grupo são também referidos.

5.2 Particularidades das redes de grande escala

Na capítulo anterior, descreveram-se os protocolos da arquitectura NAVTECH orientados à rede local. Resumiram-se também as suas limitações para aplicação em grande escala. Um aspecto inovador dos protocolos aqui referidos, consiste no facto destes terem em consideração as seguintes particularidades das redes de grande escala:

- *Estrutura hierárquica*: este aspecto já foi referido no Capítulo 3. Esta característica será utilizada no protocolo de ordenação causal.
- *Susceptibilidade a partições*: este aspecto deve ser tido em consideração na concepção do algoritmo de filiação (não abordado nesta tese). Para além disso, e sempre que possível, deve-se recorrer a algoritmos que não exijam informação

de filiação permanentemente actualizada (um exemplo será descrito no próximo capítulo).

- *Baixa fiabilidade*: sempre que possível, deve-se recorrer a qualidades de serviço que tolerem eventuais perdas de mensagens. O serviço de *mensagens transparentes* tem este aspecto em consideração.

- *Latência elevada e com grande variação*: este aspecto é explicitamente considerado no protocolo de ordenação total.

- *Custo não desprezável no envio de mensagens*: aconselhando o uso de propagação diferida de actualizações, com aglomeração de várias mensagens lógicas numa única trama. Este aspecto é tido em consideração pelo serviço de mensagens retidas.

5.3 Rede abstracta e detecção de falhas em grande escala

Tal como para a rede local, um modo eficaz de garantir a portabilidade do código e de optimizar o seu desempenho consiste em definir uma *rede abstracta* que torne visíveis os atributos tecnológicos relevantes para o desenvolvimento de aplicações. A rede abstracta, para além de oferecer uma interface genérica para a infra-estrutura física de comunicação, deve concretizar os serviços elementares que são directamente dependentes desta infra-estrutura. Nomeadamente, a rede abstracta deve oferecer serviços de difusão melhor-esforço. A concretização do serviço de detecção de falhas (que poderá ser parametrizado pela aplicação (Cosquer *et al.*, 1995b)) é naturalmente, extremamente dependente das propriedades da rede abstracta. A rede abstracta (e o serviço de detecção de falhas) para grande escala da arquitectura NAVTECH encontra-se a ser desenvolvida no âmbito de uma tese de mestrado (Frazão, 1995). Assume-se que a rede abstracta fornece, pelo menos, um serviço de difusão do tipo melhor esforço, com a interface indicada na Tabela 5.1.

Nome	Tipo	Parâmetros
g_ra_mesforço	pedido	Grupo g, IdProcLista destinatários, Msg m
g_ra_entrega	pedido	Grupo g, IdProc remetente, Msg m

Tabela 5.1: Interface mínima da rede abstracta em grande escala

5.4 Filiação em grande escala

Um algoritmo de filiação em grande escala difere em diversos aspectos de um algoritmo de filiação orientado à rede local. Em primeiro lugar, a susceptibilidade a partições e a inerente assincronia deste tipo de redes aconselham o uso de detectores de falha bem afinados (de modo a optimizarem a precisão sem comprometerem a animação). É possível optar por um serviço de filiação linear desde que só se permita o progresso na partição maioritária, ou oferecer vistas parcialmente ordenadas (mais aconselhável, mas também mais complexo). Em segundo lugar, a latência na comunicação aconselha que se tentem utilizar protocolos com baixo número de turnos de "envio-resposta" e que se tente executar os protocolos de filiação em paralelo com a comunicação. Dados os custos inerentes à satisfação da sincronia virtual (Babaoğlu *et al.*, 1995), definições enfraquecidas têm sido procuradas (Dolev *et al.*, 1993a; Friedman & Renesse, 1995).

À data da escrita da dissertação, o 1º protótipo NAVTECH oferecia um serviço de filiação *linear* (isto é, só entregando vistas na partição maioritária) oferecendo *sincronia virtual forte* (veja-se a secção 2.3.1.2). Embora este seja o tipo de serviço assumido nesta dissertação, um serviço oferecendo vistas parcialmente ordenadas está ser desenvolvido no âmbito de uma tese de mestrado (Sargento, 1995). Assume-se que o serviço de filiação fornece, pelo menos, primitivas de filiação, indicação de alteração na filiação, e duas primitivas de comunicação, uma de difusão virtualmente síncrona e outra de difusão uniforme virtualmente síncrona, tal como indicado na Tabela 5.2 (tal como para a rede local, pressupõe-se a disponibilidade de endereçamento selectivo, isto é, a possibilidade de endereçar apenas um subconjunto dos membros do grupo).

Nome	Tipo	Parâmetros
Filiação		
g_f_entra	pedido	Grupo g
g_f_sai	pedido	Grupo g
g_f_vista	indicação	Grupo g, IdProcLista vista
Sincronia virtual		
g_sv_difusão	pedido	Grupo g, IdProcLista destinatários, Msg m
g_sv_uniforme	pedido	Grupo g, IdProcLista destinatários, Msg m
g_sv_entrega	indicação	Grupo g, IdProc remetente, Msg m

Tabela 5.2: Interface mínima do serviço de filiação de grande escala

5.5 Grupos de baixo-custo

A concretização de um serviço de filiação requer a reserva de diversos recursos (memória e tempo de processamento) para cada grupo que é mantido no sistema. Exemplos destes recursos são as estruturas e actividades associadas ao detector de falhas, listas de mensagens pendentes, descritores de grupo, entre outros. Para além disso, sempre que um elemento se filia ou abandona o grupo, é necessário executar um protocolo que, como se referiu anteriormente, pode requerer vários turnos de troca de mensagens.

Existem diversos estilos de programação, tal como a orientação aos objectos, que encorajam o programador a desenvolver os seus programas de modo modular. Usando estes estilos, uma aplicação é composta por um vasto número de componentes que são executados simultaneamente (e frequentemente, dentro do mesmo espaço de endereçamento). Quando estes componentes usam o serviço de filiação (abrindo um grupo dedicado), uma única aplicação pode utilizar um vasto número de grupos que possuem aproximadamente (quando não exactamente) a mesma filiação. Este comportamento tem sido observado em diversas aplicações (Powell, 1991; Hagsand *et al.*, 1992; Glade *et al.*, 1993) incluindo a experiência com o Romance (Rodrigues & Veríssimo, 1993b).

Estas duas observações aconselham o desenvolvimento de um módulo capaz de projectar diversos grupos ao nível da aplicação num único grupo ao nível do serviço de filiação. Este módulo, designado por serviço de *grupos de baixo custo*, permite

a partilha de recursos e limita o processamento redundante, sempre que uma falha obriga a alterações equivalentes na filiação dos diversos grupos ao nível da aplicação. Podem-se citar duas concretizações deste serviço:

- No sistema Delta-4 (Powell, 1991) existe um módulo que projecta diversos grupos ao nível sessão em diversos grupos ao nível MAC (geridos pelo xAMp). Esta projecção é definida estaticamente com base numa etiqueta atribuída a cada grupo em tempo de configuração.
- Um módulo semelhante foi desenvolvido para o sistema ISIS, designado ISIS-LWG (Glade *et al.*, 1993) (do Inglês, Light Weight Group), oferecendo uma atribuição dinâmica entre os grupos ao nível da aplicação e os grupos ISIS. Infelizmente, esta camada obriga a que o utilizador indique, no momento da criação, qual a filiação esperada para o grupo.

Nenhuma destas aproximações é completamente transparente, no sentido em que os grupos ao nível da aplicação não preservam exactamente a mesma semântica dos grupos oferecidos pelo protocolo de filiação. No caso do Delta-4 é necessário etiquetar o grupo antes de o criar, e no ISIS-LWG indicar qual a sua filiação. Para ultrapassar estas limitações, o autor sugeriu uma técnica para oferecer grupos de baixo custo que preserva a interface oferecida pelo serviço de filiação, fazendo a projecção de modo automático e transparente (Rodrigues, 1994). Esta ideia está a ser desenvolvida no âmbito do projecto GODC, em colaboração com a Universidade de Cornell.

5.6 Ordem causal

Nesta secção apresenta-se uma técnica que usa o conhecimento acerca da topologia do sub-sistema de comunicação para reduzir a quantidade da informação que necessita de ser trocada para preservar a ordenação causal. Esta técnica, que melhora resultados anteriores, baseia-se no conceito de *separador causal*, um conjunto de processos que pode ser usado para filtrar informação (Rodrigues & Veríssimo, 1995b). Apresenta-se uma concretização desta optimização.

Adicionalmente, apresenta-se uma metodologia para modelar os sistemas de comunicação como um grafo, no qual os agrupamentos físicos e administrativos se podem projectar em separadores causais. Esta metodologia deriva do modelo de rede usado na arquitectura NAVTECH e apresentado no Capítulo 3. Relembra-se que de acordo com este modelo, a rede é constituída por aglomerados de nós interligados por um conjunto bem definido de agentes. Na secção 5.6.7 mostra-se como a noção de separador causal se pode aplicar a estes conjuntos de agentes.

5.6.1 Motivação

A experiência tem demonstrado (Schneider, 1990; Birman & Joseph, 1987; Peterson *et al.*, 1989; Renesse *et al.*, 1992) que o desenvolvimento de aplicações distribuídas pode ser simplificado através do uso de protocolos de comunicação que preservem as relações de causalidade entre as mensagens. Infelizmente, de um modo geral este serviço requer que cada processo mantenha — e troque com os restantes processos — informação acerca das mensagens enviadas "no passado". A quantidade desta informação aumenta com o número de processos envolvidos na comunicação e com a diversidade de endereços utilizados. Como se irá referir de seguida, caso não se recorra a informação adicional, é necessário guardar — e trocar — informação respeitante a pelo menos n^2 mensagens (em que n é o número de processos do sistema). Em sistemas com elevado número de nós este custo pode tornar-se proibitivo. Por esta razão é interessante desenvolver protocolos que minimizem a informação que é necessário manter. Uma das técnicas possíveis para atingir este fim consiste em explorar conhecimento acerca da topologia da rede.

5.6.2 Trabalho relacionado

As primeiras concretizações de serviços de comunicação respeitando a precedência lógica (também designada por relação "aconteceu-primeiro") foram baseadas em *relógios lógicos* (Lamport, 1978), uma técnica que introduz um atraso sistemático na entrega de mensagens e que ordena mais mensagens do que as estritamente ne-

cessárias (Schwarz & Mattern, 1991). Para evitar esta desvantagem, foram propostos algoritmos baseados em *histórias causais* ou *relógios vectoriais* (Birman & Joseph, 1987; Peterson *et al.*, 1989; Ladin *et al.*, 1990). Enquanto algumas destas aproximações não consideravam a possibilidade de grupos se interseptarem (Peterson *et al.*, 1989), a maioria das soluções recentes permite que diferentes mensagens sejam enviadas para diferentes grupos de processos (Birman & Joseph, 1987; Ladin *et al.*, 1990; Mostefaoui & Raynal, 1993). Infelizmente, o endereçamento em grupo aumenta o volume da informação que necessita de ser mantida para preservar a causalidade.

Um exemplo pioneiro da utilização de histórias causais para ordenar mensagens em difusão consistiu na concretização da primitiva CBCAST do sistema ISIS (Birman & Joseph, 1987). Nesta concretização, as histórias causais incluem uma cópia de todas as mensagens no passado e, sempre que uma mensagem é enviada, uma cópia da história causal do emissor é agregada à mensagem. A vantagem desta técnica é a garantia que a entrega das mensagens nunca necessita de ser diferida (dado que a própria trama transporta as mensagens no "passado" que se tenham eventualmente perdido). O algoritmo usado no PSYNC (Peterson *et al.*, 1989) é semelhante ao CBCAST mas requer apenas a troca de identificadores de mensagens. Os relógios vectoriais não são mais que uma representação das histórias causais, em que o identificador da última mensagem conhecida de cada processo é guardado numa posição pré-definida do vector. No entanto, mesmo que se use apenas um grupo, e que se guardem apenas os identificadores das mensagens, o tamanho dos vectores cresce linearmente com o número de processos (Charron-Bost, 1991). Caso se usem vários grupos que se interseptem, o volume da informação cresce quadraticamente com o número de processos (Raynal *et al.*, 1991). Por exemplo, o protocolo descrito por Stephenson (1991) requer a manutenção de um relógio vectorial por cada grupo.

Existem soluções genéricas para diminuir o volume da informação utilizada, mas que implicam o diminuição do paralelismo no sistema (Stephenson, 1991; Mostefaoui & Raynal, 1993). Por exemplo, é possível usar um único inteiro para ordenar todas as mensagens desde que se estabeleça uma ordem total no sistema (naturalmente, perdendo toda a concorrência e com um pesado custo em termos de latência). Uma solução possível para diminuir o volume da informação sem diminuir a concorrência do sistema

consiste em explorar informação acerca da topologia da comunicação. Demonstrou-se que se o grafo de comunicação contém processos que representam uma "ponte" entre dois sub-grafos, é possível diminuir o volume de informação que necessita de ser mantida por cada processo (Meldal *et al.*, 1991). Foram também sugeridas optimizações baseadas em padrões de comunicação (Stephenson, 1991).

5.6.3 Separadores causais

A técnica aqui descrita estende alguns dos resultados referidos anteriormente, aplicando o mesmo tipo de optimizações a estruturas de comunicação arbitrárias. Nomeadamente:

- mostra-se que, mesmo quando o grafo contém ciclos, é possível reduzir a informação trocada definindo *separadores causais*, um conjunto de nós que pode ser usado para filtrar informação de causalidade.
- apresenta-se uma metodologia para modelar o sistema de comunicação que permite aplicar o resultado anterior às redes de grande escala.

Estes dois aspectos serão pormenorizados nas sub-secções seguintes. Apresenta-se primeiro uma panorâmica de conjunto.

Parte-se do princípio que cada processo p só consegue comunicar directamente com um subconjunto dos nós do sistema, $\mathcal{D}_p \subseteq \mathcal{P}$. A topologia de comunicação pode assim ser representada por um grafo $G(\mathcal{P}, A)$, no qual os processos são representados pelos vértices e os elos de comunicação pelos arcos: existe um arco incidente a $\{p_1, p_2\}$ quando p_1 pode enviar mensagens directamente para p_2. Assume-se que o grafo é ligado. Um conjunto de processos S é denominado um (S, S^a, S^p) *separador de vértices*, onde os conjuntos S^a e S^p são designados, respectivamente, *conjunto anterior* e *conjunto posterior* ao separador, sse S, S^p e S^a são disjuntos dois a dois e todos os caminhos ligando $p \in S^p$ a $a \in S^a$ passam pelo menos por um membro $s \in S$. No contexto da comunicação causal, chamam-se a estes separadores de vértices, *separadores causais* (veja-se a Figura. 5.1).

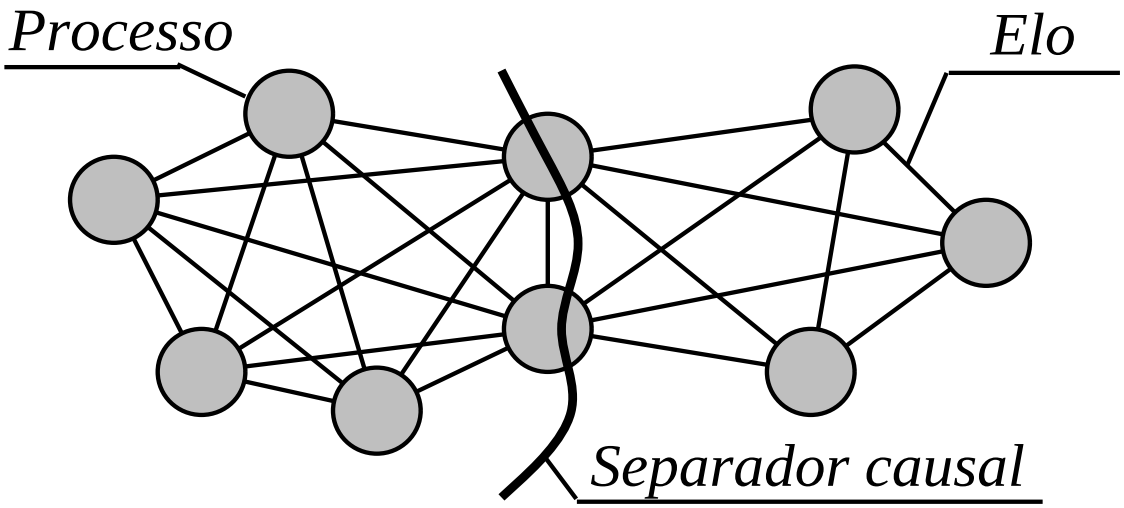


Figura 5.1: Separadores causais.

Na sub-secção 5.6.5 mostra-se que um separador causal pode funcionar como uma barreira que filtra toda a informação respeitante a mensagens destinadas exclusivamente a elementos do *conjunto posterior* ao separador e "relatadas" aos restantes elementos do separador (define-se a relação "relatadas" com precisão mais adiante). Designa-se um conjunto de processos limitado por um ou mais separadores causais *uma zona causal*. Uma consequência interessante da observação anterior é que a informação respeitante a mensagens enviadas para elementos de uma zona causal pode não necessitar de ser propagada para além das fronteiras definidas pelo separador causal, desde que seja "relatada" aos membros do separador.

Na sub-secção 5.6.7 mostra-se que é possível modelar o sistema de comunicação de maneira a que a noção de separadores causais possa ser usada na prática. A técnica explora as propriedades das redes existentes, em particular, a característica hierárquica destas, para criar um grafo de comunicação no qual os separadores causais se projectam nas fronteiras dos agrupamentos físicos e administrativos.

5.6.4 Histórias causais estendidas

Apresenta-se agora uma concretização de histórias causais que permite utilizar a noção de separadores causais para limitar a quantidade de informação que necessita de ser trocada no sistema.

5.6.4.1 Definições

Ao longo do texto que se segue assume-se que o sistema é composto por um conjunto de processos, $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ com espaços de endereçamento distintos e modos de falha não relacionados. Assume-se que cada processo possui um identificador único $p \in \mathcal{P}$ (por conveniência, usa-se a mesma notação para denotar tanto o próprio processo como o seu identificador). Assume-se igualmente que existe uma relação de ordem total $\prec$ entre os identificadores. Os processos comunicam por troca de mensagens: a identificação do emissor $s_m \in \mathcal{P}$ e o conjunto de destinatários $\mathcal{A}_m \subseteq \mathcal{P}$ são associados a cada mensagem m. Não se coloca nenhuma restrição no conjunto de destinatários: uma mensagem pode ser enviada para qualquer sub-conjunto de processos em $\mathcal{P}$. Assume-se também que cada processo atribui um valor inteiro c_m, gerado localmente, a cada mensagem, de modo a que se m_1 fôr enviada antes de m_2 então têm-se $c_{m_1} < c_{m_2}$ (isto pode ser facilmente obtido recorrendo a um contador[1] c_p em cada processo p). Apesar do par (s_m, c_m) identificar unicamente uma mensagem, por conveniência define-se o *identificador único de mensagem* de modo a conter também o conjunto de destinatários, isto é, $idu_m \equiv (s_m, c_m, \mathcal{A}_m)$. Note-se que num sistema usando exclusivamente difusão não selectiva, o campo $\mathcal{A}_m$ é constante ($\forall_m\ \mathcal{A}_m = \mathcal{P}$) e pode ser omitido.

Tal como na grande maioria dos trabalhos na área da difusão fiável, para além do evento *envio*, são definidos dois eventos adicionais: *recepção* e *entrega*. O evento de recepção identifica a chegada da mensagem a um determinado processo e não é visível ao nível da aplicação. O evento de entrega identifica a chegada da mensagem à aplicação. De modo a garantir uma ordem de entrega de acordo com a causalidade o protocolo pode ter de atrasar a entrega de mensagens já recebidas. Parte-se do princípio que o protocolo é executado sobre uma camada de difusão fiável como a referida na secção 5.4. Não são feitas hipóteses acerca da ordem pela qual as mensagens são recebidas. Devido às características de fiabilidade assumidas para o nível inferior, as histórias causais *não necessitam* de incluir as próprias mensagens mas *apenas* os seus

[1] Uma vez que este contador deve ser armazenado num vector de dígitos binários com capacidade limitada, não é possível na prática ter contadores sempre crescentes. No entanto, existem técnicas que permitem superar esta limitação. (Lloyd & Kearns, 1990).

identificadores únicos. No entanto, para facilitar a descrição, sempre que o conteúdo das histórias causais é referido usa-se simplesmente a palavra "mensagem" em vez da expressão mais precisa, mas também mais longa, "identificador único da mensagem".

5.6.4.2 Usando "químicos"

Introduz-se agora a nossa representação de histórias causais. O leitor interessado notará que não existem diferenças radicais entre a representação que aqui se utiliza e representações alternativas que se podem encontrar na bibliografia (Birman & Joseph, 1987; Peterson *et al.*, 1989). No entanto, esta representação facilita de sobremaneira a concretização de optimizações baseadas no conceito de separadores causais. Esta representação das histórias causais, que foi denominada de *história causal estendida*, armazena a informação sobre a causalidade em três entidades diferentes: uma *história causal* $\mathcal{C}_p$, uma lista de mensagens que precede a próxima mensagem a ser enviada por p; a *história de entrega* $\mathcal{E}_p$; e uma *história papel-químico* $\mathcal{Q}_p$ que memoriza para onde a informação causal já foi "relatada" (a história papel-químico é usada para reduzir a informação trocada e o seu uso será esclarecido adiante).

A história de entrega é local a um processo, e mantém um registo de todas as mensagens que já foram entregues a esse processo. A história causal mantém um registo de todas as mensagens que precedem a próxima mensagem a enviar por um determinado processo, e é construída por cada processo, com informação relativa às mensagens que são enviadas e entregues localmente. Apesar da história causal conter a história de entrega, diferentes regras para eliminar informação redundante serão aplicadas a cada história, daí que se tenha decidido manter esta informação explicitamente em duas histórias separadas (uma separação explicita entre a história causal e a história de entrega é também utilizada noutras aproximações (Raynal *et al.*, 1991)). Estas histórias são usadas do seguinte modo:

Cada vez que uma mensagem m é enviada pelo processo p, é estampilhada com a história causal $\mathcal{C}_p$ do emissor. Todas as mensagens em $\mathcal{C}_p$ dizem-se então "relatadas" aos destinatários de m. O conjunto de processos a que cada mensagem foi relatada é mantido na história papel-químico $\mathcal{Q}_p$ de p. Isto é, $\mathcal{Q}_p$ mantém para cada mensagem

$m \in \mathcal{C}_p$ o conjunto $\mathcal{Q}_p(m)$ de processos aos quais m foi relatada.

Quando uma mensagem é recebida por um processo q, o destinatário compara a estampilha da mensagem com a sua história de entrega e verifica se todas as mensagens precedentes (e endereçadas a q) já foram localmente entregues: a entrega é feita de imediato ou atrasada até que esta condição se verifique. Quando uma mensagem é entregue, o destinatário actualiza a sua história de entrega (acrescentando-lhe o identificador da mensagem entregue) e a sua história causal (integrando-lhe o identificador *e a estampilha* da mensagem entregue)[2].

Mais precisamente, a entrega causal é garantida através da utilização da história causal e da história de entrega de acordo com a seguintes regras (para maior clareza, difere-se a utilização da história papel-químico até que a regra 6 seja introduzida):

R1 (Estado inicial): *Quando p inicia a sua execução, $\mathcal{C}_p$, $\mathcal{E}_p$, e $\mathcal{Q}_p$ encontram-se vazias. De igual modo, $c_p = 0$.* □

R2 (Estampilhagem): *Antes de ser enviada pelo processo p, a mensagem m recebe um novo identificador idu, obtido através do incremento do contador local c_p. Seguidamente, m é estampilhada com a história causal de p, ou seja, $\mathcal{C}_m = \mathcal{C}_p$.* □

R3 (Entrega causal): *Na recepção de uma mensagem m enviada pelo processo p e estampilhada com a história causal $\mathcal{C}_m$, o processo $q \in \mathcal{A}_m$ atrasa a entrega de m até que todas as mensagens em $\mathcal{C}_m$ também endereçadas a q tenham sido localmente entregues. Precisamente, q atrasa a entrega de m até que a condição seguinte se verifique:*

$$\forall (i \in \mathcal{C}_m : q \in \mathcal{A}_i) \;\; i \;\; \textit{está-em} \;\; \mathcal{E}_q$$

em que a relação "está-em" se define como: m está-em $\mathcal{E} \iff m \in \mathcal{E}$. *De acordo com esta regra, uma mensagem pode sempre ser entregue de imediato ao seu próprio emissor.* □

R4 (Registo): *Quando a mensagem m é entregue a q, a estampilha $\mathcal{C}_m$ de m é integrada*

[2] Desde modo, um processo pode manter na sua história causal informação respeitante a mensagens que não lhe são endereçadas.

em $\mathcal{C}_q$. Para além disso, m é adicionada a $\mathcal{C}_q$ e a $\mathcal{E}_q$. □

As regras 1-4 acima referidas permitem garantir a entrega causal das mensagens. No entanto, esta solução tem o inconveniente de permitir o crescimento indeterminado das histórias causais e de entrega. Nos próximos parágrafos apresenta-se um conjunto adicional de regras que permitem eliminar elementos redundantes destas histórias.

Começa-se por reduzir o tamanho da história de entrega. A regra usa o facto das mensagens originárias de um mesmo processo serem sempre entregues pela ordem em que foram enviadas. De acordo com a regra R3, se uma mensagem m enviada pelo processo p é entregue a q, então todas as mensagens anteriores vindas de p, e endereçadas a q, foram já entregues a q. Como resultado desta regra, as histórias de entrega não necessitam de manter mais que uma única mensagem por cada processo. Ou seja,

R5 **(Entrega FIFO):** *No máximo uma única mensagem por cada emissor necessita de ser armazenada na história de entrega. Sempre que uma mensagem m enviada por p é adicionada a $\mathcal{E}_q$, m substitui a mensagem anterior enviada de p e entregue a q.*

Uma vez que agora alguns elementos da história de entrega são removidos à medida que novos elementos são adicionados, a definição "está-em $\mathcal{E}$" necessita de ser ligeiramente alterada. Diz-se que uma mensagem m está-em $\mathcal{E}$ se e apenas se existe uma mensagem em $\mathcal{E}$, enviada pelo mesmo processo, com um identificador igual ou maior. Precisamente, m está-em $\mathcal{E} \iff \exists_{n \in \mathcal{E}} : (s_m = s_n \wedge c_m \leq c_n)$. □

Elimina-se agora informação redundante da história causal. A ideia é remover todos aqueles elementos não estritamente necessários para garantir a entrega de acordo com a causalidade. Essencialmente, elimina-se informação acerca do passado. O método pode ser visto como uma extensão da utilização de vectores tipo "último-enviado" e "último actualizado" sugerida por Singhal e Kshemkalyani (1990), e também por Stephenson (1991) para optimizar o tamanho dos relógios vectoriais em grupos não disjuntos. Este método será estendido para esquemas de endereçamento arbitrário. A optimização pode ser apresentada informalmente do seguinte modo: antes de ser enviada, uma mensagem m é estampilhada com a história causal do seu emissor $\mathcal{C}_m = \mathcal{C}_p$. Esta mensagem só será entregue a um determinado elemento de $\mathcal{A}_m$, após todas as

mensagens incluídas em $\mathcal{C}_m$, e endereçadas a esse processo, terem sido entregues. Deste modo, toda a mensagem $n : \mathcal{A}_n \subseteq \mathcal{A}_m$ que seja estampilhada com m, não necessita de ser estampilhada com as mensagens já incluídas na estampilha de $\mathcal{C}_m$, uma vez que esta será necessariamente entregue depois de m, logo, depois de todas as mensagens em $\mathcal{C}_m$. Aplicar esta optimização requer que seja mantida informação acerca de a quem cada mensagem foi "relatada", isto é, quais os destinatários de estampilhas incluindo a mensagem. Esta informação pode ser mantida numa história adicional, denominada *história papel-químico* $\mathcal{Q}$. A história papel-químico contém um elemento para cada elemento da história causal, no qual é armazenada a lista de processos aos quais o correspondente elemento da história causal foi "relatado". A história papel-químico deve ser actualizada de acordo com a seguintes regras:

R6 (História papel-químico): *Cada processo p mantém uma história papel-químico $\mathcal{Q}_p$ que contém um elemento $\mathcal{Q}_p(m)$ associado a cada mensagem $m \in \mathcal{C}_p$. Estes elementos são usados de acordo com as regras R6.1, R6.2 e R6.3.* □

R6.1- Estampilhagem estendida (opcional): *Quando a mensagem m é enviada, para além de ser estampilhada com $\mathcal{C}_p$, pode também ser estampilhada com $\mathcal{Q}_p$. Designa-se o campo da estampilha de m referente à história papel-químico por $\mathcal{Q}_m$.* □

Esta regra tem por objectivo permitir uma mais rápida actualização do conteúdo das histórias papel-químico dos diversos processos.

R6.2- Actualização no envio: *Após enviar a mensagem m, e antes de inserir m em $\mathcal{C}_p$, todos os campos de $\mathcal{Q}_p$ são actualizados, memorizando que as mensagens correspondentes já foram "relatadas" a $\mathcal{A}_m$, ou seja:*

$$\texttt{para-todos} \quad (i \in \mathcal{Q}_p) \ \texttt{execute-se} \quad \mathcal{Q}_p(i) := \mathcal{Q}_p(i) \cup \mathcal{A}_m \cup \{s_m\}$$

Após esta actualização, insere-se m em $\mathcal{C}_p$ e inicia-se $\mathcal{Q}_p(m) := \emptyset$. Estas actualizações são realizadas atomicamente. □

R6.3- Actualização na entrega: *Após entregar uma mensagem m, o processo $q \neq s_m$ actualiza as histórias papel-químico das mensagens anteriores (do mesmo emissor)*

do seguinte modo (isto é, mensagens enviadas pelo emissor de m, antes de m, estão implicitamente "relatadas" aos destinatários de m):

$$\textsf{para-todos} \quad (a \in \mathcal{C}_q : s_a = s_m \wedge c_a < c_m) \quad \textsf{execute-se} \quad \mathcal{Q}_q(a) := \mathcal{Q}_q(a) \cup \mathcal{A}_m$$

Seguidamente, integra todos os elementos $n \in \mathcal{C}_m$ em $\mathcal{C}_q$. O elemento da história papel-químico correspondente a cada mensagem n é iniciado com a união dos seguintes campos: a história papel-químico de n incluída na estampilha de m, caso exista (se a regra R6.1 não for usada, considere-se $\mathcal{Q}_m(n) = \emptyset$); o endereço de m (uma vez que m "relata" n a $\mathcal{A}_m$); o emissor de m (que necessariamente já "conhece" n); e, finalmente, com a união dos destinatários de todas as mensagens provenientes de s_n e posteriores a n (n pode não ser a última mensagem de s_n recebida localmente em q). Formalmente:

$$\mathcal{Q}_q(n) := \mathcal{Q}_m(n) \cup \mathcal{A}_m \cup \{s_m\} \cup \bigcup_{i \in \mathcal{C}_q : s_i = s_n \wedge c_i > c_n} \mathcal{A}_i$$

Caso $n \in \mathcal{C}_m$ já pertença a $\mathcal{C}_q$ procede-se apenas à actualização da história papel-químico, reunindo $\mathcal{Q}_q(n)$ com o resultado da expressão anterior. Finalmente, q insere m em $\mathcal{C}_q$ e inicia $\mathcal{Q}_q(m) := \{s_m, q\}$. Estas actualizações devem ser executadas numa única operação atómica. □

Tal como referido, as mensagens podem ser também estampilhadas com a história papel-químico do seu emissor. Isto é conseguido à custa do aumento do tamanho das mensagens. Sempre que for inaceitável aumentar o tamanho das mensagens, a história papel-químico pode ser actualizada usando exclusivamente informação acerca das mensagens localmente enviadas ou recebidas. A história papel-químico é usada para comprimir a história causal do seguinte modo: (1) um elemento da história causal não necessita de ser incluído na estampilha de uma mensagem m caso já tenha sido incluído na estampilha de uma mensagem anterior, enviada para o mesmo destinatário; e (2) assim que um elemento $\mathcal{Q}_q(m)$ da história papel-químico, local a um processo q, inclui todos os destinatários de m, o elemento $\mathcal{C}_q(m)$ correspondente pode ser eliminado da história causal uma vez que já foi "relatado" a todos os processos relevantes. Mais

precisamente,

R7 (Redundância na Estampilha): *Ao estampilhar uma mensagem m, o processo p inclui apenas em $\mathcal{C}_m$ os elementos da sua história causal $\mathcal{C}_p$ que ainda não foram relatados a $\mathcal{A}_m$, de acordo com o conhecimento de p, ou seja:* $\mathcal{C}_m := \bigcup i \in \mathcal{C}_p : \mathcal{A}_m \nsubseteq \mathcal{Q}_p(i)$. □

R8 (Redundância na História): *Numa história causal $\mathcal{C}_p$, se existe uma mensagem m tal que $\mathcal{A}_m \subseteq \mathcal{Q}_p(m)$, m pode ser eliminada de $\mathcal{C}_p$ e de $\mathcal{Q}_p$.* □

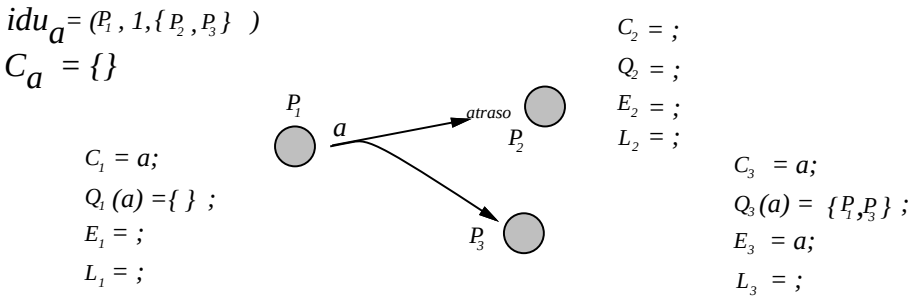
A Figura 5.2 apresenta um pequeno exemplo que ilustra o uso das regras que descartam informação redundante. Consideram-se três processos P_1, P_2, P_3. A figura mostra o conteúdo da sua história causal e história de entrega. A lista de mensagens à espera para serem entregues no processo P_i é designada $\mathcal{L}_i$. P_1 envia uma mensagem a para $\mathcal{A}_a = \{P_2, P_3\}$ a qual é recebida em P_3 mas não recebida em P_2 devido a um atraso na rede. Note-se que devido à regra R6.3, $\mathcal{Q}_3(a) = \{P_1, P_3\}$. De seguida P_3 envia uma mensagem b para P_2 a qual sofre também atrasos na rede. Note-se que devido à regra R6.2, $\mathcal{Q}_3(a) = \{P_1, P_3\} \cup \{P_2\} = \{P_1, P_2, P_3\}$. Uma vez que $\mathcal{A}_a \subset \mathcal{C}_3$, e de acordo com a regra R8, a mensagem a é removida de $\mathcal{C}_3$ e de $\mathcal{Q}_3$. Seguidamente, o processo P_3 envia outra mensagem c para P_2. A mensagem c ficará em fila de espera no processo P_2 até que $b \in \mathcal{C}_c$ seja entregue (esta, por sua vez, deverá esperar por a).

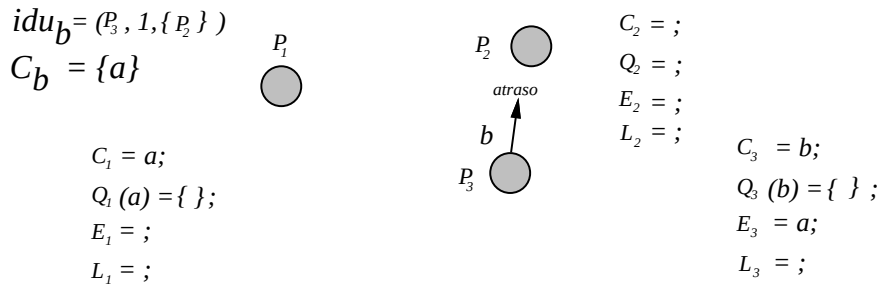
5.6.5 Estampilhagem topológica

Os separadores causais podem ser usados para reduzir o tamanho das estampilhas das mensagens do seguinte modo. Quando um membro de um separador causal estampilha uma mensagem endereçada exclusivamente a processos localizados no *conjunto posterior*, este omite da estampilha todas as mensagens da história causal que foram endereçadas exclusivamente para o *conjunto anterior* e que já foram "relatadas" aos restantes elementos do separador. Mais precisamente,

R9 (Estampilhagem topológica): O processo p envia uma mensagem m. Todas as mensagens $n \in \mathcal{C}_p$ para os quais existe um separador causal[3] (S, S^p, S^a), tal que: $p \in$

[3]Onde S^p e S^a representam respectivamente o conjunto posterior e o conjunto anterior ao separador





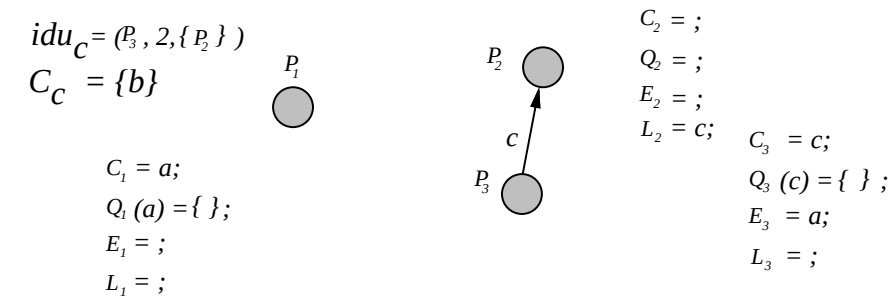


Figura 5.2: Um exemplo de utilização de histórias causais estendidas.

$S \wedge \mathcal{A}_m \subseteq S^p \wedge \mathcal{A}_n \subseteq S^a \wedge S \subseteq \mathcal{Q}_p(n)$, não necessitam de ser inseridas em $\mathcal{C}_m$. □

A compressão conseguida com a estampilhagem topológica pode ser melhorada à custa de "relatar" a informação para todos os elementos de um separador. Note-se que para uma determinada mensagem n poder ser omitida por p, de acordo com a regra R9, é necessário que $S \subseteq \mathcal{Q}_p(n)$. Ora de acordo com a regra R6.3, se a mensagem n pertencer à estampilha de uma mensagem x, isto é $n \in \mathcal{C}_x$, e se o endereço de x for tal que $S^p \subseteq \mathcal{A}_x$, após a entrega de x em p tem-se necessariamente $S \subseteq \mathcal{Q}_p(n)$. Na prática, isto significa que ao enviar mensagens para o grupo posterior, é possivel omitir da estampilha todas as mensagens endereçadas ao grupo anterior, bastando para isso que os membros do separador sejam endereçados em difusão. Na secção 5.6.9, dá-se um exemplo prático deste tipo de optimização.

Existem diversas dificuldades associadas ao uso da estampilhagem topológica. Em primeiro lugar, redes arbitrárias podem possuir um elevado número de separadores causais: a estampilhagem topológica pode ser aplicada a todos os separadores ou apenas a um subconjunto destes.

Em segundo lugar, os separadores causais necessitam de ser calculados antes da aplicação da estampilhagem topológica. Existem diversos algoritmos para identificar separadores em grafos (por exemplo, veja-se Even (1979)). No entanto, estes podem ser muito dispendiosos para ser executados frequentemente em tempo de execução. Deste modo, este método é mais aconselhado para aplicações onde a topologia seja relativamente estática ou possa ser calculada em tempo de compilação. Neste último caso, os separadores causais podem ser calculados em avanço, e os correspondentes conjuntos S, S^p e S^a carregados em todos os membros do separador de modo a permitir uma execução rápida da estampilhagem topológica. Uma topologia relativamente estática é aquela definida pela infra-estrutura de comunicação que liga os diversos nós de um sistema distribuído. Este caso é suficientemente importante — e de facto, motivou este trabalho — para merecer o estudo numa secção próxima.

Nesta secção mostrou-se como uma história papel-químico adicional pode ser usada para eliminar elementos da história causal usando informação acerca da topologia da

(veja-se a secção 5.6.3).

rede. Isto foi conseguido através de maiores custos de armazenagem, uma vez que esta informação auxiliar necessita de ser mantida em todos os processos (neste caso, a informação referente à história papel-químico). Aproximações relacionadas sofrem da mesma desvantagem (Singhal & Kshemkalyani, 1990; Stephenson, 1991).

5.6.6 Topologias não-estáticas

A outra dificuldade associada com o uso de separadores causais é que qualquer alteração na topologia pode alterar a filiação destes. Neste caso, é necessário tomar acções específicas sempre que novos processos se juntam ao sistema. Estas serão focadas nos próximos parágrafos.

Considera-se que a falha de um processo é permanente e que, caso a máquina em que aquele se executa recupere, ele será considerado um novo processo. Deste modo, existe apenas um cenário onde a falha de um processo pode pôr em perigo a entrega de mensagens de acordo com a ordem causal. Isto acontece quando o processo é o único membro do separador e a sua falha desliga o conjunto anterior do conjunto posterior. Neste caso, quando ambos os conjuntos são religados, através da criação de um novo separador causal, é necessário assegurar a recuperação da informação de causalidade mantida por esse processo. Distinguem-se quatro cenários para a entrada de novos processos no sistema:

- (i) O novo processo não modifica os separadores causais existentes no grafo.
- (ii) O novo processo cria um novo separador causal ligando dois sub-grafos nunca antes ligados.
- (iii) O novo processo expande um separador causal já existente, mas não cria novos separadores.
- (iv) O novo processo cria um novo separador que liga dois sub-grafos ligados no passado por outro separador.

Analisam-se os últimos dois casos uma vez que os dois primeiros casos são triviais (o novo processo não necessita de tomar nenhuma acção especial). No terceiro caso,

o processo deve obter a história causal dos outros membros do separador antes de começar a enviar ou receber mensagens. Por questão de simplicidade, assume-se que em cada separador as alterações na filiação são realizadas em série, de acordo com algum algoritmo de filiação distribuído. A nossa aproximação requer que o protocolo de filiação seja estendido para fornecer a cada novo membro a história causal dos restantes elementos do separador. Finalmente, no quarto caso, quando dois sub-grafos são religados após a falha completa de um separador anterior, três soluções podem ser aplicadas:

- Exigir que cada novo processo obtenha um estado global do sistema, de modo a obter todas as histórias de todos os membros do conjunto anterior. Este método é muito dispendioso para ser aplicado na prática.

- Fornecer a cada processo membro de um separador uma Fonte de Alimentação Não-Interrompível (tal como usada por Ladin *et al.* (1990)) ou uma memória RAM[4] não volátil. Considerando modelos de faltas mais restritivos, isto permite que a história do separador seja preservada em memória estável. Neste caso, as duas zonas poderiam ser religadas apenas por processos capazes de ler a memória estável do último processo no separador (usando por exemplo o método proposto por Skeen (1985) para determinar o último processo a falhar) de modo a obter a história causal do separador anterior.

- Prevenir a ocorrência deste caso, assegurando que os separadores contêm pelo menos $f+1$ membros de modo a tolerar f faltas. Isto pode ser obtido adicionando membros ao separador causal cujo objectivo será exclusivamente armazenar a história causal do separador. Estes membros adicionais funcionariam como *testemunhas* do tráfego atravessando o separador. Neste caso, os conjuntos anterior e posterior poderiam ser religados por qualquer processo capaz de obter uma história causal de um processo testemunha.

[4]Do Inglês, Random-Access Memory.

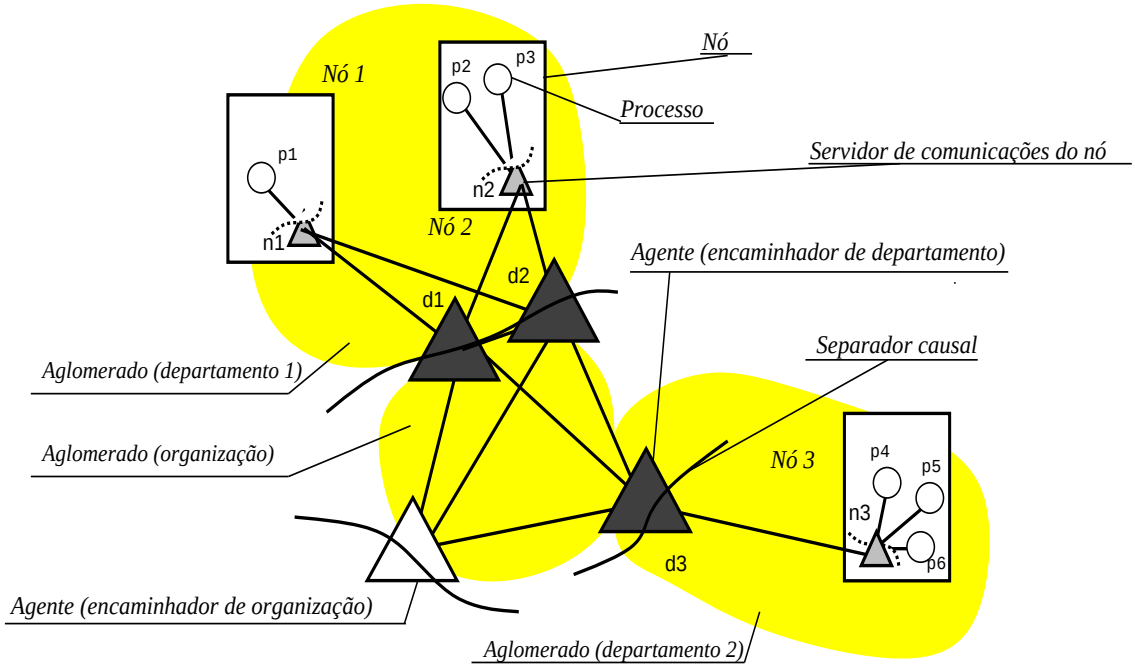


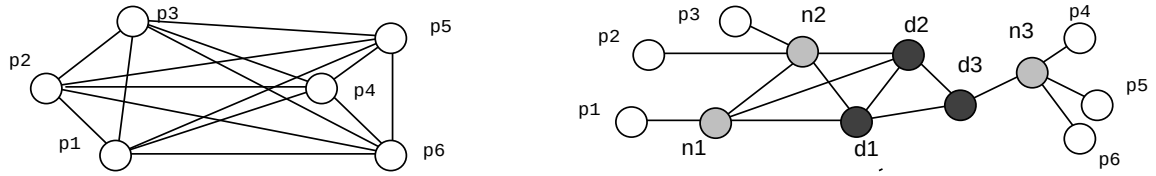
Figura 5.3: Arquitectura de comunicação.

5.6.7 Usando a topologia da comunicação

Até agora usou-se o termo *processo* para identificar as entidades computacionais de um modo bastante genérico. Uma possível concretização poderia aplicar o algoritmo à comunicação entre os processos (do sistema operativo). Nesse caso, o grafo de comunicação iria reflectir a estrutura de comunicação ao nível aplicação.

No Capítulo 3 identificou-se o carácter hierárquico da infra-estrutura de comunicação para a qual a arquitectura NAVTECH se encontra concebida. Estas propriedades estão ilustradas na Figura 5.3. Com base nessa estrutura realista, propõe-se a seguinte metodologia para fornecer ordem causal em sistemas de grande escala:

- a entidade que liga um nó à rede assume o papel de um processo na estrutura da comunicação. Este processo é um separador causal que liga a máquina à rede.
- o encaminhamento de mensagens entre nós é também executado por um processo específico, executando-se habitualmente em encaminhadores dedicados (designados por *agentes* na arquitectura NAVTECH). Estes processos são capazes de executar a estampilhagem topológica ao encaminharem as mensagens.



Grafo de comunicação lógico (nível aplicação). *Grafo de comunicação estendido.*

Figura 5.4: Projecção da estrutura da rede no grafo de comunicação.

Usando esta metodologia, é possível construir um *grafo de comunicação estendido* que tem em consideração a organização física da infra-estrutura de comunicação. Este grafo, inclui novos nós, correspondentes aos servidores de comunicações e processos encaminhadores (veja-se um exemplo na Figura 5.4). Promover entidades de comunicação a nós do grafo de comunicação (estendido) tem diversas vantagens, a saber:

- Fornece um método útil de projectar as caraterísticas físicas e organizacionais da infra-estrutura de comunicação num grafo de comunicação estendido ao qual se aplica a estampilhagem topológica. Isto permite identificar de maneira prática e útil os separadores causais relevantes (por exemplo, o servidor de comunicações, os encaminhadores do departamento, etc).

- A estrutura física da comunicação é relativamente estática. Uma vez que alterações à topologia são raras, os seus custos tornam-se desprezáveis, o que aumenta a eficiência da estampilhagem topológica.

- Fornece um mecanismo prático de aproveitar a localidade da comunicação. Aplicando a estampilhagem topológica ao grafo de comunicação estendido, é possível impedir que informação puramente local seja disseminada pela rede. Mensagens trocadas entre processos de uma mesma máquina são filtradas pelo servidor de comunicações; mensagens trocadas entre máquinas de um mesmo departamento, são-no pelo encaminhador desse departamento; e assim consecutivamente.

Naturalmente, também existem algumas desvantagens ligadas à utilização da estampilhagem topológica. No entanto, julga-se que estas são mínimas e virão a tornar-se ainda menos significativas com os avanços da tecnologia:

- Os servidores de comunicações e os encaminhadores devem executar a estampilhagem topológica. Deste modo, encaminhadores especializados necessitam de ser utilizados e isto pode ser uma desvantagem em comparação com soluções baseadas exclusivamente em componentes normalizados. No entanto, não só é tecnologicamente viável concretizar estes encaminhadores dedicados, como se espera que as normas venham a evoluir na direcção aqui proposta (em particular devido à necessidade de responder a novos requisitos tais como uso de difusão e fornecimento de diversas qualidades de serviço).

- Ao adicionar novos nós à estrutura de comunicação, o grau de concorrência do sistema diminui. Isto porque são estabelecidas relações de causalidade onde estas não existiam previamente. Deve-se no entanto salientar que esta limitação teórica tem pouco impacto prático, uma vez que por meio desta aproximação apenas são criados novos nós em pontos onde as mensagens já são fisicamente serializadas. Assim, as novas dependências causais só têm um impacto negativo quando se perdem mensagens. Tendo em conta que a taxa de erro das redes actuais é bastante baixa, a probabilidade de atrasar uma mensagem devido a outra mensagem originalmente não relacionada é praticamente desprezável.

- Os encaminhadores podem tornar-se um ponto de estrangulamento do sistema, e a execução da estampilhagem topológica aumenta a sobrecarga computacional do processo que executa o encaminhamento. No entanto, a capacidade de processamento tem vindo a aumentar (e a descer de custo) nos últimos anos. Deste modo, a razão custo/benefício de possuir máquinas dedicadas ao encaminhamento irá decrescer no futuro.

5.6.8 Concretização

O serviço de comunicação causal oferece três primitivas, cuja interface se resume na Tabela 5.3. A primitiva `g_c_causal` permite enviar uma mensagem em difusão virtualmente síncrona respeitando a causalidade. A primitiva `g_c_uniforme` oferece as mesmas garantias de ordenação, mas utiliza difusão uniforme virtualmente síncrona

Nome	Tipo	Parâmetros
g_c_causal	pedido	Grupo g, IdProcLista destinatários, Msg m
g_c_uniforme	pedido	Grupo g, IdProcLista destinatários, Msg m
g_c_entrega	indicação	Grupo g, IdProc rem, Msg m, IdProcLista lib, IdProcLista perd

Tabela 5.3: Interface do serviço causal

(veja-se a secção 2.3.2). A indicação `g_c_entrega` entrega a mensagem de acordo com a ordenação escolhida (a utilização dos parâmetros `lib` e `perd` será clarificada na secção 5.7).

Os mecanismos necessários para preservar a ordem causal foram descritos nas secções anteriores. Basicamente, é necessário preservar informação respeitante ao passado, na forma de identificadores únicos de mensagens e listas de identificadores de processos. Esta informação encontra-se estruturada em conjuntos designados por histórias, nomeadamente, a história causal, a história de entrega, e a história papel-químico.

Para além das estruturas atrás referidas, para concretizar a estampilhagem topológica, cada processo precisa de saber a que separadores causais pertence e quais os conjuntos anterior e posterior a estes associados. Apesar de existir a possibilidade de iniciar esta informação de modo automático, nos primeiros protótipos esta será iniciada manualmente, através de procedimentos de administração.

Usando as estruturas atrás referidas, uma concretização do protocolo causal limita-se a executar as diversas regras em sequência, tal como ilustrado na Figura 5.5. Mensagens recebidas que não possam ser entregues de imediato (por dependerem de mensagens em trânsito), são guardadas numa lista de espera designada por $\mathcal{L}_p$.

5.6.9 Desempenho

De modo a avaliar o desempenho desta aproximação recorreu-se à simulação. Para tal foi usado o simulador do grupo "MIT LCS Advanced Network Architecture" designado por NETSIM (Heybey, 1990). Mediu-se o tamanho das histórias causais e das estampilhas das mensagens, com e sem a aplicação da regra R9 (estampilhagem

quando *g_c_causal/g_c_uniforme (Grupo g, IdProcLista dest, Msg m) invocada no processo* p
execute-se
  início
    incrementar o contador local c_p*;*
    atribuir um identificador à mensagem retida $idu_m := (p, c_p,$ *dest);*
    estampilhar m *com* $\mathcal{C}_p$ *e* $\mathcal{Q}_p$*; // aplicar R7 e R9*
    actualizar $\mathcal{C}_p$ *e* $\mathcal{Q}_p$*; // aplicar R6.2 e R8*
    acrescentar idu_m *a* $\mathcal{C}_p$ *e a* $\mathcal{Q}_p$*;*
    g_sv_difusão (g, dest, m); // (para g_c_uniforme, use-se g_sv_uniforme)
  fim

quando *g_sv_entrega (Grupo g, IdProc remetente, Msg m)*
execute-se
  acrescentar m *a* $\mathcal{L}_p$*;*

quando $m \in \mathcal{L}_p \wedge \forall (x \in \mathcal{C}_m)(idu_x$ está-em $\mathcal{E}_p)$
execute-se
  // todas as mensagens precedentes já entregues
  início
    g_c_entrega (g, remetente, m*,* $\emptyset$*,* $\emptyset$ *); // entregar* m
    integrar $\mathcal{C}_m$ *em* $\mathcal{C}_p$ *(actualizar* $\mathcal{Q}_p$*); //aplicar R6.3 e R8*
    acrescentar idu_m *a* $\mathcal{C}_p$ *e a* $\mathcal{Q}_p$*;*
    acrescentar idu_m *a* $\mathcal{E}_p$*;*
  fim

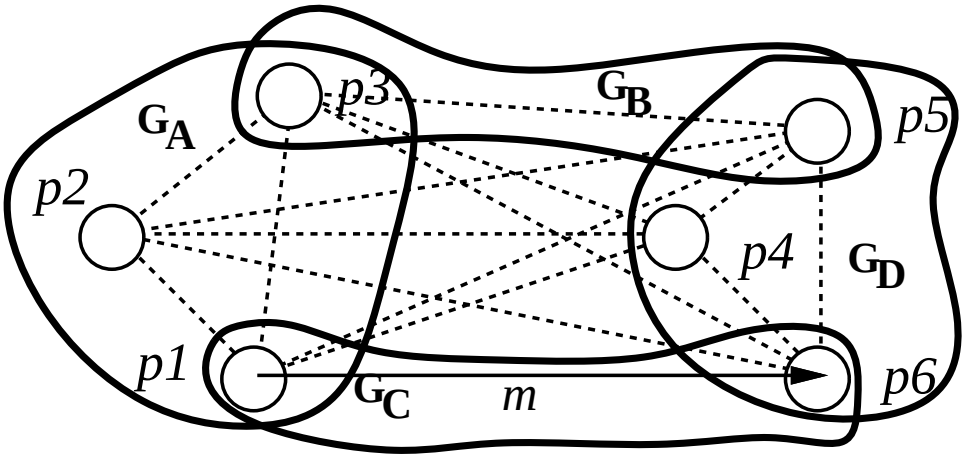
Figura 5.5: Concretização da ordem causal

topológica). Estes valores são comparados com os requeridos por versões não optimizadas do relógio-matricial (Raynal *et al.*, 1991) e dos relógios vectoriais (Stephenson, 1991).

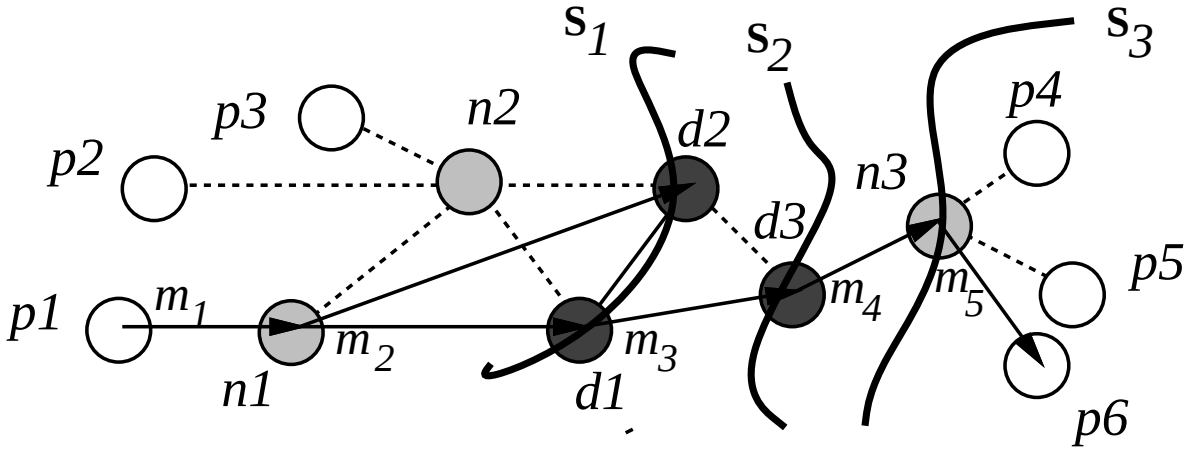
Os primeiros resultados apresentados foram obtidos com a rede da Figura 5.4. Ao nível da aplicação, são definidos quatro grupos como se ilustra na Figura 5.6. Os processos podem enviar e receber mensagens para os grupos a que pertencem (por exemplo, o processo p_1 pode enviar mensagens para os grupos G_A e G_C). Uma mensagem enviada no grafo do nível lógico é transformada numa cadeia de mensagens ao nível do grafo estendido (por exemplo, a mensagem m enviada no nível físico do processo p_1 para $G_C = \{p_1, p_6\}$ é projectada numa cadeia de mensagens m_1, m_2,m_3, m_4, m_5, com os seguintes endereços, $\mathcal{A}_{m_1} = \{n_1\}$, $\mathcal{A}_{m_2} = \{d_1, d_2\}$, $\mathcal{A}_{m_3} = \{d_3\}$, $\mathcal{A}_{m_4} = \{n_3\}$, $\mathcal{A}_{m_5} = \{p_6\}$). As histórias causais estendidas são mantidas e trocadas ao nível do grafo de comunicações estendido.

Neste exemplo, com seis processos, uma aproximação do tipo relógio-matricial não optimizada requer $6\text{x}6 = 36$ elementos na estampilha. Uma aproximação do tipo relógio-vectorial (com um vector para cada grupo) exigiria $3 + 3 + 2 + 2 = 10$ elementos. O valor médio obtido através da simulação é apresentado na Tabela 5.4 (foram usados os seguintes parâmetros: taxa de geração de mensagens de $10 mensagens/s$ em cada processo e latência média em cada elo de $50ms$). Tal como pode ser observado, mesmo sem recorrer à estampilhagem topológica, o tamanho médio das estampilhas é significativamente menor do que o obtido com as versões não optimizadas dos relógios-matriciais ou dos relógios-vectoriais. Estes valores são diminuídos ao se aplicar a regra da estampilhagem topológica, apenas no separador $S_2 = \{d_3\}$ (cenário 2) e também nos separadores $S_1 = \{d_1, d_2\}$ e $S_3 = \{n_3\}$ (cenário 3).

A tabela também contém valores obtidos com uma rede onde dois processos adicionais foram acrescentados aos grupos G_A (no nó n_1) e outros dois no grupo G_D (no nó n_3), resultando numa rede com 10 nós. O histograma com o tamanho das estampilhas para este cenário é apresentado na Figura 5.8. Tal como se pode ver, o tamanho médio das estampilhas diminui menos (em proporção) do que com as restantes aproximações. Simulou-se a mesma topologia com diferentes atrasos na rede sem obter diferenças



Grafo de comunicação lógico (nível aplicação)



Grafo de comunicação estendido.

Figura 5.6: Topologia com seis processos.

Cenário	$n = 6$	$n = 10$
Relógio-matricial	36	100
Relógios-vectoriais	10	14
Histórias causais estendidas		
Sem R9	3.55	3.46
R9 em S_2	2.70	3.09
R9 em S_1, S_2, e S_3	2.10	2.76

Tabela 5.4: Tamanho médio das estampilhas.

significativas.

No exemplo anterior, o grafo de comunicações estendido possuía baixa conectividade, pelo que a sua definição — por si só — permitia reduzir o tamanho das estampilhas mesmo sem aplicar a regra da estampilhagem topológica. As vantagens desta regra são mais evidentes se, em vez de se medirem valores médios no sistema como um todo, se medir o impacto da aplicação da regra na comunicação local. Considere-se por exemplo a rede da Figura 5.9, na qual um pequeno grupo de nós interage com um grupo maior através de um único encaminhador. Sem se usar a estampilhagem topológica, a informação causal respeitante ao tráfego no grupo G2 é propagada para o grupo G1, induzindo um tamanho médio das estampilhas de 32 elementos. Caso o encaminhador seja usado como um separador, o tamanho médio das estampilhas no grupo G1 é reduzido para menos que 5 elementos, o que corresponde à influência do grupo local adicionada aos grupos G3 e G4.

A característica interessante da técnica aqui apresentada é que o mesmo tipo de ganhos pode ser obtido mesmo quando existem dois ou mais encaminhadores. Considere-se agora o grafo da Figura 5.10, consistindo de três redes de difusão interligadas. Claramente, neste grafo existem ciclos tanto ao nível lógico como ao nível físico. Assuma-se que o encaminhamento é realizado de tal maneira que, em cada rede de difusão, ambos os encaminhadores recebem o tráfego para o exterior (uma vez que estão interligados por um meio de difusão, isto pode ser obtido com um custo desprezável), e que estes encaminhadores podem aplicar a estampilhagem topológica de modo a filtrar informação associada a comunicação puramente local.

O impacto de aplicar a estampilhagem topológica neste caso está ilustrado na Figura 5.11. A figura mostra a distribuição do tamanho das estampilhas das mensagens trocadas no grupo $G1$ com e sem aplicação da estampilhagem topológica. A figura mostra também a distribuição de história causal no processo $p1$ para os mesmos dois casos. Pode-se observar que a história causal no processo $p1$ é muito maior quando não se utiliza a estampilhagem topológica, devido à necessidade de armazenar localmente informação respeitante a mensagens trocadas em grupos remotos.

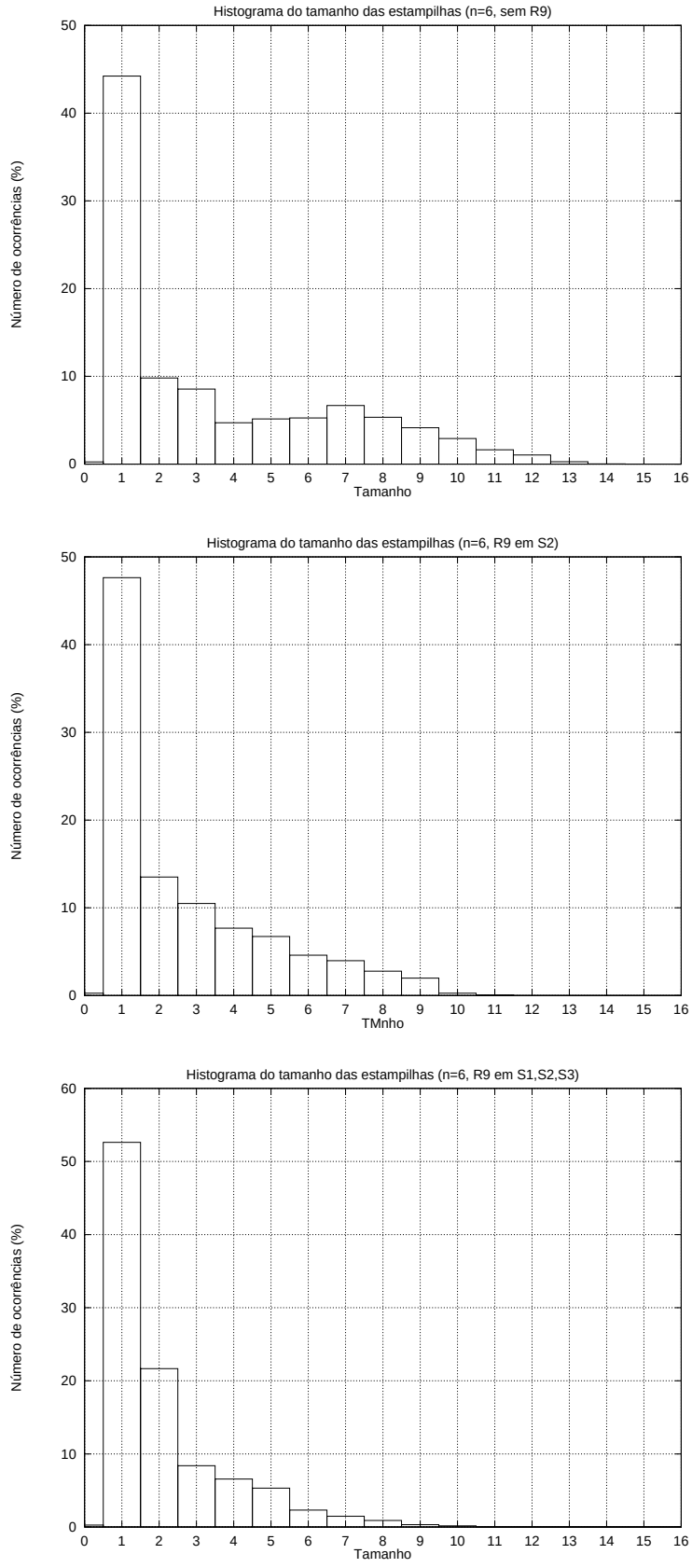


Figura 5.7: Tamanho das estampilhas (topologia com 6 processos).

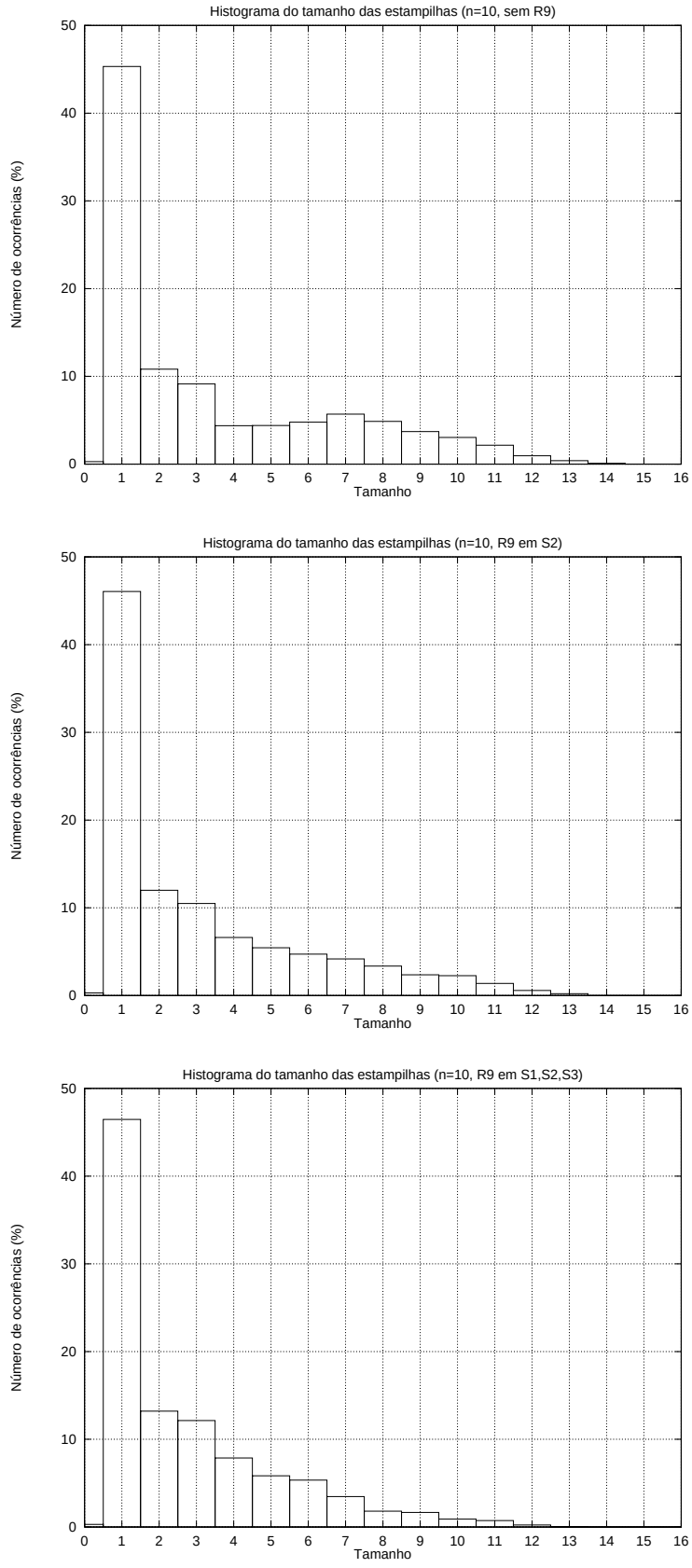


Figura 5.8: Tamanho das estampilhas (topologia com 10 processos).

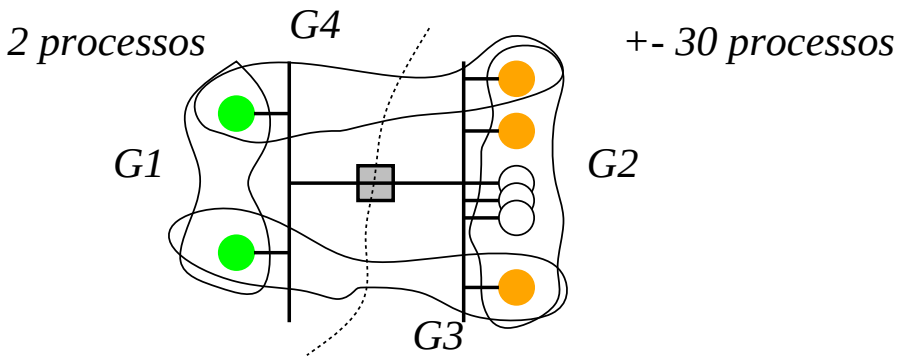


Figura 5.9: Um pequeno grupo que interage com um grupo grande.

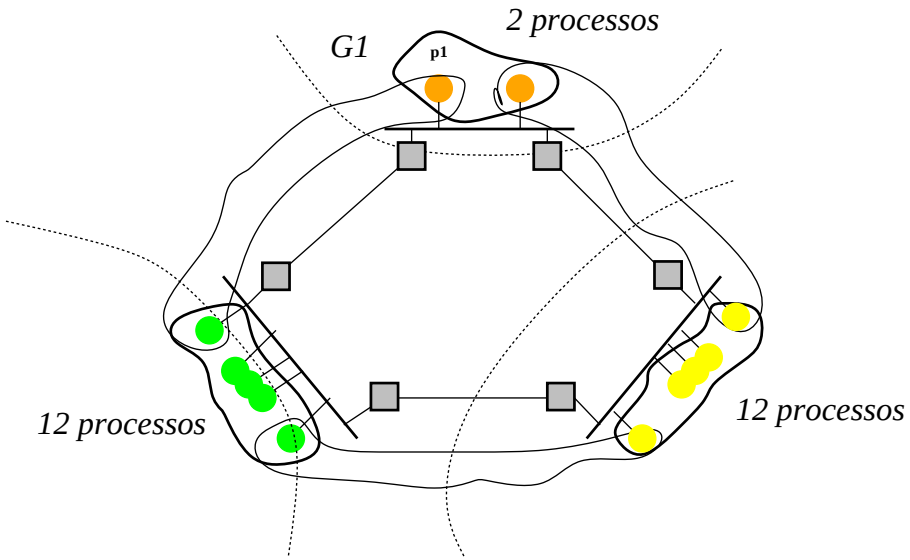


Figura 5.10: Três redes de difusão interligadas.

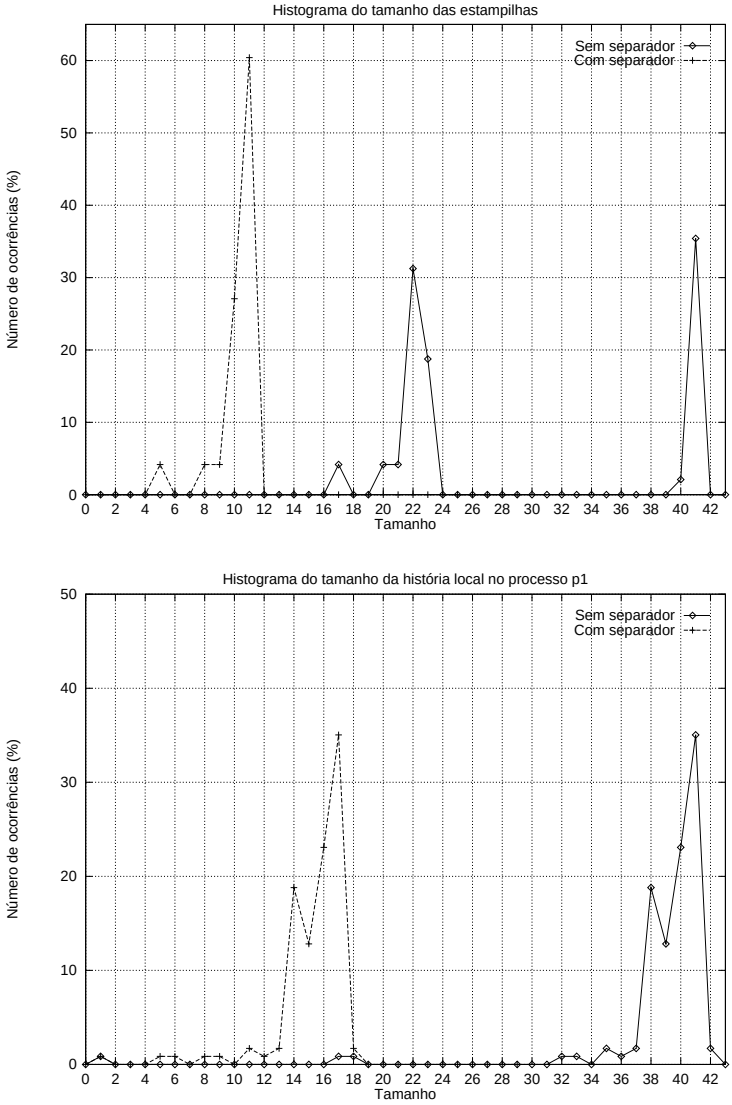


Figura 5.11: Impacto no grupo $G1$.

5.6.10 Observações

Estudou-se o problema de garantir a entrega causal de mensagens em redes de grande escala. Propôs-se uma representação de histórias causais que não impõe restrições ao endereçamento. Mostrou-se que esta representação permite melhorar resultados anteriores na redução das estampilhas através da informação acerca da topologia da rede. A técnica baseia-se na identificação de *separadores causais*, i.e., separadores de nós no grafo de comunicações, e filtrar a informação que atravessa esses separadores. De modo a fazer uso prático destes resultados, apresentou-se uma técnica que explora as características das redes existentes, em particular a sua natureza

hierárquica, para criar um grafo de comunicações onde os separadores causais emparelham com as organizações administrativas e físicas subjacentes. Mostrou-se que esta aproximação pode ser aplicada a sistemas de grande escala existentes, fornecendo os meios para a aplicação da estampilhagem topológica com baixa sobrecarga.

5.7 Mensagens Transparentes

5.7.1 Motivação e definição

Uma da críticas feitas aos sistemas de comunicação causal é a destes requererem a utilização obrigatória de primitivas de comunicação fiável (Cheriton & Skeen, 1993). De facto, desde que uma mensagem introduza uma relação de dependência causal no sistema, a entrega dessa mensagem deve ser garantida; caso contrário as mensagens que dela dependem não mais poderão ser entregues. Isto obriga a que a entrega de qualquer mensagem causal a um dos seus destinatários seja atrasada até que se possa garantir a entrega a todos os outros destinatários. Para além disso, por vezes o próprio emissor fica impedido de transmitir novas mensagens até que essas garantias estejam asseguradas.

Estas limitações não invalidam a utilidade do uso de comunicação causal. Como se viu anteriormente, existem muitas aplicações que beneficiam da existência deste tipo de primitivas. Existem também sinais claros de que quando a ordenação causal é necessária, soluções ad-hoc para o problema acabam por ser bastante complexas (Birman, 1994). Para resolver esta contradição, propõe-se uma técnica que distingue dois tipos de mensagens: mensagens *opacas* e mensagens *transparentes* (Rodrigues & Veríssimo, 1994). Mensagens causais transparentes são mensagens que são entregues respeitando a ordem causal em relação às mensagens opacas mas que não introduzem dependências causais. Assim:

- nenhuma mensagem pode ser atrasada devido a uma mensagem transparente;
- não são impostos requisitos de fiabilidade à disseminação das mensagens transparentes.

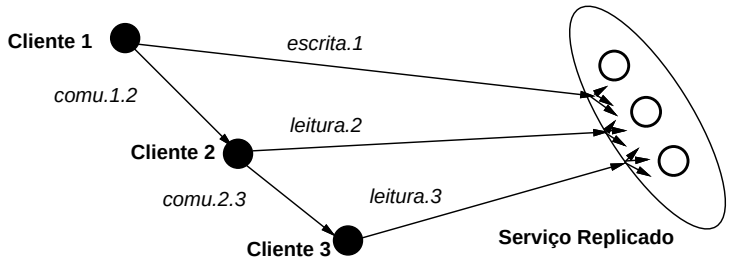


Figura 5.12: Um exemplo de utilização de mensagens transparentes.

A ordem de entrega das mensagens transparentes em relação às mensagens opacas é descrita na tabela seguinte (na qual as mensagens opacas são representadas em maiúsculas, mensagens transparentes representadas em minúsculas, e a seta $\rightarrow$ representa a relação transitiva de causalidade como definida anteriormente).

relação	ordem de entrega	relação	ordem de entrega
$M_1 \rightarrow M_2$	M_2 após M_1	$m_1 \rightarrow M_2$	indefinido
$M_1 \rightarrow m_2$	m_2 após M_1	$m_1 \rightarrow m_2$	indefinido

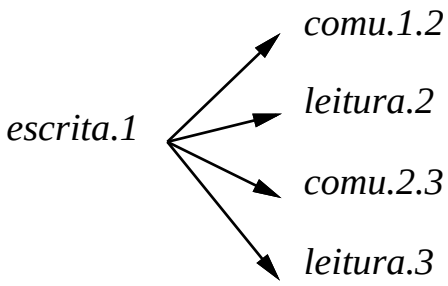
5.7.2 Exemplos de utilização

No âmbito dos sistemas de comunicação de grande escala, as mensagens transparentes podem ser extremamente úteis para concretizar interacções cliente-servidor que respeitam a ordem causal. Isto porque, devido às suas características, estes sistemas tornam relativamente dispendioso oferecer garantias de entrega fiável.

Considere-se o exemplo da Figura 5.12, que ilustra o acesso a um serviço replicado oferecendo duas primitivas de *leitura* e *escrita* respectivamente. Assuma-se que os clientes interagem com o serviço replicado usando difusão fiável. O modo como a coerência do servidor é mantida é ortogonal a este exemplo. Essa coerência pode ser obtida por recurso a comunicação adicional entre as réplicas ou reforçando a semântica da comunicação com os clientes, por exemplo, obrigando os clientes a usar comunicação totalmente ordenada. Neste exemplo, os clientes também podem comunicar entre si (quer através de comunicação ponto-a-ponto, quer através de difusão). Para ilustrar as vantagens das mensagens transparentes recorre-se de uma execução em que são observadas as seguintes relações de causalidade entre as mensagens:

$$escrita.1 \rightarrow comu.1.2 \rightarrow leitura.2 \rightarrow comu.2.3 \rightarrow leitura.3$$

Caso seja oferecida uma única primitiva de comunicação causal (opaca), não só todas as difusões devem ser fiáveis mas também qualquer atraso na difusão, por exemplo, da mensagem *leitura.2*, iria atrasar a execução da operação *leitura.3*. Isto pode representar um custo inaceitável para algumas aplicações. Agora, assuma-se que todos os pedidos de leitura e respectivas respostas (não representadas na figura), são realizados recorrendo à utilização de mensagens transparentes. Os clientes 2 e 3 continuam a ter a garantia de observar os resultados da operação de escrita executada pelo cliente 1. No entanto, as operações de leitura podem agora ser executadas recorrendo às mensagens transparentes, que oferecem maior desempenho pois não asseguram as mesmas garantias de fiabilidade (estas garantias podem ser obtidas através da espera de respostas e eventuais retransmissões). Sobretudo, o uso de mensagens transparentes evita que atrasos numa dada operação de leitura se propaguem a outras operações. Por exemplo, a mesma sequência de mensagens geraria as seguintes relações de causalidade:



O exemplo anterior, apesar de bastante simples, ilustra claramente as vantagens do uso de mensagens transparentes. Este exemplo pode ser expandido, de modo a realizar todas as interacções entre o cliente e os servidores através de mensagens transparentes, restringindo a utilização de mensagens opacas à comunicação entre as diversas réplicas do servidor. No capítulo 6 apresenta-se um protocolo de invocação remota com estas características.

Nome	Tipo	Parâmetros
g_c_transparente	pedido	Grupo g, IdProcLista destinários, Msg m

Tabela 5.5: Interface do serviço de mensagens transparentes

5.7.3 Concretização

O serviço de mensagens transparentes oferece uma nova primitiva, cuja interface se resume na Tabela 5.5. Utilizando a técnica apresentada na secção 5.6, a concretização deste serviço é trivial. Uma mensagem transparente é estampilhada com a história causal do seu emissor mas nunca dá origem a um novo identificador único e como tal, nunca é introduzida nem nas histórias de entrega nem nas histórias causais (nem, consequentemente, nas histórias papel-químico). Note-se que, deste modo, um processo que utilize exclusivamente mensagens transparentes nunca contribui para o volume da informação que é necessário armazenar para preservar a causalidade. Quando a mensagem é recebida, a sua entrega é atrasada até que todas as mensagens precedentes tenham sido entregues. Quando a mensagem é entregue, a sua estampilha é adicionada à história causal e à história papel-químico do destinatário. Este procedimento encontra-se ilustrado na Figura 5.13.

Dado que as mensagens transparentes não são transmitidas de modo fiável, não existe a garantia de que a informação causal que transportam na sua estampilha seja "relatada" aos destinatários. Por este motivo, a transmissão ou a entrega de uma mensagem transparente não permite concretizar todas as optimizações descritas na secção 5.6. Deste modo, aplicam-se as seguintes regras:

R10- Envio transparente: *Antes de ser enviada, a mensagem m é estampilhada com a história causal $\mathcal{C}_p$ e, opcionalmente, também com a história papel-químico do remetente $\mathcal{Q}_p$. A transmissão de m não altera o conteúdo destas estruturas.* □

R11- Entrega transparente: *Após entregar uma mensagem m, o processo $q \neq s_m$ integra todos os elementos n de $\mathcal{C}_m$ em $\mathcal{C}_q$. O elemento da história papel-químico correspondente a cada mensagem n é iniciado com a união dos seguintes campos: a história papel-químico de n incluída na estampilha de m, caso exista (caso contrário, considere-se $\mathcal{Q}_m(n) = \emptyset$); e a união dos destinatários de todas as mensagens provenientes de s_n*

quando *g_c_transparente (Grupo g, IdProcLista dest, Msg m) invocada no processo* p
execute-se
início
estampilhar m *com* $\mathcal{C}_p$ *e* $\mathcal{Q}_p$*; // R10*
g_ra_mesforço (g, dest, m);
fim

quando *g_ra_entrega (Grupo g, IdProc remetente, Msg m)*
execute-se
acrescentar m *a* $\mathcal{L}_p$

quando $m \in \mathcal{L}_p \wedge \forall (x \in \mathcal{C}_m)(idu_x$ está-em $\mathcal{E}_p)$
execute-se
// todas as mensagens precedentes já entregues
início
g_c_entrega (g, remetente, m*,* $\emptyset$*,* $\emptyset$ *); //entregar* m
integrar $\mathcal{C}_m$ *em* $\mathcal{C}_p$ *(actualizar* $\mathcal{Q}_p$*); // R11*
fim

Figura 5.13: Concretização das mensagens transparentes

e posteriores a n *(*n *pode não ser a última mensagem de* s_n *recebida localmente em* q*). Formalmente:*

$$\mathcal{Q}_q(n) := \mathcal{Q}_m(n) \cup \bigcup_{i \in \mathcal{C}_q : s_i = s_n \wedge c_i > c_n} \mathcal{A}_i$$

Caso $n \in \mathcal{C}_m$ *já pertença a* $\mathcal{C}_q$ *procede-se apenas à actualização da história papel-químico, reunindo* $\mathcal{Q}_q(n)$ *com o resultado da expressão anterior.* □

5.8 Mensagens Retidas

5.8.1 Motivação

Muitos sistemas tentam aproveitar conhecimento acerca da semântica das aplicações com o objectivo de aumentar o seu desempenho, através do enfraquecimento selectivo da coerência das réplicas. Em consequência, obtêm-se geralmente réplicas que, apesar de não serem idênticas, não violam os requisitos de congruência

das aplicações. Com o objectivo de fornecer aos clientes execuções que não violam a causalidade, sem sacrificar os benefícios que se podem obter através do uso de modelos de coerência enfraquecida, propõe-se oferecer um serviço de comunicação que suporta aquilo que se designou por *mensagens retidas*.

Uma mensagem retida é uma mensagem que é enviada em termos lógicos mas que não é fisicamente propagada para o meio. Sempre que a transmissão de uma mensagem retida é requerida, cria-se uma dependência causal tal como se uma mensagem opaca tivesse sido enviada mas, em vez de transmitir fisicamente os dados a ela associados, estes são armazenados pela aplicação para futura transmissão. Deste modo uma aplicação pode ir retendo diversas mensagens que são depois enviadas numa única transmissão. No entanto, o sistema de comunicação reconhece a existência destas mensagens e, caso a entrega de uma mensagem esteja comprometida devido a uma dependência em relação a uma mensagem retida, o sistema de comunicação invoca o seu emissor para que este a transmita de imediato.

O mecanismo de mensagens retidas permite que diversas mensagens sejam aglomeradas e enviadas numa única transmissão de modo a reduzir o número de mensagens transmitidas fisicamente. Permite-se assim que a aplicação assuma o controlo dos mecanismos de aglomeração sem que com isso fique obrigada a manter a informação de precedência causal dessas mesmas mensagens. Nomeadamente, toda a informação necessária para manter a causalidade continua a ser gerida por um único módulo.

5.8.2 Interface

O serviço de mensagens retidas oferece duas novas primitivas: uma primitiva `g_c_retida` que permite "transmitir" uma mensagem retida; e uma primitiva `g_c_liberta` que permite ao sub-sistema de comunicação requisitar a um processo a transmissão física, para um conjunto de processos, do aglomerado de mensagens ainda retidas. A primitiva de indicação de recepção de uma mensagem é também enriquecida com dois novos parâmetros, cuja funcionalidade será descrita de seguida. A interface encontra-se resumida na Tabela 5.6.

Nome	Tipo	Parâmetros
g_c_causal	pedido	Grupo g, IdProcLista destinatários, Msg m
g_c_retida	pedido	Grupo g, IdProcLista dest
g_c_entrega	indicação	Grupo g, IdProc rem, Msg m, IdProcLista lib, IdProcLista perd
g_c_liberta	indicação	Grupo g, IdProcLista necessárias

Tabela 5.6: Interface do serviço de mensagens retidas

- `g_c_causal` aceita como parâmetros um identificador de grupo, uma lista de destinatários e uma mensagem a enviar. Pressupõe-se que, caso existam mensagens retidas para qualquer um dos processos na lista de destinatários, a mensagem submetida contém a agregação de todas essas mensagens.

- `g_c_retida` cria uma dependência causal para uma mensagem que fica retida no contexto do cliente. Uma vez que fisicamente não é transmitida nenhuma informação, não é necessário especificar uma mensagem, bastando apenas indicar os destinatários.

- `g_c_entrega` é a indicação gerada quando uma mensagem é entregue. É fornecida, para além da mensagem entregue, a indicação do remetente. Quando a mensagem entregue depende de mensagens retidas que ainda não tinham sido recebidas no instante da sua recepção (e, consequentemente, obrigou à execução de um pedido de libertação de mensagens retidas), a indicação dos processos contactados é fornecida no parâmetro `lib` . Quando a mensagem entregue depende de mensagens retidas que se perderam devido à falha do remetente, a identidade dos remetentes falhados é fornecida em `perd` .

- `g_c_liberta` é uma indicação gerada pelo sistema de comunicação para requisitar a transmissão de mensagens retidas. Esta indicação é geralmente provocada pela recepção, num outro processo, de uma mensagem causal que depende de mensagens retidas pelo cliente local. A lista de processos onde a mensagem retida é necessária para permitir o progresso do sistema é indicada no parâmetro `necess´arias` . Para um melhor desempenho do sistema, o cliente deve enviar de imediato uma mensagem causal contendo o agregado das mensagens retidas (destinadas aos processos em causa).

Considere-se o exemplo da Figura 5.14. O Cliente 1 pede uma actualização na Réplica 1. Esta não propaga de imediato esta actualização para a Réplica 2, utilizando antes uma mensagem retida. O mesmo se passa com a actualização do Cliente 2. As confirmações retornadas para os clientes dependem das mensagens retidas entretanto enviadas (as dependências são ilustradas por círculos de fundo branco e a propagação física dos dados por círculos de fundo escuro). Se o Cliente 2 contactar um outro cliente, neste caso o Cliente 3, estas dependências vão ser propagadas. Finalmente, se o Cliente 3 decidir realizar uma actualização na Réplica 2, o sub-sistema de comunicação detecta automaticamente a falta das mensagens retidas e pede para estas serem enviadas. Quando a Réplica 1 recebe a indicação `g_c_liberta`, envia ambas as actualizações numa única mensagem.

5.8.3 Concretização

O objectivo do serviço de mensagens retidas é dar ao utilizador controlo sobre políticas de agregação de mensagens, reutilizando os mecanismos de preservação de causalidade concretizados pelo sistema de comunicação. Não é pois surpreendente, que os mecanismos descritos na secção 5.6 deste capítulo não necessitem de alterações significativas para suportar este serviço adicional.

A principal diferença consiste na necessidade de distinguir os identificadores das mensagens retidas dos identificadores das restantes mensagens. Para isso, o identificador único de mensagem $idu_m \equiv (t_m, s_m, c_m, \mathcal{A}_m)$ é enriquecido com campo t_m que especifica o tipo da mensagem, o qual pode tomar um dos seguintes valores: `retida` ou `causal`.

A invocação da primitiva `g_c_retida` não faz mais do que atribuir um identificador à mensagem retida. Este identificador é acrescentado à história causal $\mathcal{C}_p$ e à história papel-químico $\mathcal{Q}_p$ do remetente, tal como o identificador de uma mensagem causal. Dado que nenhuma informação é disseminada, nenhuma outra alteração é aplicada a estas histórias. Este procedimento está ilustrado na Figura 5.15.

O procedimento de entrega de mensagens necessita também de ser ligeiramente

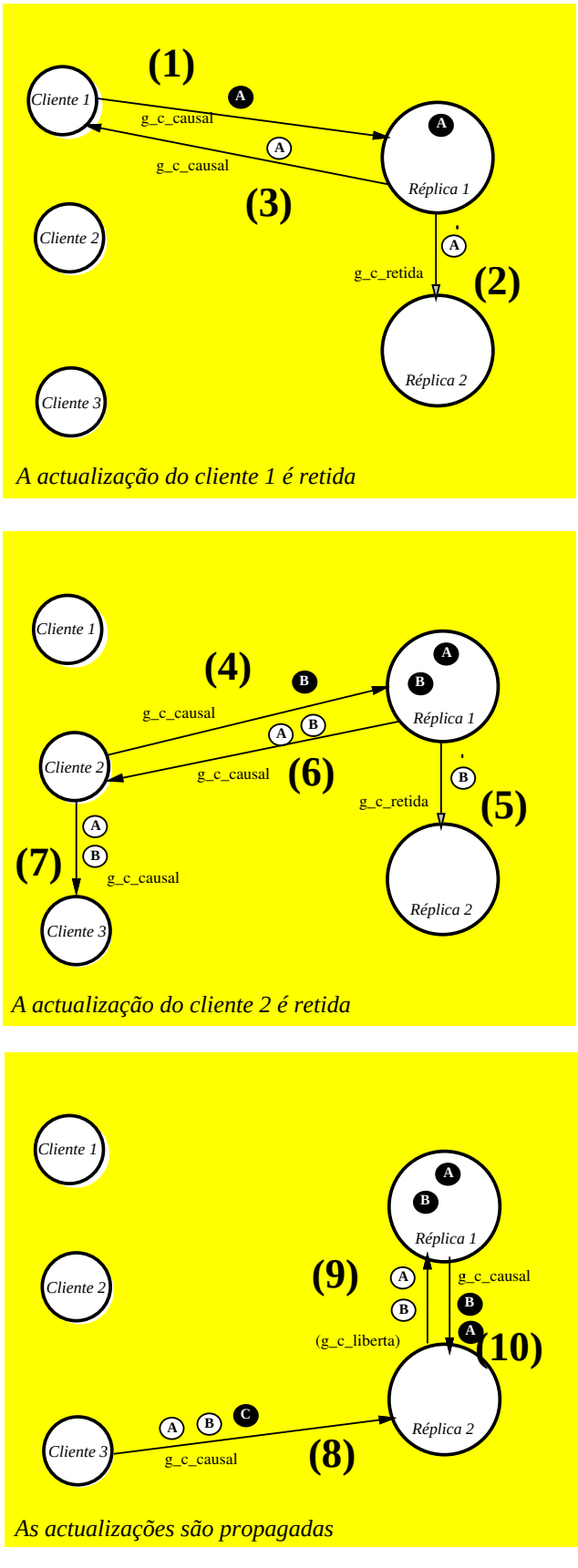


Figura 5.14: Utilização de mensagens.

```
quando  g_c_retida (Grupo g, IdProcLista dest) invocada no processo p
execute-se
   início
      incrementar o contador local c_p;
      atribuir um identificador à mensagem retida idu_m := (retida, p, c_p, dest);
      acrescentar idu_m a C_p e a Q_p;
   fim
```

Figura 5.15: Marcação de uma mensagem retida

alterado, uma vez que uma mensagem causal pode depender de mensagens retidas (veja-se a Figura 5.16). Assim, qualquer mensagem recebida é mantida em lista de espera $\mathcal{L}_p$ até que todas as mensagens *causais* precedentes sejam entregues. Assim que esta condição se verifica, avalia-se o estado das mensagens retidas precedentes (ou seja, de todas as mensagens $x \in \mathcal{C}_m$ tal que $t_x =$ `retida`). Cada uma destas mensagens x pode encontrar-se num dos seguintes estados:

- a mensagem retida x já foi entregue, isto é, $idu_x \in \mathcal{E}_p$. Neste caso, x não impede a entrega de m.
- a mensagem retida x vem agregada a m ou a outra mensagem causal que precede m. Por definição, isto verifica-se sempre que x precede m e ambas as mensagens são enviadas pelo mesmo processo. Neste caso, x também não impede a entrega de m.
- a mensagem retida x não se encontra em nenhuma das situações anteriores. Neste caso, envia-se um pedido de libertação das mensagens retidas para o remetente de x. Esse pedido, ao ser recebido, despoleta uma indicação do tipo `g_c_liberta` .

5.8.4 Agregação automatizada de mensagens

O serviço de mensagens retidas permite à aplicação concretizar políticas de agregação especializadas. Por esta razão espera-se que, na grande maioria dos casos, a aplicação tome o controlo explícito destes mecanismos, usando conhecimento

```
quando  g_sv_entrega (Grupo g, IdProc remetente, Msg m) indicada no processo p
execute-se
   início
      acrescentar m a L_p;
      liberta_m := ∅;
      perdida_m := ∅;
   fim

quando  m ∈ L_p ∧ ∀(x ∈ C_m : t_x = causal)(idu_x está-em E_p)
execute-se
// todas as mensagens causais já entregues
   se (∀(x ∈ C_m : t_x = retida)(idu_x está-em E_p ∨ s_x = s_m)) então
      g_entrega (Grupo g, s_m, m, liberta_m, perdida_m);
      integrar C_m em C_p (actualizar Q_p);
      acrescentar idu_m a E_p;
   caso-contrário se (∃(x ∈ C_m : t_x = retida ∧ s_x ∉ vista-de-grupo)) então
      início
         remover x de C_m;
         perdida_m := perdida_m ∪ {x};
      fim
   caso-contrário então
      // existem mensagens retidas
      início
         liberta_m := liberta_m ∪ remetentes-de-mensagens-em-falta;
         // pede libertação das mensagens
         g_sv_difusão (g, ⟨ liberta ⟩, remetentes-de-mensagens-em-falta);
      fim
   fim-se

quando  g_sv_entrega (Grupo g, remetente, ⟨ liberta ⟩)
execute-se
   g_c_liberta (g, remetente);
```

Figura 5.16: Entrega de uma mensagem causal/retida

Nome	Tipo	Parâmtros
saam_entrega	indicação	Grupo g, IdProc remetente, Msg m
saam_diferido	pedido	Grupo g, IdProcLista destinatários, Msg m
saam_imediato	pedido	Grupo g, IdProcLista destinatários, Msg m
saam_emite	pedido	Grupo g, IdProcLista destinatários

Tabela 5.7: Interface do SAAM

semântico para optimizar o número e dimensão das mensagens trocadas. Por exemplo, um serviço de gestão de dados replicados pode concretizar um serviço de disseminação de actualizações "a-pedido", marcando cada actualização local com uma mensagem retida e, quando necessário, enviando a lista das alterações feitas numa única mensagem. Para permitir uma rápida prototipagem, é possível fornecer bibliotecas concretizando políticas de agregação pré-definidas. A título de exemplo de utilização do serviço de mensagens retidas, apresentamos um Serviço de Agregação Automatizada de Mensagens (SAAM).

O SAAM oferece quatro primitivas para gerir as mensagens retidas, tal como ilustrado na Tabela 5.7: a indicação `saam_entrega` é utilizada para entregar uma mensagem ao utilizador; `saam_diferido` permite submeter uma mensagem para posterior transmissão ao SAAM, a qual será retida automaticamente; `saam_imediato` envia uma mensagem para a rede, agregando automaticamente as mensagens entretanto retidas; `saam_emite` permite enviar o agregado de todas as mensagens retidas (para os destinatários indicados) sem ter de emitir nenhuma mensagem adicional. O SAAM é também encarregue de interseptar a indicação `g_c_liberta` gerada pelo sistema de comunicação e, automaticamente, libertar todas as mensagens pedidas.

Uma concretização simplificada do SAAM encontra-se na Figura 5.17 (para simplificar a descrição, omite-se o identificador de grupo). O objectivo desta concretização é ilustrar a funcionalidade do serviço de mensagens retidas e não optimizar o protocolo. As mensagens trocadas entre entidades correspondentes do nível SAAM são o resultado da agregação ordenada de diversas mensagens do utilizador. Ao enviar uma mensagem imediata para um conjunto de destinatários, o SAAM agrega todas as mensagens retidas que possuam um destinatário comum com a mensagem imediata que se pretende transmitir, e envia a mensagem composta. Aquando da sua recepção,

```
Iniciação:
    declara   ListaDeMsg pendentes := ∅;

    declara   função saamAgrega (IdProcLista dest, Msg m) devolve   ListaDeMsg, IdProcLista
      IdProcLista endereço := dest;
      ListaDeMsg agrega := m;
      ∀ m em pendentes
        se ((A_m ∩ endereço) ≠ ∅) então
          retira m de pendentes;
          agrega := agrega ∪ m;
          endereço : = endereço ∪ A_m;
        fim-se;
      devolve (agrega, endereço);
Corpo:
    quando   saam_diferido (IdProcLista dest, Msg m) invocada
    execute-se
      início
        pendentes := pendentes ∪ m;
        g_c_retida (dest);
      fim

    quando   saam_imediato (IdProcLista dest, Msg m) invocada
    execute-se
      início
        (agregação, endereço) := saamAgrega (dest, m);
        g_c_causal (agregação, endereço);
      fim

    quando   saam_emite (IdProcLista dest) ou g_c_liberta ( IdProcLista dest) invocada
    execute-se
      início
        (agregação, endereço) := saamAgrega (dest, ∅);
        g_c_causal (agregação, endereço);
      fim

    quando   g_c_entrega (IdProc remetente, Msg m, IdProcLista lib, IdProcLista perd) invocada
    execute-se
      indicar saam_entrega   para todas as mensagens agregadas destinadas ao processo;
```

Figura 5.17: Serviço de Agregação Automatizada de Mensagens

o SAAM desagrega a mensagem composta e entrega as mensagens pela ordem por que foram enviadas.

5.9 Ordem total

5.9.1 Motivação

Os protocolos de difusão fiável totalmente ordenados provaram ser de extrema utilidade para suportar muitas aplicações distribuídas. Por exemplo, a ordem total é um requisito para o desenvolvimento de máquinas-de-estado replicadas (Schneider, 1990), um paradigma genérico para a concretização de aplicações tolerantes a faltas. Apesar de diversos protocolos se encontrarem descritos na literatura (Chang & Maxemchuck, 1984; Birman & Joseph, 1987; Peterson *et al.*, 1989; Veríssimo *et al.*, 1989; Ladin *et al.*, 1990; Kaashoek & Tanenbaum, 1991; Birman *et al.*, 1991a; Dolev *et al.*, 1993b; Amir *et al.*, 1993), poucos foram especialmente concebidos para operar em grande escala.

Em sistemas de grande escala os padrões de tráfego dos processos são habitualmente heterogéneos. O mesmo se aplica às propriedades dos meios de inter-ligação: alguns processos estão localizados na mesma rede local, outros separados por ligações sujeitas a grandes atrasos (veja-se o modelo de redes apresentado no Capítulo 3). Neste tipo de ambientes, nenhuma das técnicas anteriormente referidas pode fornecer um desempenho óptimo. Assim, importa desenvolver protocolos que tentem unificar os aspectos positivos das diversas aproximações até agora seguidas.

5.9.2 Trabalho relacionado

De entre os diversos algoritmos para concretizar ordem total, as aproximações tipo *passagem-de-testemunho* (Chang & Maxemchuck, 1984; Kaashoek & Tanenbaum, 1991) e tipo *simétrica* (Peterson *et al.*, 1989; Dolev *et al.*, 1993b) são as que têm tido maior utilização. Na aproximação tipo passagem-de-testemunho, um (ou mais) processo é responsável por ordenar as mensagens por delegação dos restantes processos no

sistema. Na aproximação simétrica, a ordenação é estabelecida por todos os processos de um modo descentralizado. Ambos os métodos têm vantagens e desvantagens.

Protocolos baseados na passagem de testemunho oferecem bom desempenho quando um único processo está a enviar mensagens. Neste caso, o remetente mantém o testemunho e ordena as mensagens à medida que as envia. No entanto, quando mais que um processo está a transmitir, a latência fica limitada pelo tempo necessário para circular o testemunho (ou para delegar a ordenação). Numa rede de grande escala os atrasos na rede não são desprezáveis (de facto, apesar de o débito ter vindo a aumentar significativamente com o advento de novas tecnologias, a latência não tem vindo a diminuir na mesma proporção). Infelizmente, a latência para as mensagens dos processos que não detêm o testemunho é pelo menos $2D$ (onde D é o atraso na rede), isto é, o tempo para disseminar a mensagem mais o tempo para obter o testemunho ou para delegar a ordenação. Deste modo, e apesar de ser particularmente apropriada ao uso em redes locais, a aproximação baseada na passagem de testemunho é ineficiente em face de redes com elevada latência.

Os protocolos simétricos possuem um conjunto de propriedades atraentes. São completamente descentralizados e dado que todos os processos são tratados de igual modo oferecem uma boa distribuição de carga. Deste modo, os protocolos simétricos oferecem bom desempenho quando todos os processos transmitem mensagens a uma taxa elevada. De facto, é possível com esta aproximação obter uma latência perto de $D + t$ (onde t é o maior intervalo de tempo entre mensagens consecutivas do sistema), independentemente da relação entre t e D. Infelizmente, os protocolos simétricos obrigam a que todos os processos enviem mensagens para garantir a entrega das mesmas; a latência destes protocolos será sempre uma função da taxa de envio do processo mais lento no sistema. Deste modo, os protocolos simétricos tendem a oferecer um baixo desempenho em ambientes onde a maioria dos processos tende transmite com taxas baixas (Mahlis *et al.*, 1992).

5.9.3 Protocolo híbrido dinâmico

Propõe-se um método híbrido para a concretização de ordem total em sistemas de grande escala. Neste método todos os processos difundem as mensagens directamente para os destinatários. No entanto, apenas alguns processos estão encarregues de ordenar as mensagens: estes processos designam-se por processos *activos*. Os processos activos emitem números de ordem, também chamados *estampilhas*, para as suas próprias mensagens e para os processos que operam em modo *passivo*. Num determinado momento, cada processo passivo está associado a um único processo activo (embora esta associação possa ser alterada no tempo). As estampilhas emitidas por diferentes processos activos são ordenadas de acordo com um protocolo simétrico. Deste modo, neste esquema híbrido, e com a finalidade de minimizar a latência, alguns processos operam usando uma aproximação simétrica e outros usando uma aproximação baseada em testemunho.

Quando todos os processos transmitem a uma taxa elevada o protocolo híbrido assemelha-se a um protocolo simétrico puro. Quando apenas um processo se encontra a transmitir, o protocolo híbrido assemelha-se a um protocolo de passagem de testemunho puro. Nas situações intermédias, os modos de operação são seleccionados com base nas características dos processos e da rede. Mostra-se que para topologias heterogéneas, nas quais os nós estão ligados por redes com diferentes propriedades e produzem mensagens a diferentes taxas, um protocolo híbrido oferece uma menor latência que qualquer um dos métodos anteriores isoladamente.

Finalmente, de modo a permitir adaptabilidade a alterações no débito dos clientes ou dos atrasos da rede, apresenta-se um algoritmo que permite aos nós mudarem de modo operacional em tempo de execução. Mostra-se que o protocolo híbrido dinâmico pode ser aplicado com sucesso em sistemas em que os padrões de tráfego e de atraso na rede não são conhecidos à partida. Os méritos desta aproximação são ilustrados com resultados obtidos através de simulação.

5.9.4 Hipóteses

O protocolo de ordem total está baseado na existência de um protocolo de difusão fiável não ordenado, de acordo com um modelo de sincronia virtual forte (veja-se a secção 2.3.1.2). Opta-se por um serviço de filiação linear, isto é, por um serviço que mantenha no máximo uma única partição activa no caso de falhas na rede, tal como o referido na secção 5.4. Oferece-se um serviço de ordenação total fraca (veja-se a secção 2.3).

De modo a medir o desempenho dos protocolos recorreu-se à simulação. Para isso, foi usado de novo o simulador NETSIM (Heybey, 1990). Os valores aqui apresentados foram obtidos fazendo uma média ponderada do tempo máximo de entrega das mensagens. Os parâmetros de simulação são descritos de seguida.

Assume-se que as mensagens são entregues pela camada de difusão fiável com uma latência que é função do atraso da rede entre o emissor e o destinatário. O atraso na rede $D_{(e,r)}$, entre o emissor e e um destinatário r, é representado por uma certa distribuição, com um valor médio de $\mu_{(e,r)}$, e uma variância $\sigma^2_{(e,r)}$. Deste modo, se um dado processo e emite uma mensagem no instante de tempo real t, o processo n irá recebê-la, em média, no instante $t + \mu_{(e,n)}$, o processo m, no instante $t + \mu_{(e,m)}$, etc. Um processo recebe as suas próprias mensagens instantaneamente. Geralmente, a distribuição depende do tipo de rede em utilização. Neste trabalho foram considerados dois tipos de rede: uma rede local, com pequenos valores de μ e σ^2 (na ordem de 20ms e 0.05, respectivamente) e uma rede de grande escala, com valores de μ e σ^2 na ordem de 500ms e 0.3-0.5, respectivamente. Uma distribuição do tipo qui-quadrado deslocado (*shifted chi-square*) foi usada para modelar os dois tipos de redes. Para além disso, assume-se que cada processo está sujeito a uma carga diferente. A carga de cada processo também é descrita por uma distribuição. Apresentam-se resultados obtidos com geradores de tráfego do tipo de *Poisson* e do tipo *Quase-Periódico*. No gerador de Poisson, o intervalo entre mensagens é descrito por uma variável aleatória com uma distribuição exponencial. No gerador Quase-Periódico, o intervalo entre mensagens é descrito por uma variável aleatória com uma distribuição normal de pequena variância.

5.9.5 Latência de protocolos simétricos

Na secção 5.9.2 enumeraram-se as vantagens de protocolos de ordenação total de características simétricas. Sendo completamente descentralizados, estes oferecem boa recuperação de faltas e repartição de carga. Para além disso, estes protocolos possuem o potencial para oferecer baixa latência na entrega das mensagens: uma vez que uma mensagem pode ser entregue assim que uma mensagem com uma estampilha superior for recebida de cada processo, se todos os processos estiverem a produzir mensagens com um ritmo elevado, as mensagens poderão estabilizar rapidamente. De facto pode-se ambicionar aproximar uma latência de $D + t$, onde D é o atraso na rede (o tempo necessário para disseminar a mensagem) e t o maior intervalo entre-mensagens de todas as fontes (o tempo necessário para a mensagem ficar estável).

5.9.5.1 O efeito de taxas diferentes

Para este trabalho foi simulada uma variante de um protocolo simétrico que é semelhante aos protocolos apresentados por Schneider (1990) ou por Ezhilchelvan *et al.* (1995). O protocolo opera do seguinte modo: cada processo possui um identificador único p_i (existe uma ordem total dos identificadores de processo) e mantém um relógio lógico de Lamport (1978) e_i que é actualizado nos eventos de envio e recepção. Todas as mensagens $\langle m \rangle$ são estampilhadas com o valor do relógio lógico do emissor no momento da transmissão. Cada relógio lógico é actualizado de acordo com as regras inicialmente propostas por Lamport. Antes de uma mensagem ser enviada, e_i é incrementado, e sempre que uma mensagem $\langle m \rangle$, com estampilha e_m é recebida, ao relógio lógico é atribuído o valor $e_i = max(e_i, e_m)$. Finalmente as mensagens são entregues pela ordem das suas estampilhas (e mensagens com a mesma estampilha são entregues pela ordem dos identificadores únicos dos seus emissores).

Ao experimentar este protocolo com diversos atrasos na rede e diversos intervalos entre-mensagens observa-se que, sempre que os processos exibem diferentes padrões de comunicação a latência aproxima-se mais de $2D$ do que $D + t$, mesmo nos casos em que t é muito menor que D. A Figura 5.18 mostra resultados de simulações usando

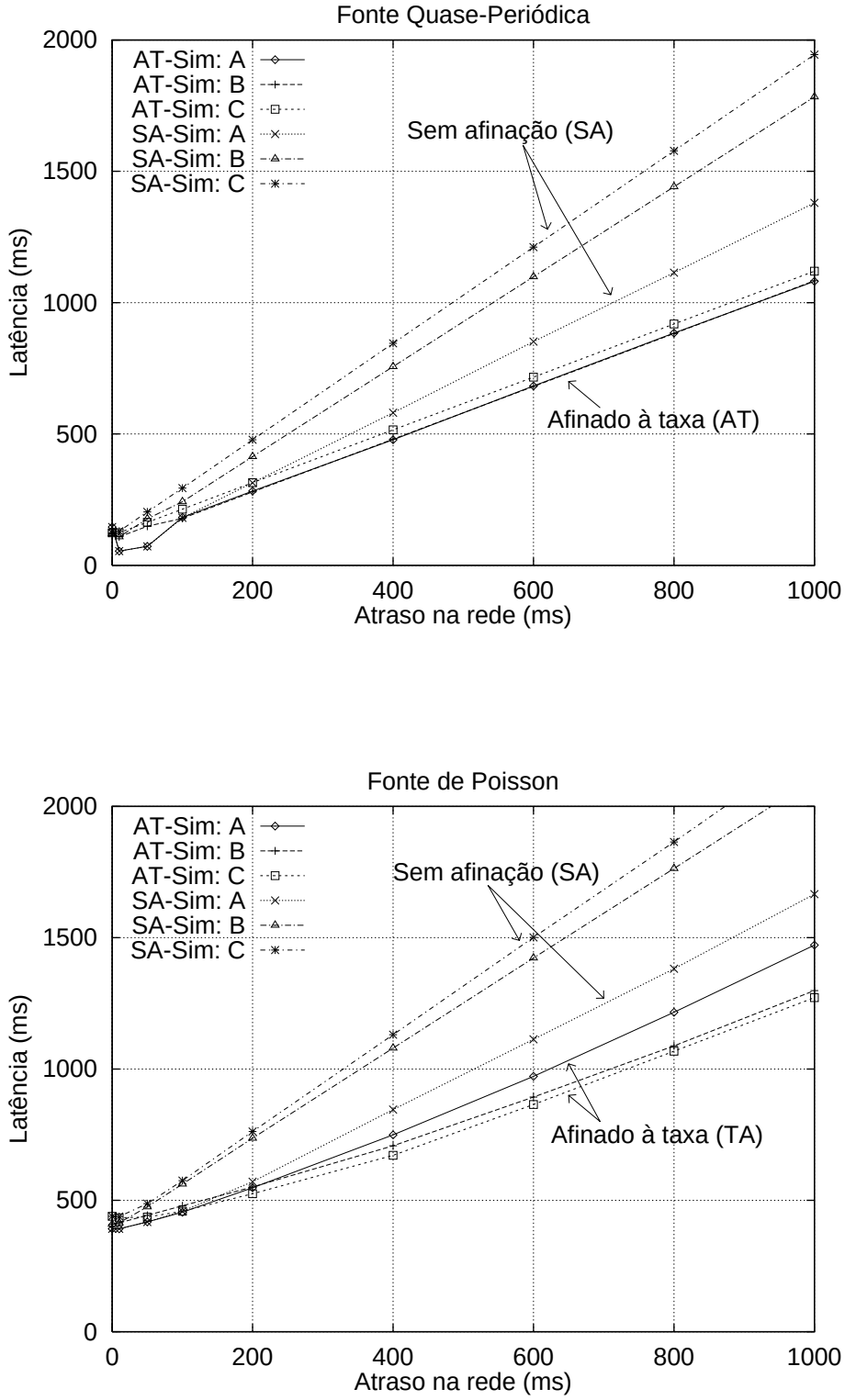


Figura 5.18: Protocolo simétrico com afinação-à-taxa

uma combinação de cinco processos ligados por uma rede local com atraso variável. Neste cenário um processo está sujeito a uma carga elevada e os restantes quatro processos estão sujeitos a uma carga baixa. A carga baixa utilizada foi de 5 mensagens/s (200ms de intervalo entre mensagens) e foi a mesma em todas as medições. A figura mostra resultados com diferentes valores para a carga elevada, respectivamente: 10 mensagens/s (linhas A), 50 mensagens/s (linhas B) e 100 mensagens/s (linhas C). Os resultados foram obtidos com diversos atrasos na rede, variando no eixo dos xs (a latência é representada no eixo dos ys). Cada linha representa o tempo máximo de entrega para diferentes tipos de fontes: neste caso Quase-Periódica e Poisson. O comportamento é ilustrado nas linhas marcadas como "sem afinação".

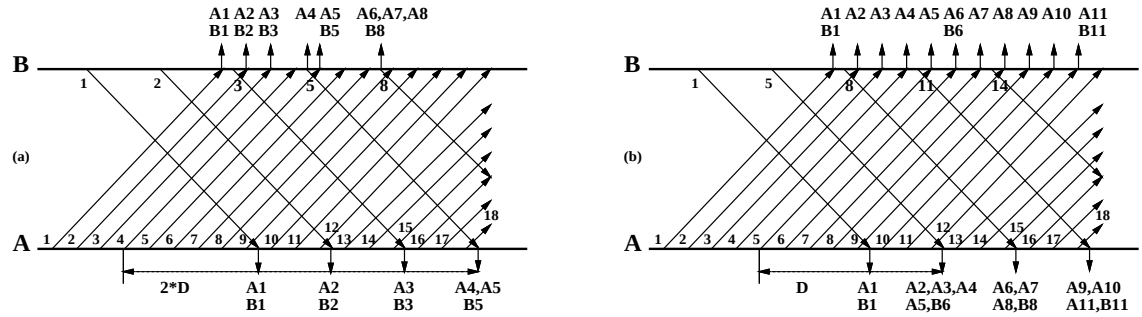


Figura 5.19: Afinação-à-taxa

Este comportamento, apesar de ser em certa medida contra-intuitivo, pode ser explicado. Se alguns processos produzem mensagens a um ritmo muito superior aos restantes, as estampilhas emitidas para as suas mensagens vão possuir valores muito superiores aos das estampilhas emitidas pelos processos mais lentos. Os processos mais lentos só actualizam os seus contadores quando recebem mensagens (D após a sua emissão) e os processos mais rápidos só recebem uma estampilha actualizada após um novo atraso de D, fazendo com que as suas mensagens se atrasem $2D$. Este comportamento está ilustrado na Figura 5.19a para um cenário com dois processos: o processo A produz mensagens a um ritmo superior ao processo B e vê a entrega das suas próprias mensagens constantemente atrasada.

Uma solução possível para este problema seria obrigar os processos mais lentos a enviarem mensagens de ressincronização (tal como é feito quando um processo se mantém inactivo por um longo período). No entanto, isto obrigaria ao consumo adicio-

nal de processador e da rede. Propõe-se um método inovador para reduzir a latência das mensagens baseado na sincronização da taxa a que as estampilhas são incrementadas, mesmo quando os processos não enviam mensagens ao mesmo ritmo. Isto é possível uma vez que as estampilhas, apesar de necessitarem de apresentar valores sempre crescentes, não precisam de exibir valores consecutivos (esta propriedade é usada, por exemplo, por Ezhilchelvan *et al.* (1995)). De modo a diminuir a latência, faz-se com que os processos produzindo mensagens a um ritmo mais lento incrementem o valor absoluto das suas estampilhas de modo a que este aumente aproximadamente à mesma taxa das estampilhas do processo mais rápido. A Figura 5.19b ilustra o comportamento ideal. Apesar do processo B produzir menos mensagens, as suas estampilhas avançam ao mesmo ritmo que as estampilhas do processo A. Tal como pode ser observado na figura, a maioria das mensagens estão sujeitas a atrasos menores do que no caso não sincronizado; por exemplo, o processo A entrega as suas próprias mensagens muito mais cedo que no caso não sincronizado. O exemplo da figura mostra claramente as vantagens de sincronizar as estampilhas no caso de emissores periódicos.

5.9.5.2 Afinação-à-taxa

De modo a fazer uso prático da observação anterior, e uma vez que geralmente não há conhecimento prévio da taxa a que cada processo vai produzir mensagens, é necessário desenvolver uma técnica que permita a sincronização das estampilhas em tempo de execução.

A técnica mais simples consiste em utilizar relógios sincronizados quando estes estão disponíveis. Utilizando o valor dos relógios para estampilhar a mensagem garante-se que as estampilhas permanecem suficientemente sincronizadas para optimizar a latência do protocolo. Para os sistemas em que relógios sincronizados não estão disponíveis, foi experimentada uma heurística simples baseada na avaliação dinâmica dos seguintes valores: a taxa a que os diferentes processos transmitem e os atrasos na rede entre os processos.

Cada processo estampilha as suas mensagens com o valor do seu relógio local no momento da transmissão. Com base nos valores destas estampilhas, os destinatários

podem estimar a taxa de transmissão no remetente. Os atrasos na rede são estimados ecoando algumas mensagens. Medindo no seu relógio local o intervalo entre a transmissão de uma mensagem e a recepção de uma mensagem de eco correspondente, um processo pode estimar o atraso na rede em relação ao processo que gerou a mensagem de eco (tipicamente, o atraso na rede será estimado pela rede abstracta (Frazão, 1995)). Dado que o protocolo simétrico pressupõe que todos os processos enviam mensagens frequentemente, rápidamente se estimam os valores das taxas de transmissão e dos atrasos na rede. Nas simulações aqui apresentadas usou-se um detector simples do tipo média-com-deslocamento: um valor inicial para o atraso na rede e para a taxa de transmissão é calculado usando um valor k pré-definido de amostragens[5]; sempre que um conjunto de k amostragens consecutivas cai acima ou abaixo da média esta é recalculada e o novo valor usado para uma nova iteração. Este e outros métodos para estimar este tipo de variáveis encontra-se descrito em pormenor por John (1990) e, como trabalho futuro, planeia-se experimentar outros detectores e avaliar o seu desempenho nesta aplicação.

Representa-se por X_j^i a avaliação no processo i do tempo médio entre mensagens no processo j. Representa-se também por $D_{(j,i)}^i$ a avaliação no processo i do atraso na rede entre j e i. Usando estes valores (que são actualizados sempre que é recebida uma mensagem, como descrito anteriormente), o processo i pode sincronizar as suas estampilhas com as estampilhas do processo mais rápido do seguinte modo: cada vez que recebe uma mensagem $\langle m \rangle$, com estampilha e_m, do processo f com taxa mais elevada ($\forall_{q \neq f} X_f^i < X_q^i$), atribui ao seu marcador de estampilha e_i o seguinte valor $e_i = max(e_i, e_m + D_{(f,i)}^i / X_f^i)$.

Naturalmente, o resultado desta heurística simples não mantém as estampilhas completamente sincronizadas. Os resultados dependem do número de amostras mantidas para realizar as médias e da distribuição dos geradores de tráfego. De modo a avaliar as potencialidades desta técnica, comparou-se uma versão de um protocolo simétrico com uma versão em que se aplicou a afinação-à-taxa das estampilhas. Os resultados estão também ilustrados na Figura 5.18. A figura apresenta valores de latência

[5]Nas simulações, usou-se $k = 7$.

para os protocolos com e sem afinação-à-taxa. As vantagens do método são evidentes. Por exemplo, os valores obtidos com uma fonte Quase-Periódica, seguem de perto a linha $D + t_{\text{taxa-baixa}}/2$ (e tendem para $D + t_{\text{taxa-baixa}}$ com uma fonte de Poisson).

Apresentou-se uma técnica de sincronização-à-taxa que reduz a latência de protocolos de ordenação simétricos. Enquanto a latência do protocolo não sincronizado pode atingir valores tão altos como $2D$, mesmo com intervalos entre transmissões muito mais pequenos que D, com a sincronização-à-taxa a latência diminui para valores próximos de $D + t_{\text{taxa-baixa}}$ (onde $t_{\text{taxa-baixa}}$ é o intervalo entre mensagens do processo mais lento). Isto significa que sempre que o intervalo entre mensagens seja menor que D, a latência de uma aproximação simétrica será menor do que a de um protocolo baseado em testemunho (a qual será pelo menos $2D$). Por outro lado, para valores de $t_{\text{taxa-baixa}} >> D$, a aproximação baseada em testemunho oferece uma melhor latência. Na próxima secção, mostra-se como este resultado pode ser usado para obter configurações híbridas, nas quais alguns processos operam de acordo uma aproximação simétrica e outros de acordo com uma aproximação baseada em testemunho.

5.9.6 Protocolo híbrido para topologias estáticas

Apresenta-se um esquema híbrido para topologias estáticas, ou seja, topologias onde os padrões de tráfego, taxas e atrasos na rede são conhecidos à partida e não são alterados no tempo. Mais tarde, estendem-se estes resultados para topologias dinâmicas. No esquema híbrido alguns processos operam de acordo com um protocolo baseado em testemunho (estes processos dizem-se *passivos*) e outros processos operam de acordo com um protocolo simétrico (estes processos dizem-se *activos*). Num determinado instante, cada processo passivo está associado com um único processo activo que emite estampilhas em seu nome.

O protocolo opera do seguinte modo. Cada processo possui um identificador único p_i e mantém um contador crescente c_i para as mensagens que emite. Deste modo, cada mensagem é identificada de modo único pelo par (p_i, c_i). As mensagens são difundidas directamente para todos os membros do grupo. Os processo activos mantêm

um contador adicional, o *contador de estampilha*, e_i. Os contadores de estampilha são actualizados tal como no protocolo simétrico descrito na secção 5.9.5, isto é, sempre que uma mensagem é recebida ou enviada. Uma estampilha é constituída por um trio $(p_i, e_i, (p_j, c_j))$ consistindo de: a identificação do processo activo p_i; um contador de estampilha e_i; e um identificador de mensagem (p_j, c_j). Um processo activo emite estampilhas para as suas próprias mensagens e para as mensagens dos processos a ele associados. Num determinado instante, um processo passivo está associado a um único processo activo, denominado o seu *sequenciador* (um processo activo pode ser sequenciador de vários processos passivos). De modo a serem disseminadas por todos os processos, as estampilhas são agregadas às mensagens enviadas pelos processos activos. As estampilhas são ordenadas como num protocolo simétrico (isto é, por ordem crescente dos seus valores, e estampilhas com o mesmo valor por ordem crescente dos identificadores dos processos que as emitiram). Finalmente, as mensagens são entregues pela ordem das suas estampilhas.

Considere-se o exemplo da Figura 5.20. O conjunto de processos A-G está organizado de acordo com a topologia ilustrada. Os processos B e E são designados como processos activos e, como tal, são encarregues de atribuir estampilhas às mensagens. Os restantes processos são designados passivos e associados aos processos activos: A e C são associados ao processo B; e D, F, e G são associados ao processo E. No exemplo, o processo A transmite uma mensagem ⟨`A/i` ⟩, o processo C uma mensagem ⟨`C/i` ⟩ (às quais o processo B atribui respectivamente, as estampilhas `B/1` e `B/2`), e o processo E duas mensagens ⟨`E/i` ⟩ e ⟨`E/ii` ⟩ (às quais atribui as estampilhas `E/1` e `E/2`). Estas mensagens são entregues por ordem das estampilhas, independentemente da sua origem, sendo as mensagens com estampilha de igual valor entregues por ordem da identificação do emissor da estampilha.

A parte crítica do algoritmo híbrido é a atribuição dos modos de operação aos processos. A decisão deve tomar em conta a taxa a que os processos produzem mensagens e os atrasos na rede. De modo a configurar o sistema, usou-se uma heurística que analisa pares de processos isoladamente. Considere um processo n, sujeito a uma carga caracterizada por um intervalo entre mensagens de t_n. Considere-se que o atraso na rede para o processo a mais próximo (em termos de atraso) é $D_{(n,a)}$. A condição

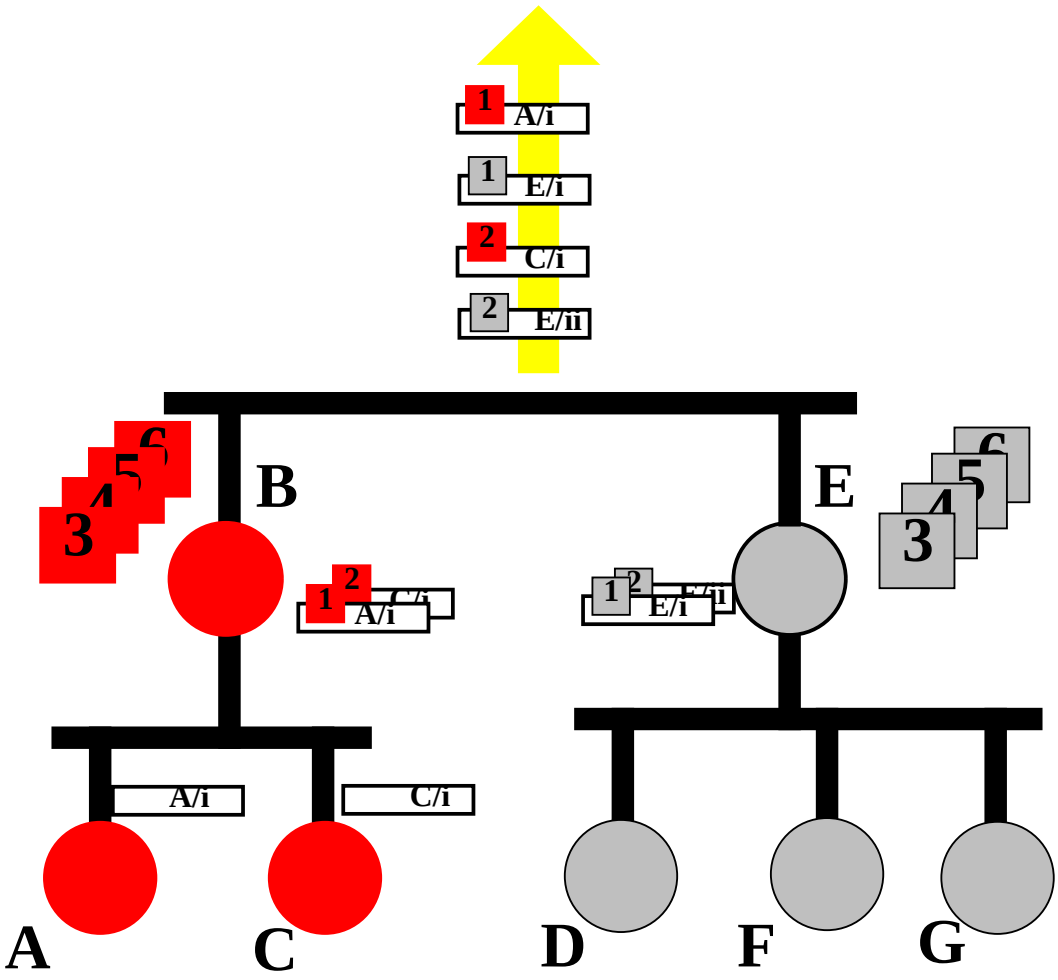


Figura 5.20: Protocolo híbrido.

```
para todos os processos n atribua a n o modo passivo
seja a o processo com a maior taxa; atribua-se a a o modo activo;
atribua-se o valor VERDADEIRO à variável mudou
enquanto (mudou = VERDADEIRO ) execute-se // ciclo
    in´ıcio
        atribua-se a mudou o valor FALSO
        para-todos j tal que o modo de j é passivo execute-se
            seja a o processo activo mais próximo de j
            se (D(j,a) + tj < 2D(j,a)) ent˜ao
                atribua-se a j o modo activo;
                atribua-se a mudou o valor VERDADEIRO
            caso-contr´ario
                atribua-se a j o sequenciador a
            fim-se ;
        fim-para-todos
    fim //enquanto
```

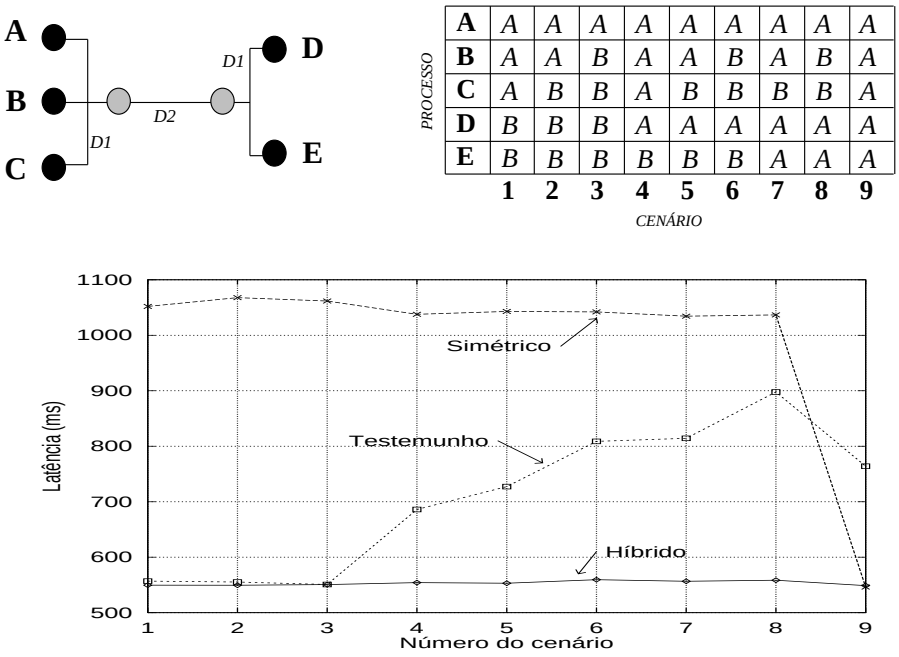
Figura 5.21: Heurística de atribuição de modo

que deve ser satisfeita para o processo n assumir o papel passivo é $D_{(n,a)} + t_n > 2D_{(n,a)}$ (ou seja $t_n > D_{(n,a)}$). Neste caso, o intervalo entre mensagens é maior que o dobro do atraso entre os dois processos, e p pode pedir e obter uma estampilha a partir de a antes que uma nova mensagem seja enviada. Por outro lado, se $D_{(n,a)} + t_n \leq 2D_{(n,a)}$ (ou seja, se $t_n \leq D_{(n,a)}$), uma vez que n envia mensagens com um intervalo inferior ao tempo necessário para obter uma estampilha, é preferível que o processo assuma um papel activo (isto não só diminui a latência como também melhora a distribuição da carga).

O algoritmo completo para atribuir os modos de operação aos processos pode ser obtido aplicando a regra anterior de modo recursivo, tal como descrito na Figura 5.21. Inicialmente, todos os processos são considerados passivos. Como pelo menos um processo activo deve existir no sistema, o processo (ou um dos processos[6], se mais que um for elegível) com o menor intervalo entre-mensagens é escolhido para assumir o papel activo. Após esta iniciação a regra é aplicada a todos os processos do sistema de modo a verificar se algum deve ser promovido a activo. O procedimento é repetido até que exista uma interacção em que nenhum processo muda de estado.

De modo a testar a eficácia desta aproximação comparou-se o desempenho do protocolo híbrido com o desempenho de um protocolo simétrico e com o desempenho de um protocolo baseado em testemunho. O algoritmo simétrico usado foi aquele

[6]Nesta heurística, escolhido aleatoriamente.



	1	2	3	4	5	6	7	8	9
A	*A*	*A*	*A*	*A*	*A*	*A*	*A*	*A*	*A*
B	*A*	*A*	*B*	*A*	*A*	*B*	*A*	*B*	*A*
C	*A*	*B*	*B*	*A*	*B*	*B*	*B*	*B*	*A*
D	*B*	*B*	*B*	*A*	*A*	*A*	*A*	*A*	*A*
E	*B*	*B*	*B*	*B*	*B*	*B*	*A*	*A*	*A*

Figura 5.22: Protocolo híbrido estático

descrito na secção 5.9.5. O algoritmo baseado em testemunho usado para comparação foi a versão não tolerante a faltas do protocolo de Kaashoek e Tanenbaum (1991), no qual apenas um processo emite estampilhas para todas as mensagens geradas no grupo.

Os resultados estão ilustrados na Figura 5.22. Os resultados foram obtidos num sistema constituído por cinco processos interligados por uma rede com a topologia também esquematizada na figura (nós cinzentos representam encaminhadores): os processos A, B e C (e os processos D e E) estão separados por uma distância de $D1$; no entanto, a distância entre um processo no primeiro grupo e um processo no segundo grupo (e vice-versa) é de $2D1 + D2$. Cada processo pode estar sujeito a dois tipos diferentes de carga, designados respectivamente por *alta* (A) e *baixa* (B). Foram simulados nove cenários distintos, diferindo na carga relativa de cada processo. Esta última está indicada na tabela da figura (por exemplo, no cenário 1, o processo A está sujeito a uma carga alta, o processo D está sujeito a uma carga baixa, etc). Foram usados os seguintes valores para os atrasos na rede e para os valores de carga: $D1 = 20\,ms$; $D2 = 500\,ms$; uma carga alta de $100\,mensagens/s$ e uma carga baixa de $1\,mensagens/s$ (ambos caracterizados por uma fonte Quase-Periódica). A Figura 5.22 apresenta a latência dos três protocolos para cada cenário (no caso do protocolo baseado em testemunho, o processo A mantém o testemunho em todos os cenários). No caso

do protocolo híbrido, os processos sujeitos a uma carga alta assumem um papel activo e processos sujeitos a uma carga baixa assumem o papel passivo. O protocolo baseado em testemunho exibe a menor latência quando todos os processos sujeitos a uma carga alta estão perto do nó que mantém o testemunho e vê o seu desempenho degradado quando a carga muda para os processos mais distantes. O protocolo simétrico oferece a menor latência no cenário 9, quando todos os processos estão sujeitos a uma carga elevada.

A linha quase plana representa a latência do protocolo híbrido. Nos casos limite, o protocolo híbrido assemelha-se a um protocolo baseado em testemunho ou, no outro extremo, a um protocolo simétrico. Para além disso, em todos os cenários intermédios para os quais o protocolo simétrico e o protocolo baseado em testemunho degradam o seu desempenho, o protocolo híbrido continua a exibir um desempenho perto do exibido nos casos mais favoráveis.

5.9.7 Protocolo de mudança de modo

Para aplicar o esquema híbrido a topologias dinâmicas, é necessário um algoritmo que permita a um processo mudar o seu modo de operação em tempo de execução. Esta secção descreve tal protocolo.

Existem três tipos de transições que podem ocorrer num protocolo híbrido dinâmico, nomeadamente: (i) um processo passivo pode mudar de sequenciador; (ii) um processo passivo pode mudar para activo; (iii) um processo activo pode mudar para passivo. As transições podem ocorrer devido a três tipos de razões: alterações na carga dos processos; alterações nos atrasos da rede; e falhas de processos. Quando um processo activo falha pode deixar diversos processos passivos sem sequenciador. Neste caso, os processos passivos afectados devem escolher um novo processo activo para seu sequenciador ou então tornarem-se activos. As transições devido a alterações na carga ou nos atrasos na rede acontecem para adaptar o sistema às novas condições operacionais.

Para garantir a operação correcta, todos os processos correctos no sistema devem

observar a mesma sequência de configurações. Deste modo, a ordem pela quais as transições são executadas, em relação ao fluxo de mensagens e de mudanças de vista, deve ser acordada antes da sua execução. De modo a chegar a acordo acerca da $(i+1)$-ésima configuração usam-se a propriedades da comunicação virtualmente síncrona e a ordem total das mensagens estabelecida pela $i$ésima configuração.

A vantagem de se usar comunicação virtualmente síncrona é a de, no caso de falhas, ter-se a garantia que todos os processos sobreviventes recebem o mesmo conjunto de mensagens antes da nova vista ser instalada. Isto significa que a entrega de uma nova vista é um ponto de sincronização no qual é possível tomar decisões coerentes. Em contrapartida, no protocolo de mudança de estado não se fazem hipótese acerca da coerência das avaliações dos atrasos na rede e das cargas nos processos. Deste modo, nenhum processo pode assumir que outro processo vai transitar de estado apenas por que essa transição parece plausível de acordo com as avaliações locais. Ou seja, todas as transições não despoletadas directamente por falhas devem ser iniciadas e disseminadas pelo processo que muda de estado.

De seguida, é feita uma descrição informal do protocolo, acompanhada por uma descrição em pseudo-código (para simplificar a descrição, omite-se o identificador de grupo nas chamadas ao serviço de filiação). Apresentam-se, em primeiro lugar, algumas definições (veja-se a Figura 5.23). Cada processo i é identificado por um trio denominado *descritor de processo*, denotado $D_i = (p_i, modo_i, nm_i)$, no qual p_i é o identificador do processo, $modo_i$ é o modo de operação (um de *activo* ou *passivo*), e nm_j é um *número de modo* (números de modo são iniciados a zero e incrementados sempre que um processo muda de modo). Define-se uma *configuração de sistema* $C = \{V^i, \bigcup_{j \in V^i} D_j\}$, como a concatenação de uma vista com a reunião dos descritores de todos os processos nessa vista. Assume-se também que cada processo i mantém um registo u_i da última das suas próprias mensagens a ser entregue. Para além disso, assume-se que cada processo passivo mantém o descritor do seu sequenciador numa variável denominada S_i. Finalmente, os processos mantêm aínda uma lista de mensagens pendentes P_i, uma lista não ordenada de estampilhas recebidas ENO_i, uma lista de estampilhas recebidas e ordenadas EO_i, e uma lista de estampilhas emitidas EE_i.

código para o processo i

```
Declaração de Tipos
   Modo um-de (activo, passivo);
   Estado um-de (activo, mudaseq, parapassivo, passivo, paraactivo);

Declaração de variáveis
   IdentificadorDeProcesso        p_i;
   Inteiro                        c_i;      // contador de mensagens
   Inteiro                        u_i;      // última entregue
   Modo                           modo_i;
   Inteiro                        nm_i;     // número de modo
   Estado                         estado_i;
   Inteiro                        e_i;      // contador de estampilha
   DescritorDeProcesso            S_i;      // o sequenciador de i
   Configuração                   C_i;      // a configuração actual
   FilaDeMsg                      P_i;      // mensagens pendentes
   ListaDeEstampilhas             ENO_i;    // lista de estampilhas não ordenadas
   ListaDeEstampilhas             EO_i;     // lista de estampilhas ordenadas
   ListaDeEstampilhas             EE_i;     // lista de estampilhas emitidas
   // E(M) denota a estampilha emitida para a mensagem M
```

Figura 5.23: Protocolo híbrido: iniciação

5.9.7.1 Configuração inicial

Quando o protocolo híbrido dinâmico se inicia, todos os processos devem concordar numa determinada configuração inicial. A configuração exacta não é muito relevante uma vez que o sistema tem a capacidade de se reconfigurar (desde que exista pelo menos um processo activo na configuração inicial). Em todas as simulações aqui apresentadas usou-se uma configuração inicial em que todos os processos se iniciam no estado activo e permanecem neste estado até receberem mensagens suficientes para avaliarem as taxas de transmissão e os atrasos na rede.

5.9.7.2 Operação em repouso

No protocolo híbrido dinâmico, um processo a operar em modo passivo não fica com um sequenciador atribuído de modo estático. Pelo contrário, um processo passivo pode instruir qualquer processo activo para ordenar as suas mensagens, e pode alterar estas instruções em tempo de execução (usualmente, um processo passivo só muda de sequenciador como resultado de uma alteração na configuração). Isto confere ao

```
quando  [estampilhas]⟨M⟩ recebido do processo j
execute-se
  início
    se ([estampilhas] ≠ ∅) então
      ENO_i := ENO_i ∪ [estampilhas];
      passar as estampilhas ordenadas de ENO_i para EO_i;
    fim-se;
    actualizar e_i usando a afinação-à-taxa;
    se (M = ⟨reatribuição, p_j, ]d, a], D_novo⟩) então
      para-todos  c_x ∈]d, a] execute-se
        Mc := ⟨qualquer-tipo, p_j, c_x, D_Mc, dígitos-binários⟩ ∈ P_i;
        alterar D_Mc para D_novo in Mc;
      fim-para-todos  ;
    caso-contrário
      P_i := P_i ∪ {⟨M⟩};
    fim-se;
    invocar entregarPorOrdem;
    se (estado_i = activo) então
      invocar emiteEstampilhas;
    fim-se  ;
fim ;

quando  existe uma mensagem M_utilizador para enviar e o estado_i = passivo
execute-se
  início
    c_i := c_i + 1;
    g_sv_difusão ([]⟨dados, p_i, c_i, S_i, M_utilizador⟩);
  fim ;

quando  existe uma mensagem M_utilizador para enviar e o estado_i = activo
execute-se
  início
    c_i := c_i + 1;
    M := ⟨dados, p_i, c_i, D_i, M_utilizador⟩;
    emite estampilha E(M) para M; //(actualiza e_i)
    EE_i := EE_i ∪ {E(M)};
    gs_sv_difusão ([EE_i]⟨M⟩);
    EE_i := ∅;
  fim ;
```

Figura 5.24: Protocolo híbrido: recepção e transmissão de mensagens.

sistema uma maior flexibilidade e permite uma maior adaptabilidade a alterações nas condições de operação. Para suportar esta associação entre o processo passivo e o seu sequenciador, as mensagens de dados são encapsuladas numa mensagem protocolar com o seguinte formato: $\langle$*dados*, p_i, c_i, S_i, *dados-utilizador*$\rangle$, onde "*dados*" indica o tipo da mensagem; p_i é a identificação do emissor, c_i o número de sequência da mensagem e S_i o descritor do processo designado como sequenciador para essa mensagem. O sequenciador designado irá emitir uma estampilha para a mensagem desde que o seu número de modo seja aquele especificado no descritor S_i incluído na mensagem (veja-se a Figura 5.24).

Uma vez que as mensagens podem ser transmitidas concorrentemente com a ocorrência de eventos que podem despoletar mudanças na configuração, é possível que o sequenciador designado para uma mensagem falhe ou incremente o seu número de modo antes de receber a mensagem. Para cobrir estes casos o protocolo usa outra mensagem reservada, chamada uma mensagem de *reatribuição*, com o formato seguinte: $\langle$*reatribuição*, $p_i,]u_i, c_i], S_{\text{novo}}\rangle$, onde p_i é a identificação do remetente, $]u_i, c_i]$ um intervalo de números de sequência de mensagens (correspondente a mensagens ainda não ordenadas) e S_{novo} o novo sequenciador para as mensagens no intervalo especificado. Como se irá ver, mensagens de reatribuição são apenas enviadas quando o sequenciador inicial falha ou transita para passivo (Figuras 5.25 e 5.27).

Se um processo passivo falha, todas as suas mensagens por si enviadas e entregues pelo serviço virtualmente síncrono antes da entrega da nova vista, mas ainda não ordenadas pelo sequenciador, são descartadas por todos os processos. Este procedimento é seguro dado que as propriedades da comunicação virtualmente síncrona garantem que as estampilhas emitidas para essas mensagens são também ordenadas em relação à instalação da nova vista.

Tanto os processos passivos como os processos activos armazenam as mensagens recebidas numa *fila de mensagens pendentes*. Os processos activos emitem estampilhas para as mensagens a eles atribuídas (ou seja, sempre que o descritor de processo especificado na mensagem é igual ao seu próprio descritor de processo). As estampilhas vão sendo ordenadas à medida que são recebidas (agregadas nas mensagens emitidas

pelos processos activos). Finalmente, as mensagens para as quais já foi recebida uma estampilha são removidas da lista de mensagens pendentes e entregues pela ordem das estampilhas respectivas. As estampilhas são incrementadas de acordo com a técnica de sincronização à taxa descrita na secção 5.9.5. Apesar de apenas os processos activos emitirem estampilhas, todos os processos (incluindo os passivos) mantêm os números usados para gerar estampilhas sincronizadas: um processo passivo pode necessitar de se tornar activo e o protocolo irá apresentar um melhor desempenho se estes valores se encontrarem já actualizados.

Algumas mensagens têm uma utilização reservada no protocolo e não são entregues ao utilizador. A utilização destas mensagens será clarificada adiante. Em particular, a mensagem $\langle$*reatribuição*$, p_i,]u_i, c_i], S_i\rangle$ nunca é entregue: é usada para actualizar o campo referente ao sequenciador das mensagens especificadas (que se devem encontrar na fila de mensagens pendentes).

5.9.7.3 Eventos que despoletam reconfigurações

Transições no modo de operação dos processos (Figura 5.26) e a ocorrência de falhas (Figura 5.27) introduzem uma sequência de configurações de sistema. Com a finalidade de alterar o seu modo de operação, os processos difundem mensagens reservadas, nomeadamente mensagens do tipo $\langle paraActivo, p_i, c_i, S_i\rangle$ e $\langle paraPassivo, p_i, c_i, S_i\rangle$. Estas mensagens são enviadas em ordem total tal como as mensagens de dados. A entrega destas mensagens despoleta a instalação de uma nova configuração de sistema. Mais precisamente, existem três tipos de eventos que podem alterar a configuração do sistema:

Mudança de vista. Assuma-se que o sistema está na configuração $C^n = \{V^i, \bigcup_{j \in V^i} D_j\}$ quando a vista V^{i+1} é entregue pela camada de comunicação virtualmente síncrona (Figura 5.27). Uma nova configuração $C^{n+1} = \{V^{i+1}, \bigcup_{j \in V^{i+1}} D_j\}$ é criada.

Se não existe nenhum processo activo nessa configuração (isto é, todos os processos activos falharam) o processo m com o identificador mais alto na vista

```
declara  função emiteEstampilhas
  para-todas  mensagem M = ⟨qualquer-tipo, p_j, c_j, D_M, dígitos-binários⟩ ∈ P_i execute-se
    se (D_M = (p_i, modo_i, nm_i) ∧ E(M) ∉ EE_i) então
      emite uma estampilha E(M) para M; // (actualizar e_i)
      EE_i := EE_i ∪ {E(M)};
    fim-se;

declara  função entregarMensagem (Msg M)
  se (M = ⟨dados, p_j, c_j, S, M_utilizador⟩)) então // mensagem de dados
    entregar M_utilizador;
    se (p_j = p_i) então
      u_i := c_j;
      se (estado_i = mudaseq ∧ u_i = c_i) então
        S_i := seleccionaNovoSequenciador();
        estado_i := passivo; // recomeçar a enviar
      fim-se;
    fim-se;
  fim-se
  se (M = ⟨paraActivo, p_j, c_j, S⟩) então
    C_i := nova configuração;
    se (p_j = p_i) então
      nm_i := nm_i + 1; u_i := c_j;
      estado_i := activo; S_i := (p_i, modo_i, nm_i);
      invocar emiteEstampilhas; // recomeça a enviar
    fim-se;
  fim-se;
  se (M = ⟨paraPassivo, p_j, c_j, S⟩) então
    C^temp := nova configuração;
    se (não existe um processo activo em C^temp) então // abortar
      se (p_j = p_i) então
        u_i := c_j; estado_i := activo; S_i := (p_i, modo_i, nm_i);
        invocar emiteEstampilhas; // recomeça a enviar
      fim-se ;
    caso-contrário
      C_i := C^temp;
      se (p_j = p_i) então
        nm_i := nm_i + 1; u_i := c_j; estado_i := passivo; // recomeça a enviar
      fim-se;
      se (estado_i ≠ activo) então
        u_i := c_j;
        atribuir a S_i o Descritor do processo activo mais próximo;
        g_sv_difusão([]⟨reatribuição, p_i, ]u_i, c_i], S_i⟩);
      fim-se;
    fim-se;
  fim-se;

declara  função entregarPorOrdem
  enquanto  (E(M) = primeiroDaFila (EO_i) ∧ M ∈ P_i) execute-se
    início
      remover E(M) de EO_i; remover M de P_i;
      invocar entregarMensagem (M);
    fim;
```

Figura 5.25: Protocolo híbrido: entrega de mensagens

```
quando  estado_i = activo e é altura de transitar para passivo
execute-se
   início
      c_i := c_i + 1;
      M := ⟨paraPassivo, p_i, c_i, S_i⟩;
      emite estampilha E(M) para M;// (actualizar e_i)
      EE_i := EE_i ∪ {E(M)};
      estado_i := parapassivo; // pára de emitir
      g_sv_difusão ([EE_i]⟨M⟩);
      EE_i := ∅;
   fim ;

quando  estado_i = passivo e é tempo para mudar de sequenciador
execute-se
   estado_i := mudaseq; // pára de emitir

quando  estado_i = passivo e é tempo de transitar para activo
execute-se
   início
      c_i := c_i + 1;
      M := ⟨paraActivo, p_i, c_i, S_i⟩;
      estado_i := paraactivo; // pára de emitir
      g_sv_difusão ([]⟨M⟩);
   fim ;
```

Figura 5.26: Protocolo híbrido: transição voluntária.

V^{i+1} transita automaticamente para modo activo, atribuindo a $modo_m$ = *activo*, e incrementando nm_m. Então, a nova configuração C^{n+1} é instalada.

Caso exista um processo passivo i tal que $S_i \notin C^{n+1}$, este processo escolhe um novo sequenciador m e atribui $S_i = (p_m, modo_m, nm_m)$. Para além disso, o processo envia a seguinte mensagem de reatribuição: $\langle$*reatribuição*$, p_i,]u_i, c_i], S_i\rangle$.

Entrega de uma mensagem $\langle paraActivo, p_j, c_j, S\rangle$**.** Esta mensagem é enviada por um processo passivo quando deseja tornar-se activo e é entregue em ordem total, tal como uma mensagem de dados. Assuma-se que o sistema está na configuração C^n quando esta mensagem é entregue (Figura 5.25). Uma nova configuração C^{n+1} é criada atribuindo-se $modo_j$ = *activo*, e incrementando nm_j. Então, a configuração C^{n+1} é instalada.

Entrega de uma mensagem $\langle paraPassivo, p_j, c_j, S_j\rangle$**.** Esta mensagem é enviada por um processo activo quando ele deseja tornar-se passivo e é entregue em ordem total, tal como uma mensagem de dados. Assuma-se que o sistema está na configuração

```
quando  vista V^i é entregue execute-se
  início
    C^temp := V^i, ⋃_{j∈V^i} D_j;
    se (S_i ∉ C^temp ∧ ∃_{j∈C^temp} : modo_j = activo) então
      S_i := descritor do processo activo mais próximo;
      g_sv_difusão([]⟨reatribuição, p_i, ]u_i, c_i], S_i⟩);
    fim;
    se (∄_{j∈C^temp} : modo_j = activo) então
      seleccionar processo activo m;
      modo_m := activo;
      nm_m := nm_m + 1; // actualizar C^temp
      se (m = i) então
        // Sou o meu próprio sequenciador agora
        estado_i := activo;
        S_i := (p_i, modo_i, nm_i);
        g_sv_difusão ([]⟨reatribuição, p_i, ]u_i, c_i], S_i⟩)
        invocar emiteEstampilhas;
    fim-se ;
    C_i := C^temp;
  fim;
```

Figura 5.27: Protocolo híbrido: recepção de vista

C^n quando esta mensagem é entregue (Figura 5.25). Caso p_j seja o único processo activo na configuração, a mensagem é descartada e não é instalada uma nova configuração uma vez que pelo menos um processo activo deve sempre existir em qualquer configuração. Caso contrário, uma nova configuração C^{n+1} é criada atribuindo-se $modo_j$ = *passivo*, e incrementando nm_j. Então a configuração C^{n+1} é instalada.

Se existe um processo passivo i tal que $S_i \notin C^{n+1}$, este processo selecciona um novo sequenciador m e atribui $S_i = (p_m, modo_m, nm_m)$. Para além disso, o processo passivo envia a seguinte mensagem de reatribuição: $\langle$*reatribuição*$, p_i,]u_i, c_i], S_i\rangle$.

5.9.7.4 Mudança de sequenciador

Um processo passivo pode também mudar de sequenciador se um outro processo mais próximo que estava passivo transita para activo. Uma vez que as mensagens de dados especificam o sequenciador desejado, trocar de sequenciador é bastante simples.

De modo a evitar violações na ordem FIFO, o processo passivo i pára temporariamente de transmitir mensagens (Figura 5.26) e espera que todas as mensagens enviadas anteriormente sejam entregues (ou seja, espera até que $u_i = c_i$). Então atribui à variável que mantém a identificação do seu sequenciador S_i o valor correspondente ao sequenciador desejado e recomeça a transmitir mensagens (Figura 5.25).

Um processo passivo i também pode mudar de sequenciador no caso do seu sequenciador anterior ter transitado para o estado passivo (Figura 5.25). Tal como no caso anterior, o processo passivo actualiza o valor da variável correspondente ao seu sequenciador S_i com o descritor de processo do novo sequenciador. Para além disso, sabendo qual foi a última mensagem u_i ordenada pelo sequenciador anterior, ele envia uma mensagem de reatribuição com o formato $\langle \textit{reatribuição}, p_i,]u_i, c_i], S_i \rangle$ instruindo o novo sequenciador para emitir estampilhas para as mensagens ainda não ordenadas.

5.9.7.5 Transição de activo para passivo

No protocolo híbrido dinâmico, existem duas razões para um processo transitar de activo para passivo: (a) a sua carga diminuiu para uma taxa em que é mais vantajoso pedir a emissão de estampilhas a outro processo; (b) um nó próximo tornou-se activo e está a transmitir a uma taxa muito mais elevada, sendo também mais vantajoso tornar-se passivo, usando esse nó como sequenciador. De modo a transitar para modo passivo (Figura 5.26), o processo p_i envia uma mensagem especial $\langle \textit{paraPassivo}, p_i, c_i, S_i \rangle$ e pára de transmitir mensagens (parando também de emitir estampilhas, mesmo para mensagens a ele atribuídas). Então, espera pela entrega da sua própria mensagem. Ao entregar a sua mensagem $\langle \textit{paraPassivo} \rangle$ ele verifica se não é o último processo activo no grupo (Figura 5.25). Note-se que diversos processos activos podem decidir mudar para passivo simultaneamente mas, como pelo menos um processo deve permanecer activo, o último pode necessitar de abortar a transição. Se a transição não necessitar de ser abortada, a nova configuração é instalada e o processo recomeça a enviar mensagens, agora em modo passivo.

5.9.7.6 Transição de passivo para activo

A transição de modo passivo para modo activo pode ocorrer devido ao facto de o processo ficar sujeito a uma maior carga ou porque todos os processos activos da configuração anterior falharam.

No primeiro caso, o processo passivo i difunde uma mensagem especial $\langle \textit{paraActivo}, p_i, c_i, S_i \rangle$ e pára a transmissão de mensagens (Figura 5.26). Então, espera que esta mensagem seja ordenada pelo seu sequenciador e entregue (Figura 5.25). Quando esta mensagem é entregue, uma nova configuração é instalada, tal como descrito na secção 5.9.7.3. Todas as mensagens enviadas após esta nova configuração ser instalada são ordenadas pelo próprio processo i (lembra-se que apesar de apenas os processos activos emitirem estampilhas, todos os processos mantêm os números de ordem sincronizados por afinação-à-taxa).

Em caso de falha do único processo activo, o processo passivo torna-se activo mal recebe a indicação de falha da camada de comunicação em grupo (Figura 5.27). Após esta transição, o novo processo activo i emite estampilhas para todas as mensagens por si enviadas mas não ordenadas por configurações anteriores, ou seja, para todas as mensagens com números de sequência no intervalo $]u_i, c_i]$.

5.9.8 Protocolo híbrido para topologias dinâmicas

Os padrões de tráfego variam com o tempo na maioria das aplicações interactivas. Os componentes reagem habitualmente a eventos que os levam a alternar períodos de baixa actividade com períodos de alta actividade. Os atrasos na rede estão também sujeitos a variações, devido a variações na carga (períodos nocturno e diurno, por exemplo) ou a falhas na rede (percursos mais rápidos podem ficar temporariamente indisponíveis). Na secção anterior, foi descrito um algoritmo que permite a um processo alterar o seu modo de operação em tempo de execução. Deste modo, o protocolo pode adaptar-se a alterações no seu envelope operacional, isto é, a variações na carga ou nos atrasos da rede.

Ao se introduzir a afinação-à-taxa em tempo de execução (secção 5.9.5.2), foram

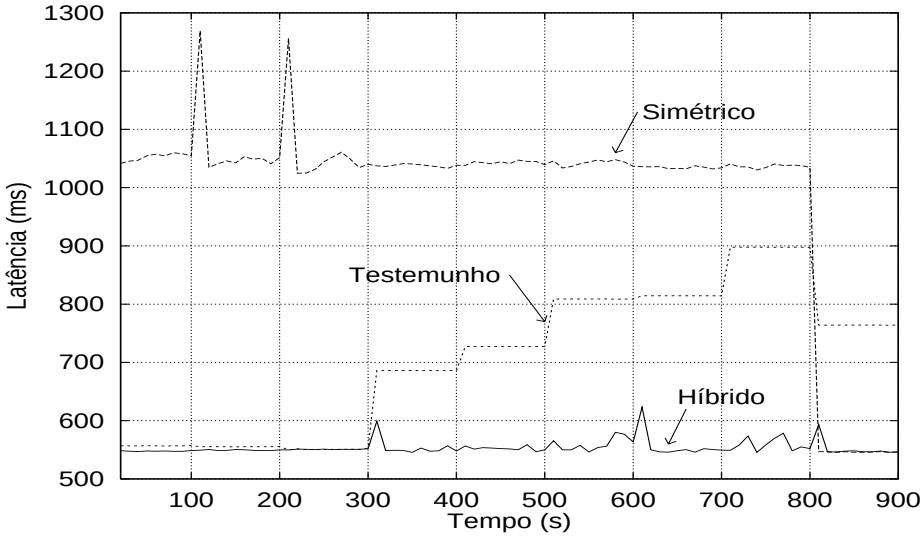


Figura 5.28: Protocolo híbrido dinâmico

descritas técnicas que permitem a um processo avaliar parâmetros de operação do sistema como a taxa de transmissão ou os atrasos na rede. Na secção 5.9.6 foi descrito o modo como um observador externo pode usar esta informação para atribuir modos de operação aos processos de modo a criar uma configuração híbrida. Uma vez que este observador externo só pode ser aproximado, e de modo a evitar soluções centralizadas, apresenta-se uma heurística que permite, a cada processo, tomar decisões locais acerca do seu modo de operação, com base na sua própria avaliação do estado do sistema.

A heurística é a seguinte: cada processo monitoriza a sua taxa de transmissão e o atraso da rede em relação aos restantes nós do sistema. Se o intervalo entre mensagens for menor que o atraso em relação ao processo mais próximo a operar em modo activo, ele deve mudar para modo activo. Por outro lado, se o intervalo entre mensagens for maior que o atraso em relação ao processo mais próximo a operar em modo activo, ele deve mudar para modo passivo, escolhendo o processo activo mais próximo como o seu sequenciador. De modo a evitar trocas muito frequentes sempre que o intervalo entre mensagens tem um valor aproximado ao dos atrasos na rede, a decisão de mudar o modo de operação só é tomada quando a diferença entre estes valores excede um determinado limite (foi usado um limite correspondente a 20% do atraso na rede).

Os resultados obtidos com esta heurística são apresentados na Figura 5.28, juntamente com resultados para os protocolos simétricos e baseados em testemunho. Por

clareza de exposição, usaram-se exactamente os mesmos cenários da Figura 5.22, embora agora o sistema tenha permanecido em funcionamento enquanto a carga dos processos variava no tempo. Deste modo, o sistema progrediu através dos nove cenários em sequência. No protocolo híbrido, sempre que a carga dos processos é alterada, o modo de operação é reavaliado e, se necessário, o algoritmo de transição executado. É possível verificar que, cada vez que existe uma alteração no envelope operacional, existe um aumento temporário na latência. Isto deve-se ao tempo necessário para tomar as decisões locais e à perturbação introduzida pelo protocolo de mudança de estado. Apesar disso, verifica-se que o protocolo híbrido apresenta um desempenho significativamente melhor de que os dois outros protocolos isoladamente, exibindo uma latência quase constante, independentemente da taxa de transmissão dos processos.

Os conceitos principais do protocolo híbrido foram ilustrados com exemplos relativamente simples. Num relatório, apresentam-se resultados obtidos com configurações mais elaboradas (Rodrigues *et al.*, 1994). Estes resultados confirmam o que aqui foi mostrado: o protocolo híbrido oferece vantagens significativas em relação aos protocolos simétricos e baseados em testemunho para redes de grande escala a ambientes heterogéneos.

5.9.9 Protocolo total/causal

O protocolo aqui descrito pode ser combinado com o protocolo de ordenação causal para fornecer um serviço que entrega as mensagens por ordem total e respeitando as relações de causalidade. As técnicas para fornecer este serviço são bem conhecidas (veja-se, por exemplo, Stephenson (1991)): as mensagens de dados devem ser enviadas tal como uma mensagem causal mas, após a sua recepção, só são entregues depois de terem sido ordenadas por ambos os protocolos. Por razões de espaço, omite-se uma descrição pormenorizada do protocolo combinado. O serviço de ordenação total para redes de grande escala da arquitectura NAVTECH , cuja interface se resume na Tabela 5.8, respeita a ordem causal.

Nome	Tipo	Parâmetros
g_c_atómica	pedido	Grupo g, IdProcLista destinatários, Msg m

Tabela 5.8: Interface do serviço de ordenação total

5.9.10 Observações

Nos parágrafos anteriores, foi proposto um novo protocolo híbrido para oferecer ordem total em sistemas de grande escala, usando uma combinação de protocolos baseados em testemunho e protocolos simétricos. Este protocolo é capaz de se adaptar dinamicamente a alterações no débito dos clientes ou no atraso da rede. O protocolo é melhorado pela utilização de uma política de afinação-à-taxa que, por si só, oferece grandes ganhos na latência em diversas configurações.

O protocolo híbrido foi simulado em diversas configurações, usando diferentes topologias de rede e padrões de tráfego. Os resultados mostraram que pode fornecer vantagens significativas no que diz respeito a latência. Usando heurísticas simples, é possível fazer as decisões de alteração do modo locais a cada processo. Será interessante estudar heurísticas alternativas, para o algoritmo de atribuição de modo, para a afinação-à-taxa, e para a política de mudança de modo, para verificar se é possível melhorar o desempenho. Tenciona-se experimentar a afinação-à-taxa com o protocolo de entrega antecipada descrito por Dolev *et al.* (1993b).

5.10 Interface do serviço NAVTECH para grande escala

A Tabela 5.9 resume parte da interface dos serviços NAVTECH para grande escala. A tabela não pretende representar a interface completa, uma vez que esta inclui primitivas de acesso a muitos dos serviços não cobertos pela tese, tal como os serviços de rede abstracta, os serviços de filiação, etc. Pretende-se simplesmente agregar num único local a interface dos serviços descritos neste capítulo, conjuntamente com a interface dos serviços de filiação utilizados, de modo a facilitar a sua referência. Para além disso, esta interface será a utilizada no próximo capítulo para concretizar um serviço

Nome	Tipo	Parâmetros
Comunicação		
g_c_transparente	pedido	Grupo g, IdProcLista destinários, Msg m
g_c_retida	pedido	Grupo g, IdProcLista dest
g_c_causal	pedido	Grupo g, IdProcLista destinatários, Msg m
g_c_uniforme	pedido	Grupo g, IdProcLista destinatários, Msg m
g_c_atómica	pedido	Grupo g, IdProcLista destinatários, Msg m
g_c_entrega	indicação	Grupo g, IdProc rem, Msg m, IdProcLista lib, IdProcLista perd
g_c_liberta	indicação	Grupo g, IdProcLista necessárias
Filiação		
g_f_entra	pedido	Grupo g
g_f_sai	pedido	Grupo g
g_f_vista	indicação	Grupo g, IdProcLista vista
Sincronía virtual		
g_sv_difusão	pedido	Grupo g, IdProcLista destinatários, Msg m
g_sv_uniforme	pedido	Grupo g, IdProcLista destinatários, Msg m
g_sv_entrega	indicação	Grupo g, IdProc remetente, Msg m
Rede abstracta		
g_ra_mesforço	pedido	Grupo g, IdProcLista destinatários, Msg m
g_ra_entrega	pedido	Grupo g, IdProc remetente, Msg m

Tabela 5.9: Interface do NAVTECH para grande escala

de invocação remota.

5.11 Camada de integração

Os componentes para grande escala da arquitectura NAVTECH são naturalmente também utilizáveis sobre rede local. Sendo assim, poder-se-ia supor que estes substituem completamente os componentes apresentados no Capítulo 4. No entanto, e apesar de o protótipo da rede abstracta para grande escala ainda estar em fase de concretização, e como tal não haver resultados práticos do seu desempenho, julga-se que será difícil a estes componentes competir com protocolos especialmente concebidos para este tipo de redes (como o AMp (Veríssimo *et al.*, 1989)/xAMp (Rodrigues & Veríssimo, 1992b) ou o Totem (Amir *et al.*, 1993)).

Deste modo, a arquitectura NAVTECH prevê a coexistência dos dois tipos de protocolos. Isto permite atingir dois objectivos:

- É possível obter duas variantes da arquitectura, uma mais genérica baseada nos componentes para rede de grande escala, e outra com maior desempenho dedicada à operação em rede local.
- É possível obter uma variante hierárquica do NAVTECH que usa ambos os tipos de componentes de um modo integrado, utilizando os componentes orientados à rede de grande escala para interligar várias organizações, e utilizando os componentes orientados à rede local em redes internas às organizações.

Como se referiu no Capítulo 3, a ligação entre os dois tipos de componentes é realizada por uma *camada de integração*. Esta camada ainda não foi desenvolvida. No entanto, os protocolos orientados à rede de grande escala estão a ser concebidos tendo em vista esta futura integração, pelo que esta tarefa não se afigura excessivamente complexa. A título de exemplo, considere-se o protocolo de ordem total. Amir *et al.* (1992) mostram que protocolos de ordem total podem ser combinados de modo hierárquico. Uma solução optimizada para redes de grande escala constituídas por redes locais interligadas pode usar esta técnica para combinar o protocolo híbrido descrito neste capítulo com o protocolo atómico descrito no capítulo anterior.

5.12 Acerca da correcção dos protocolos e sua concretização

Idealmente, tanto a verificação formal como a análise de desempenho estariam integradas no processo de desenvolvimento de protocolos. De preferência, uma mesma linguagem de especificação deveria permitir obter informações acerca do desempenho e correcção do protocolo e, posteriormente, obter um executável eficiente (Ghezzi *et al.*, 1991). Infelizmente, este processo ainda não é possível, pelo menos para protocolos de elevada complexidade como os aqui apresentados (Baptista, 1991).

De acordo com a experiência obtida no desenvolvimento do xAMp, decidiu-se validar primeiro os algoritmos através de simulações, posteriormente transportar o código para um ambiente de execução e, apenas depois de obter uma versão estável

dos algoritmos, realizar uma verificação formal dos mesmos. Até este momento, apenas o primeiro passo foi realizado e, ao longo deste capítulo, foram apresentados diversos resultados de simulações efectuadas. Os restantes passos constituirão trabalho futuro.

5.13 Discussão

A baixa fiabilidade e a susceptibilidade a partições, características das rede de grande escala, tornam a concretização de serviços de filiação e comunicação em grupo mais complexa do que sobre redes locais. Para além disso, a elevada latência constitui um factor limitativo no desempenho dos protocolos, aconselhando o uso de serviços que exijam poucos turnos de troca de mensagens.

Por estas razões, importa disponibilizar qualidades de serviço enfraquecidas, que possam ser concretizadas de modo eficiente, e qualidades de serviço que permitam à aplicação ter um maior controlo sobre o momento de disseminar a informação. Os serviços de mensagens transparentes e mensagens retidas foram desenvolvidos tendo estes objectivos em mente.

Quando qualidades de serviço mais fortes são necessárias, e à semelhança do que foi feito para redes locais, devem tornar-se visíveis as propriedades da rede que permitem melhorar o desempenho dos protocolos. Apresentaram-se dois exemplos desta estratégia: o conhecimento acerca da topologia da rede permite diminuir a quantidade de informação que necessita de ser trocada para garantir a causalidade; o conhecimento acerca dos atrasos na rede pode ser usado para configurar protocolos de ordem total.

Os algoritmos apresentados foram concebidos para operar sobre um serviço de filiação linear. Esta opção justifica-se pela necessidade de adquirir experiência, desenvolvendo primeiro um serviço que é não só de mais fácil utilização (e por isso, de grande aceitação (Birman *et al.*, 1991b)) mas também mais simples do ponto de vista algorítmico. No futuro, serão desenvolvidas versões dos algoritmos para um serviço de filiação parcial.

5.14 Sumário

Neste capítulo apresentaram-se alguns dos componentes da arquitectura NAVTECH desenvolvidos para operarem sobre redes de grande escala. Nomeadamente, apresentou-se um serviço de ordenação causal e um serviço de ordenação total. Adicionalmente, propuseram-se duas novas qualidades de serviço, nomeadamente um serviço de mensagens transparentes e um serviço de mensagens retidas. No próximo capítulo apresenta-se um protocolo de invocação remota que faz uso dos serviços aqui apresentados.

Notas

O uso de separadores causais foi sugerido pelo autor em "Causal Separators for Large-Scale Multicast Communication", Rodrigues e Veríssimo, Actas da 15ª "IEEE International Conference on Distributed Computing Systems", Vancouver, British Columbia, Canadá, Maio, 1995. Os resultados das simulações apresentados nessa secção foram obtidos com a preciosa colaboração do Eng. Henrique Fonseca. O uso de mensagens transparentes foi sugerido pelo autor em "How to avoid the cost of causal communication in large-scale system", Rodrigues e Veríssimo, Actas do 6º "ACM-SIGOPS Europe Workshop, Dagstuhl, Alemanha, 1994". As mensagens retidas foram concebidas em colaboração com a Dr. Ellen Siegel, e citadas nas actas do 13º "IEEE Symposium On Reliable Distributed Systems", Outubro, 25-27, 1994, Dana Point, Califórnia. O protocolo de ordenação total foi desenvolvido em colaboração com o Eng. Henrique Fonseca.

A interacção com os autores dos restantes módulos da arquitectura foi importante para as ideias aqui apresentadas. A rede abstracta para grande escala está a ser desenvolvida por Jorge Frazão. Os requisitos de configuração do detector de falhas surgiram do trabalho de François Cosquer. O serviço de filiação está a ser desenvolvido por António Sargento. O módulo de grupos de baixo custo está a ser desenvolvido em colaboração com António Sargento, Katherine Guo, Robert V. Renesse, Brad Glade e Ken Birman.

6

Invocação remota fiável

6.1 Enquadramento

Este capítulo apresenta um protocolo de invocação remota genérico, que permite interagir com diferentes algoritmos de gestão da replicação. O protocolo usa os serviços do NAVTECH apresentados nos capítulos anteriores. Deste modo, ilustra-se o potencial deste ambiente para resolver os problemas inerentes à concretização de algoritmos de gestão da replicação. O problema concreto aqui resolvido foi levantado durante a concepção do sistema Romance, e introduzido no Capítulo 3.

6.2 Motivação

Apesar de já terem sido desenvolvidos diversos sistemas que suportam a invocação de serviços replicados, a grande maioria inclui aspectos da gestão da replicação no próprio protocolo de invocação. Isto torna extremamente difícil modificar o protocolo de gestão da replicação sem alterar o protocolo usado pelos seus clientes. Tal constitui uma indesejável limitação na modularidade e transparência do sistema. O protocolo apresentado neste capítulo suporta a invocação tolerante a faltas de serviços replicados, oferecendo não só a usual transparência à localização, mas também transparência aos mecanismos de replicação. O protocolo, denominado GRIP (do Inglês, Generic Remote Invocation Protocol), encontra-se concebido como um conjunto modular de serviços que podem ser configurados de acordo com as necessidades da aplicação. O protocolo baseia-se na utilização dos serviços de filiação e comunicação em grupo descritos

nos capítulos anteriores. Entre os serviços fornecidos inclui-se a detecção distribuída de reinvocações, a qual pode ser discriminada pedido a pedido e activada com três diferentes graus de fiabilidade.

6.3 O modelo GRIP

Antes de se iniciar a descrição do protocolo, introduz-se o modelo de interacção e faz-se uma breve análise da sua decomposição em módulos.

6.3.1 Modelo do sistema e hipóteses

Baseiam-se os protocolos num modelo de computação distribuída em que as entidades activas, denominadas *processos*, comunicam exclusivamente através da troca de mensagens. Os processos executam-se em máquinas (possivelmente heterogéneas) interligadas por um sub-sistema de comunicação orientado para os grupos, como descrito nos capítulos anteriores. Segue-se o modelo *cliente-servidor* (veja a Figura 6.1), em que alguns processos, denominados *servidores*, fornecem serviços a outros processos, denominados *clientes*. Um cliente interage com um servidor usando um protocolo de invocação remota, concretizado por um par de rotinas de adaptação. A rotina de adaptação do cliente mascara a distribuição e a eventual replicação do servidor, actuando como um representante local do serviço remoto, empacotando o pedido numa mensagem que é enviada para a(s) rotina(s) de adaptação do servidor. Sempre que é esperada uma resposta, a rotina de adaptação do cliente espera por uma mensagem da rotina de adaptação do servidor contendo a resposta desejada, desempacota-a, e entrega a resposta ao cliente.

Os servidores podem ser replicados. Para manter a coerência do seu estado, as réplicas comunicam entre si através de um canal de comunicação que é, em termos lógicos, independente dos canais usados para oferecer o serviço aos clientes. Esta comunicação respeita um determinado protocolo de gestão da replicação e é ortogonal à que será referida durante a descrição do GRIP. No entanto, e para preservar garantias

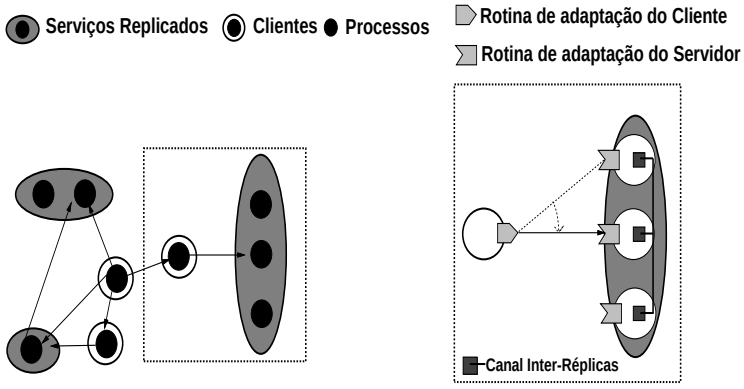


Figura 6.1: Modelo de comunicação do GRIP

de coerência global do sistema, a partilha do mesmo sub-sistema de comunicação em grupo é assumida.

Os processos estão sujeitos a falhas. As omissões na rede são mascaradas pelo sub-sistema de comunicação em grupo. Assumimos que os processos têm um modo de falha controlado, respeitando o modelo de *falha-silenciosa*. Não são pois consideradas falhas arbitrárias.

Assume-se também a existência de um serviço de nomes (SN) exterior ao protocolo (veja-se, por exemplo, o trabalho de Cheriton e Mann (1989) ou de Demers *et al.* (1987)). O serviço de nomes fornece, para cada um dos restantes serviços do sistema, um identificador de grupo e uma lista de identificadores de eventuais membros desse grupo (processos onde potencialmente se executam réplicas desses serviços). O método de actualização do serviço de nomes é ortogonal ao serviço de invocação (como se verá, o GRIP não requer que o serviço de nomes forneça informação completamente actualizada).

6.3.2 Resumo da concepção

Teve-se como objectivo conceber um protocolo a ser executado entre o cliente e o(s) servidor(es) que possuísse o seguinte conjunto de propriedades:

- **tolerância a faltas:** o cliente deve conseguir aceder ao serviço desde que: (i) exista uma réplica acessível; (ii) o serviço de nomes esteja acessível.

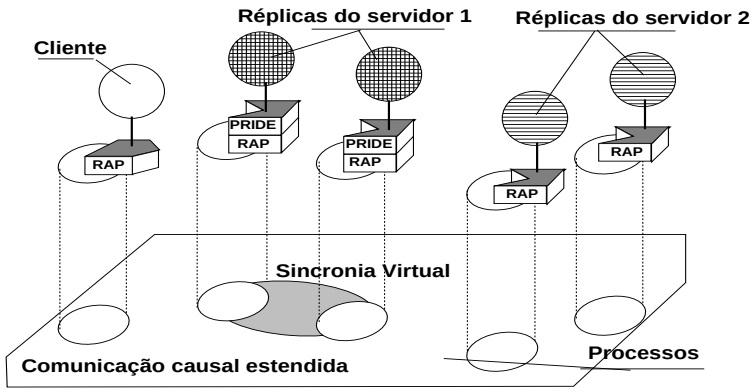


Figura 6.2: Decomposição em módulos do GRIP

- **semântica "no máximo uma vez´´:** mesmo na presença de faltas, cada operação deve ser executada no máximo uma vez no servidor. Esta propriedade deve ser opcional.

- **independência dos métodos de gestão da replicação:** as garantias do protocolo não devem depender do algoritmo usado para manter a coerência das réplicas.

- **baixa sobrecarga:** para uma combinação particular do modelo de coerência e de mecanismos de gestão da mesma, o protocolo deve exibir um desempenho aproximado ao das soluções específicas para esse caso.

De modo a atingir estes objectivos, dividiu-se a funcionalidade num conjunto de serviços modulares, tal como ilustrado na Figura 6.2. Como serviço de suporte, utilizam-se os mecanismos de comunicação e filiação em grupo descritos no Capítulo 3. Os clientes contactam os servidores usando um protocolo de acesso remoto de baixo custo denominado RAP (do Inglês, Remote Access Protocol). O RAP tem como objectivo encaminhar os pedidos e respectivas respostas entre os clientes e os servidores de modo tolerante a faltas, sem no entanto garantir a detecção de reinvocações; se necessária, esta garantia é fornecida por um protocolo de detecção distribuída de reinvocações, denominado PRIDE (do Inglês, "Protocol for Repeated Invocation DEtection").

Name	Tipo	Parâmetros	Devolve
r_invoca	pedido	Msg pedido [, IdPed id]	CódigoErro cr, Msg resposta
r_pedido	indicação	IdPed id, Msg pedido Booleano indefinido, IdProcLista contactos	**nada**
r_resposta	pedido	IdPed id, Msg resposta [, IdProc troca]	**nada**

Tabela 6.1: Interface do RAP

6.4 Acesso remoto de baixo-custo

De acordo com o protocolo GRIP, clientes de um serviço (potencialmente) replicado usam o RAP para contactarem individualmente uma determinada réplica. A principal característica deste protocolo é não requerer o uso de primitivas de comunicação dispendiosas nem uma sincronização estreita entre o cliente e o servidor. De facto, o protocolo não exige a manutenção de informação completamente actualizada acerca da filiação no grupo de réplicas e baseia toda a comunicação no uso de mensagens transparentes (descritas no Capítulo 5). Isto não impede que o RAP garanta o acesso ao serviço desde o servidor de nomes e uma das réplicas estejam acessíveis e que forneça a possibilidade de redirigir o cliente para outra réplica mais adequada. Os serviços do RAP podem ser expandidos através da camada PRIDE.

O protocolo RAP distingue três tipos de mensagens, *pedidos*, *respostas*, e *marcadores de fim de interacção*. Os pedidos são identificados através do identificador do cliente e de um contador; as respostas incluem o identificador do pedido associado e transportam um campo de `troca` cujo objectivo é esclarecido mais adiante. O marcador de fim de interacção é usado pelo cliente para indicar ao servidor a terminação de uma sequência de pedido-resposta. Este marcador pode ser gerado automaticamente pela rotina de adaptação do cliente e, apesar de não ser necessário para garantir a correcção do protocolo, pode ser usado para diminuir a informação mantida pelo GRIP.

As primitivas exportadas pelo protocolo RAP são resumidas na Tabela 6.1:

- **r_invoca** envia um pedido para o serviço replicado. O parâmetro opcional permite que o cliente identifique o pedido. Um campo de `estado` é devolvido, assim

como a resposta sempre que se consegue aceder ao serviço.

- **r_pedido** é uma chamada que encaminha um determinado pedido para a réplica local conjuntamente com uma lista (possivelmente vazia) de processos já contactados.

- **r_resposta** é chamada pela réplica com o fim de devolver uma resposta ao cliente. Um parâmetro opcional de `troca` , permite que a réplica indique uma outra réplica como contacto alternativo para o cliente obter o serviço.

A descrição em forma de pseudo-código do protocolo RAP pode ser encontrada na Figura 6.3 (código cliente) e na Figura 6.4 (código servidor). Assume-se que cada cliente só faz um pedido de cada vez (pedidos concorrentes são modelados por clientes diferentes). Para cada pedido, o cliente vai mantendo uma lista de todos os processos contactados com o fim de obter uma resposta. A aplicação pode obrigar a que esta lista seja mantida num estado `indefinido` , de modo a que o servidor não faça hipóteses acerca do número de vezes que o pedido foi retransmitido[1].

Um cliente inicia o processamento obtendo, através do serviço de nomes, uma lista de processos onde o serviço pode potencialmente ser fornecido. Esta lista não necessita de estar completamente actualizada uma vez que apenas uma réplica necessita ser contactada para aceder ao serviço. A lista é percorrida pelo cliente (o critério utilizado não é relevante embora factores como a proximidade e repartição de carga possam ser tidos em conta) que escolhe um processo como seu `r_contacto` . O cliente envia então o pedido para o contacto escolhido, usando uma mensagem ponto-a-ponto transparente, após o que espera por uma resposta. Recorda-se que as mensagens transparentes são entregues respeitando a ordem causal em relação às mensagens opacas mas não introduzem dependências causais. Reservando-se o uso de mensagens opacas para a comunicação entre réplicas, oferece-se aos clientes uma vista dos dados coerente com as relações de causalidade, sem exigir, no entanto, que estes usem primitivas de comunicação fiáveis para contactar as réplicas.

[1] Deste modo é possível suportar clientes replicados.

```
Cliente
Iniciação:
   (IdProcLista servidores, Grupo r_grupo) := sn_lista_servidores (IdentificadorDoServiço);
   IdProc r_contacto := seleccionaContacto (servidores);
   IdProc fonte := identificador_processo_local;
   Grupo c_grupo := identificador_lógico_cliente;
   Booleano indef := FALSO;
   IdProcLista contactos := nulo;
   Inteiro tentativas :=0;
Corpo:
   quando  r_invoca (pedido, [idu_cliente]) invocado
   execute-se
      se (idu_cliente ≠ nulo) entao
         idu := idu_cliente;
         indef := VERDADEIRO;
      caso-contrario
         idu := geraIdPed();
         indef := FALSO;
         contactos := nulo;
      fim-se;
      até-retornar     execute-se
         início
            tentativas := 0;
            até-que  ([fonte, idu, troca, resposta] recebido ou tentativas = MAX) execute-se
               g_c_transparente (r_grupo, r_contacto, [ c_grupo, idu, pedido, indef, contactos ]);
               tentativas := tentativas + 1;
            fim-até-que
            se ( [fonte, idu, troca, resposta] recebido) entao
               r_contacto := troca;
               servidores := servidores ∪{ r_contacto };
               devolve  resposta ao cliente;
            caso-contrario
               se (indef = FALSO) entao
                  acrescenta(r_contacto, contactos);
               fim-se ;
               r_contacto := próximoContacto (r_contacto, servidores);
               se (r_contacto = nulo) entao
                  obtém nova lista de servidores ou devolve erro;
               fim-se ;
         fim // até retornar
Terminação:
   g_c_transparente (r_grupo, r_contacto, [fonte, FIM-INTERACÇÃO]);
```

Figura 6.3: Pseudo-código do protocolo RAP: cliente

```
Servidor
Corpo: // (para cada pedido)
   quando   g_c_entrega (Grupo g, IdProc fonte, Msg [c_grupo, idu, pedido, indef, contactos], ∅, ∅)
   execute-se
      início
         troca := seleccionaContacto (); // por omissão, o próprio
         r_pedido (idu, pedido, indef, contactos);
      fim ;

   quando   r_resposta (idu, resposta, [s_troca])
   execute-se
      início
         se  (s_troca especificada) então
            troca := s_troca;
         fim-se  ;
         g_c_transparente (c_grupo, fonte, [idu, resposta, troca]);
      fim ;
```

Figura 6.4: Pseudo-código do protocolo RAP: servidor

A mensagem com o pedido é fornecida pela aplicação. A aplicação pode também, e opcionalmente, fornecer um identificador de pedido. Caso contrário, um identificador é gerado automaticamente pela rotina de adaptação do cliente. Esta, após enviar o pedido, espera um período de tempo pré-determinado por uma resposta. Caso esta não chegue, o pedido é retransmitido. Após um número também pré-determinado de retransmissões um novo processo é escolhido para `r_contacto` e o pedido enviado de novo. Estas instruções são repetidas até que se obtenha uma resposta ou até que a lista de contactos seja totalmente percorrida. No último caso, ou a interacção é cancelada ou o serviço de nomes é contactado de novo (de modo a obter uma lista de contactos actualizada). Em cada resposta, o servidor pode especificar um outro processo como o contacto privilegiado para o cliente em causa. Para melhor desempenho, o cliente deve actualizar a variável `r_contacto` de acordo com esta informação. Este mecanismo, não tendo influência na correcção do protocolo, permite que o servidor concretize as suas próprias políticas de repartição de carga.

Do lado do servidor, o RAP espera pedidos e usa a primitiva `r_pedido` para os encaminhar, juntamente com informação de estado, para a réplica local. Posteriormente, a resposta será devolvida ao cliente quando a primitiva `r_resposta` for invocada.

Tipos	p_estado p_modoresp	um-de (original, em-execução, desconhecido, bloqueado) um-de (envie, registe, registe_envie)	
Funções	Nome	Tipo	Parâmetros
	p_pedido	indicação	IdPed mesgid, Msg ped, p_estado estado
	p_registe	pedido	IdPed mesgid
	p_busque	pedido	IdPed mesgid
	p_resposta	pedido	IdPed idped, Msg resposta p_modoresp acção[, IdProc troca]
	p_propagado	pedido	IdPedLista idpeds, Msg mensg, IdProcLista dest, Booleano delegar
	p_uniforme	pedido	ReqIdLista idpeds, Msg mensg, Booleano delegar

Tabela 6.2: Interface do PRIDE

Como se referiu anteriormente, a resposta pode transportar um parâmetro de `troca` , isto é, um novo contacto para o cliente.

O Protocolo RAP é um protocolo de baixo-custo, colocando poucos requisitos de fiabilidade na comunicação e não exigindo informação completamente actualizada acerca da filiação das réplicas. O RAP não executa a detecção de reinvocações mas a arquitectura do GRIP suporta esta funcionalidade através da utilização de um protocolo adicional, denominado PRIDE, o qual será descrito na próxima secção.

6.5 Detecção de reinvocações distribuída

O protocolo de detecção de reinvocações, denominado PRIDE, pode ser executado entre as diversas réplicas de um servidor de modo a oferecer garantias do tipo no-máximo-uma-vez aos clientes. As garantias oferecidas pelo PRIDE são opcionais e podem ser discriminadas invocação a invocação. Deste modo é possível eliminar quase totalmente a sobre-carga introduzida pelo protocolo quando estas garantias não são necessárias.

O protocolo oferece três variantes diferentes de detecção de reinvocações, nomeadamente: *fiável*, que garante que o pedido é submetido no máximo a uma réplica correcta; *propagada*, que associa ao pedido uma mensagem e que garante que o pedido não é resubmetido enquanto pelo menos um dos destinatáriosdessa mensagem permanecer correcto; *uniforme*, que associa uma mensagem ao pedido e garante que este não é

resubmetido caso exista pelo menos uma réplica (correcta ou não) que tenha recebido essa mensagem. Ao oferecer estas três diferentes qualidades de serviço, permite-se que a aplicação decida qual a relação custo/benefício que lhe é mais propícia. O protocolo permite que a qualidade de serviço escolhida seja diferente para cada operação e permite também sobrecarregar a qualidade de serviço durante a execução.

6.5.1 A interface do PRIDE

A interface do PRIDE encontra-se resumida na Tabela 6.2. O PRIDE recebe os pedidos que lhe são indicados pelo protocolo RAP através da chamada `r_pedido` e encaminha-os para a aplicação através da chamada `p_pedido`. Pedidos *originais*, (isto é, que venham com uma lista de contactos vazia) são encaminhados sem qualquer processamento adicional. Isto garante o desempenho óptimo no caso que se espera mais frequente. Caso contrário (a lista de contactos não é nula ou está marcada como *indefinida*), PRIDE procura *localmente* por um registo do pedido (o PRIDE mantém um registo dos pedidos recebidos e troca esta informação periodicamente com os seus pares). Se este registo existir, e caso uma resposta já tenha sido gerada, o PRIDE retransmite a resposta sem invocar a aplicação. Caso não exista um registo local, ou caso exista um registo mas este não contenha uma resposta e o pedido nunca tenha sido submetido localmente, este é encaminhado para a aplicação com uma indicação acerca do seu estado.

Na indicação `p_pedido`, o parâmetro `estado` fornece informação acerca do estado do pedido, podendo assumir um dos seguintes valores: (a) `original`, quando o pedido não foi submetido a nenhuma outra réplica activa; (b) `em execuc¸ao˜`, quando o pedido já foi submetido a uma réplica ainda activa; (c) `desconhecido`, quando o PRIDE necessita de trocar informação com os seus pares para obter o estado do pedido; (d) `bloqueado`, quando uma réplica propagou informação acerca do pedido sem que antes tenha gerado uma resposta. Quando a semântica "no-máximo-uma-vez" é desejada, esta é activada invocando `p_registe`; isto activa a detecção de reinvocações com uma qualidade de serviço `fi´avel`. Quando o estado do pedido é *desconhecido*, a aplicação pode requisitar que o PRIDE contacte os seus pares para obter mais

informação acerca do pedido, invocando `p_busque` ; neste caso o pedido será resubmetido mais tarde à aplicação, mal a busca termine.

Após activar a detecção de reinvocações com a qualidade de serviço fiável, é possível sobrecarregar esta qualidade de serviço. A primitiva `p_propagado` permite associar ao pedido uma mensagem que é disseminada para os processos especificados no parâmetro `dest` . Esta qualidade de serviço garante que enquanto um destes processos estiver correcto o pedido não será resubmetido a nenhuma réplica. O parâmetro `delegar` , da mesma primitiva permite especificar que os destinatários da mensagem ficam responsáveis por produzir uma resposta para o pedido em causa (caso contrário, apenas o processo que invoca a primitiva fica com tal ónus). A primitiva `p_uniforme` funciona de modo semelhante, embora a resubmissão do pedido seja evitada desde que um processo em `dest` tenha recebido a mensagem, mesmo que esse processo já não se encontre acessível.

Uma resposta a um determinado pedido pode ser gerada pela aplicação em qualquer ponto da sua execução. Esta pode ser armazenada localmente para posterior transmissão ou ser imediatamente encaminhada para o cliente.

6.5.2 Os protocolos do PRIDE

De modo a evitar a resubmissão de pedidos, o PRIDE mantém um registo dos pedidos para os quais foi activada a detecção de duplicados. Uma vez que o RAP garante que um determinado cliente não faz um novo pedido sem que antes tenha obtido a resposta ao pedido anterior, é possível manter no máximo o registo de um pedido para cada cliente. Adicionalmente, com ajuda de relógios sincronizados e fazendo hipóteses acerca do tempo máximo de transmissão de mensagens é possível eliminar registos obsoletos. Cada registo contém a seguinte informação:

```
define  tipo  p_tipo = um-de (fiável, propagado, uniforme)
estrutura  Registo início
   Identificador  id;        // o identificador do pedido
   p_tipo         tp;        // o tipo de garantias fornecidas
```

```
    Msg         resp;     // a resposta gerada
    Msg         ped;      // o pedido caso não haja resposta
    IdProc      coord;    // o coordenador de uma busca
    IdProcLista exec;     // conjunto de execução
    IdProcLista prop;     // conjunto de propagação
    IdProcLista env;      // conjunto de envio
fim ;
```

Um registo é criado para um pedido, mal é activada a detecção de reinvocações ou quando é iniciada uma busca para esse pedido. Este registo é actualizado quando uma resposta é gerada, a qualidade de serviço sobre-carregada ou uma busca é terminada. As réplicas também trocam entre si informação acerca dos seus registos locais de modo a acelerar o processo de detecção de duplicados. Esta informação é agregada a todas as mensagens trocadas entre as réplicas e só raramente é necessário enviar mensagens exclusivamente para trocar estes registos. Quando uma mensagem inter-réplicas carrega registos agregados, estes são utilizados para actualizar os registos dos destinatários, como descrito na Figura 6.5.

Antes de descrever em pormenor como estes registos são utilizados, resumimos a funcionalidade das primitivas de interface do PRIDE (vejam-se também as Figuras 6.6 e 6.7).

- `r_pedido` é invocado pelo nível RAP; o procedimento `pFiltraPedidos` é chamado, o qual por sua vez encaminha o pedido para a aplicação usando a chamada `p_pedido` .
- `p_registe` activa a detecção de reinvocações com a qualidade de serviço fiável.
- `p_busque` requisita que o PRIDE contacte o seus pares para obter mais informação acerca de um pedido cujo estado é desconhecido.
- `p_resposta` armazena a resposta no registo do pedido associado (caso exista) e encaminha a reposta para o cliente. É possível especificar que apenas uma destas operações seja executada.

```
declara  função prideAgrega (Msg m, IdProcList dest)
  para-todo  o registo  r execute-se
    se (dest - r.env ≠ ∅) então
      r.env := r.env ∪ dest;
      agregar r a m;
    fim-se;

declara  função prideDesagrega (Msg m, IdProcLista dest)
  para-todo  o registo  rnovo ∈ m execute-se
    início
      rlocal := registoLocal (rnovo.id);
      se (rlocal = nulo) então
        adicionar rnovo à lista local;
      caso-contrário // actualizar os campos de rlocal
        se ((rnovo.resp ≠ nulo) ∧ (rlocal.resp = nulo) então
          rlocal.resp := rnovo.resp;
        se (rnovo.tp > rlocal.tp) então
            rlocal.tp := rnovo.tp;
            rlocal.exec := rlocal.exec ∪ rnovo.exec;
            rlocal.prop := rlocal.prop ∪ rnovo.prop;
            rlocal.sent := rlocal.env ∪ rnovo.env;
        fim-se
      fim-se ;
    fim;
```

Figura 6.5: Disseminação de registos

```
// a partir do nível RAP
quando   r_pedido (idped, pedido, indef, contactos) invocado execute-se
   pFiltraPedido (idped, pedido, indef, contactos);

// a partir da réplica local
quando   p_registe (idped) invocado execute-se
   início
      reg := novoRegisto (idped);
      reg.ped := pedido;
      reg.tp := fiável;
      reg.exec := processo_local;
   fim ;

quando   p_busque (idped) invocado execute-se
   início
      reg := novoRegisto (idped);
      reg.ped := pedido;
      reg.tp := fiável;
      reg.exec := ∅;
      pBusca (idped);
   fim ;

quando   p_resposta (idped, resposta, acção[, troca]) invocado execute-se
   início
      se ((acção = registe) ∨ (acção = registe_envie)) então
         pedido := registoLocal (idped);
         se ((pedido = nulo) ∨ processo_local ∉ pedido.exec então
            devolver um erro;
         fim-se ;
         pedido.resp := resposta;
         pedido.env := ∅;
      fim-se
      se ((acção = enviar) ∨ (acção = registe_envie)) então
         r_resposta (idped, resposta[, troca]);
      fim-se   // encaminhar a resposta para o cliente
   fim ;
```

Figura 6.6: Gestão da Interface PRIDE

```
quando  p_propagado (idpeds, m, dest, delegar) invocado execute-se
   início
      para-todo   mid ∈ idpeds execute-se
         Registo ped := registoLocal (mid);
         ped.tp := propagado;
         ped.env := ∅; ped.prop := dest;
         se (¬ delegar) então
            ped.exec := processo_local;
         caso-contrário
            ped.exec := dest;
         fim-se ;
      fim-para-todo   ;
      g_sv_difusão (grupo_réplicas, dest, m) ; // registos pendentes serão agregados
   fim ;

quando  p_uniforme (idpeds, m, delegar) invocado execute-se
   início
      para-todo   mid ∈ idpeds execute-se
         Registo reg := registoLocal (mid); reg.tp := uniforme;
         reg.env := ∅; reg.prop := todas_as_réplicas;
         se (¬ delegar) então
            reg.exec := processo_local;
         caso-contrário
            reg.exec := todas_as_réplicas;
         fim-se ;
      g_c_uniforme (grupo_réplicas, todas_as_réplicas, m); // registos pendentes serão agregados
   fim ;
```

Figura 6.7: Gestão da Interface PRIDE (continuação)

```
declara  função pFiltraPedido (IdPed id, Msg pedido, Booleano indef, IdProcLista contactos)
  se ((indef = FALSE) ∧ (contactos = ∅)) então // processar
    p_pedido (id, pedido, original);
  caso-contrário // reinvocação potencial
    reg := registoLocal (id); // procurar registo
    se (reg = nulo) então // não existe registo
      p_pedido (id, pedido, desconhecido);
    caso-contrário // registo disponível
      pProcessaRegisto (reg, busca_feita = FALSO);
    fim-se
  fim-se
```

Figura 6.8: Filtro de pedidos no PRIDE

- `p_propagado` sobre-carrega a qualidade de serviço de *fiável* para *propagada*. No campo *prop* do registo é armazenada a lista de processos para qual a mensagem foi enviada. O campo `exec` é também actualizado caso o parâmetro `delegar` tenha sido activado. A mensagem é então difundida para todos os destinatários especificados, não sem que antes a ela sejam agregados todos os registos actualizados (inclusivé este).

- `p_uniforme` sobre-carrega a qualidade de serviço para *uniforme*. Os campos do registo são actualizados de modo semelhante ao caso anterior. A mensagem é difundida usando a qualidade de serviço *uniforme*.

O mecanismo que filtra os pedidos recebidos está ilustrado nas Figuras 6.8 e 6.9. Sempre que o RAP entrega um pedido ao PRIDE, este indica uma lista de réplicas previamente contactadas. Sempre que esta lista se encontra vazia o pedido é encaminhado imediatamente para a aplicação. Caso contrário, PRIDE procura um registo local. Se este registo não existe, o pedido é entregue à aplicação com uma indicação de `desconhecido`. A entrega de um pedido com indicação de `desconhecido` pode iniciar uma busca de mais informação, a qual é descrita pelo procedimento `pBusca`.

Quando um registo existe localmente, este é examinado pelo procedimento `pProcessaRegisto` para determinar qual a acção mais apropriada. Quando uma resposta já se encontra disponível (o pedido já foi processado e registado), esta é retransmitida para o cliente. Quando ainda não se encontra disponível uma resposta,

```
declara  função pProcessaRegisto (Registo reg, Booleano busca_feita)
  se (reg.resp ≠ nulo) então
    r_resposta (id, reg.resp); // encaminhar a resposta para o cliente
  caso-contrário se (reg.exec ∩ r_vista ≠ ∅) então
    p_pedido (id, reg.ped, em-execução);
  caso-contrário se (reg.prop ∩ r_vista ≠ ∅) então
    p_pedido (id, reg.ped, bloqueado);
  caso-contrário se (reg.tipo = uniforme) então
    devolve bloqueado; // réplica inacessível
  caso-contrário se (reg.coord ∈ r_vista ∧ reg.coord ≠ processo_local) então
    p_pedido (id, reg.pedido, em-execução);
  caso-contrário se (busca_feita = TRUE) então // processar como original
    p_pedido (id, reg.ped, original);
    reg.tp := fiável;
    reg.exec := processo_local;
  caso-contrário se (p_pedido (id, reg.ped, desconhecido) = busque) então
    pBusca (id);
```

Figura 6.9: Processamento de Registos no PRIDE

os restantes campos são analisados: quando uma réplica se encontra já a executar o pedido, ou já iniciou uma busca para esse mesmo pedido, o pedido é fornecido à aplicação com indicação de `executando`; caso o pedido tenha sido `propagado` para outra réplica activa, ou disseminado de maneira `uniforme` mas nenhuma réplica activa se encontra a executar o pedido, este é indicado com a indicação de `bloqueado`. Neste ponto do processamento, o estado do pedido depende de já se ter feito uma busca ou não: em caso afirmativo o pedido é indicado como `original`, caso contrário como `desconhecido`.

Sempre que um pedido é submetido com o estado de `desconhecido`, a aplicação pode decidir iniciar uma busca. Se `p_busque` é chamada, a detecção de duplicados é activada e o PRIDE inicia uma busca por registos acerca do pedido em causa. Em alguns casos, como por exemplo o caso em que o cliente está replicado, é possível que o mesmo pedido seja enviado para diferentes réplicas em simultâneo. Deste modo, é possível que diversas buscas sejam iniciadas concorrentemente.

O procedimento de busca, `pBusca` está ilustrado na Figura 6.10. O processo de busca é iniciado com a transmissão de uma difusão fiável totalmente ordenada. O processo que inicia a busca, que passaremos a designar pelo *coordenador*, espera

```
declara  função pridеProcessaBusca (PedId id)
  início
    reg := registoLocal (id); // procurar o resultado da busca
    pProcessaRegisto (reg, busca_feita = TRUE);
  fim ;

declara  função pRecebeBusca (PedId id, IdProc coordenador)
  reg:=registoLocal(id);
  se (reg = nulo) então
    reg := novoRegisto (id);
    reg.exec := ∅;
    reg.tp := fiável;
    reg.coord := coordenador;
  caso-contrário se (reg.coordenador ∉ r_vista) então
    reg.coord := coordenador;
    g_sv_difusão (grupo_réplicas, coordenador, [BUSCA-CONF]);
    // agregar os registos locais a BUSCA-CONF
  fim-se ;

declara  função pBusca (IdPed id)
  início
    g_c_atómica (grupo_réplicas, [BUSCA, id, processo_local], todas_as_réplicas);
    esperar por um BUSCA-CONF de cada réplica activa;
    // desagregar os registos remotos do BUSCA-CONF e adicionar aos registos locais
    prideProcessaBusca (id);
  fim ;
```

Figura 6.10: Funções do busca no PRIDE

por uma resposta de cada processo. As respostas são analisadas no procedimento `prideProcessaBusca`. Tal como discutido anteriormente, em resultado da busca o pedido pode ser resubmetido à aplicação. O uso de comunicação totalmente ordenada é necessário para seriar buscas concorrentes: a primeira busca a ser recebida ganha a competição.

Ao receber um pedido de busca, as réplicas processam-no através do procedimento `pRecebeBusca`. Este procedimento procura um registo local e, de modo a garantir resultados determinísticos, cria um novo registo caso este não exista. Este novo registo é iniciado com um campo `exec` nulo para indicar que o estado de execução é desconhecido. No campo `coord` do registo memoriza-se a identidade do coordenador da busca; caso a este campo esteja já atribuído o identificador de outro processo, este só é substituído se já não pertencer à vista corrente. Finalmente, o registo local devidamente actualizado é devolvido ao coordenador usando uma mensagem ponto-a-ponto fiável.

6.5.3 Alteração na filiação

O protocolo PRIDE é particularmente robusto em relação a alterações na filiação porque utiliza uma aproximação pessimista ao iniciar uma busca sempre que não existe informação local sobre um determinado pedido. Deste modo, não é necessário executar nenhuma operação especial para integrar novas réplicas (naturalmente, ao nível da aplicação existirá um protocolo para integrar a nova réplica; no entanto, este protocolo é invisível para o PRIDE). Caso receba uma reinvocação, uma nova réplica irá simplesmente iniciar uma busca para obter informação actualizada acerca desse pedido. Quando o protocolo opera sobre redes particularmente sujeitas a omissões, e se espera que a existência de retransmissões seja frequente, pode-se obrigar a nova réplica a obter os registos dos pedidos mais recentes de outra réplica activa antes de iniciar a sua operação (isto poderá eliminar algumas buscas). Esta última alternativa é opcional e não é indispensável para garantir a correcção do protocolo.

Do mesmo modo, se acontece uma alteração à filiação devido a uma falha ou partição, o PRIDE não necessita de tomar nenhuma acção especial. A camada de comunicação virtualmente síncrona indicará uma nova vista (Figura 6.11) e a camada

```
declara   função prideMudaVista (IdProcLista nova_vista)
  se ((nova_vista - r_vista) ≠ ∅) então // novos membros
    // em opção, enviar registos recentes para os novos membros
    r_vista = nova_vista;
    para-todo reg : reg.exec ∩ nova_vista = ∅ execute-se
      pBusca (reg.id); // recuperação activa (opcional)
  fim-se ;
```

Figura 6.11: Mudança de vista no PRIDE

RAP gerará uma retransmissão do pedido. Quando essa retransmissão chegar a uma das réplicas sobreviventes, as falhas serão detectadas durante a execução do procedimento `pFiltraPedido` e a acção correctiva apropriada será iniciada (quando possível, o pedido será submetido a uma réplica activa). Como alternativa a este modo de recuperação passivo, a camada PRIDE pode iniciar uma busca para todos os pedidos afectados por uma falha mal esta é indicada. Este procedimento é equivalente a receber uma retransmissão dos pedidos em causa. Esta técnica de recuperação activa pode oferecer um menor tempo de recuperação.

6.6 Exemplos

Nesta secção ilustra-se como o protocolo GRIP pode ser usado para concretizar diferentes algoritmos de gestão da replicação. Serão dados três exemplos cobrindo técnicas significativamente distintas e diferentes modelos de coerência de memória.

O primeiro exemplo, ilustrado através da Figura 6.12, mostra como uma máquina de estados replicada pode ser desenvolvida usando o suporte oferecido pelo GRIP. Todos os pedidos de leitura são executados localmente por qualquer das réplicas que receba o pedido sem qualquer sincronização. Um pedido de escrita recebido como original é registado na camada PRIDE devolvendo `registe` ; é então difundido de modo atómico usando uma mensagem especial ECOA para o conjunto de todas as réplicas usando a primitiva `p_propagado` , reforçando a qualidade de serviço para *propagado*. A variável `delegar` é iniciada a VERDADEIRO, indicando que todos os destinatários devem executar o pedido incluído. Quando o estado do pedido é

```
quando  p_pedido (id, pedido, estado) invocado e o pedido é uma leitura
execute-se
   in´ıcio
      resposta := executar (pedido);
      r_resposta (id, resposta) // não é necessário activar o PRIDE
   fim ;

quando  p_pedido (id, pedido, estado) invocado e o pedido é uma escrita
execute-se
   se (estado = original) ent˜ao
      p_registe (id);
      p_propagar (id, ECOA [id, pedido], todas_as_réplicas, VERDADEIRO);
   caso-contr´ ario se (estado = desconhecido) ent˜ao
      p_busque (id);
   fim-se ;

quando  ECOA [id, pedido] recebida de uma réplica R
execute-se
   in´ıcio
      resposta:= executar (pedido);
      se (processo_local = R) ent˜ao
         p_resposta (id, resposta, registe_envie);
      caso-contr´ ario
         p_resposta (id, resposta, registe);
      fim-se ;
   fim ;
```

Figura 6.12: Concretização de uma máquina-de-estados replicada

`desconhecido` então devolve-se `busque` de modo a iniciar uma busca global por informação: caso seja necessário, `p_request` será invocado de novo como resultado da busca. Em todos os outros casos devolve-se `ignore` uma vez que não afectam a coerência da máquina de estados. Finalmente, quando a mensagem ECOA é recebida o pedido é executado em cada réplica (neste exemplo apenas a réplica que recebeu o pedido directamente do cliente propaga a resposta para o mesmo).

O segundo exemplo, ilustrado na Figura 6.13, demonstra a concretização de um esquema de gestão da replicação baseado num modelo de coerência enfraquecido. Assume-se que as operações de leitura podem ser executadas imediatamente após a sua recepção. Neste exemplo, as operações de escrita podem ser legalmente reexecutadas por uma segunda réplica, caso a primeira falhe sem ter conseguido propagar os resultados dessa operação. Cada réplica pode executar em paralelo pedidos de clientes concorrentes e os resultados das operações são apenas propagados de vez em quando,

```
quando  p_pedido (id, pedido, estado) invocado e o pedido é uma leitura
execute-se
   início
      resposta := executar (pedido);
      r_resposta (id, resposta) // não é necessário activar o PRIDE
   fim ;

quando  p_pedido (id, pedido, estado) invocado e pedido é uma escrita
execute-se
   se (estado = original) então // é seguro executar
      p_registe(id);
      idlista := idlista + id;
      resposta := executar (pedido);
      diferenças := coleccionaDiferenças();
      g_c_retida (grupo_réplicas, todas_as_replicas);
      p_resposta (id, resposta, registe_envie);
   caso-contrário se (estado = desconhecido) então
      p_busque (id);
   fim-se ;

quando  tempo para propagar actualizações para dest ∨ g_c_liberta (grupo_réplicas, dest)
execute-se
   início
      p_propagado (idlista, dest, diferenças, FALSO);
      diferenças := 0;
      idlista := 0;
   fim ;
```

Figura 6.13: Propagação fraca

de acordo com os critérios da política de replicação. A reexecução destes pedidos após a propagação dos resultados é prevenida através do uso da qualidade de serviço *propagada* do protocolo PRIDE.

Tal como no exemplo da máquina de estados, se o estado do pedido é original então a sua execução é segura e no caso do estado ser `desconhecido` é requerida uma busca para obter informação adicional. Quando o pedido está em execução numa outra réplica, ou no caso em que o pedido se encontra bloqueado, o destinatário não executa o pedido. Neste exemplo, a propagação dos resultados é atrasada arbitrariamente, de acordo com critérios específicos da política de replicação. Deste modo, todas as actualizações são inseridas na história causal usando as *mensagens retidas*. Quando os dados correspondentes às mensagens retidas são disseminados, invoca-se a primitiva `p_propagada` indicando quais os pedidos associados a essas mensagens, cuja reexecução será evitada. Quando um pedido é executado, a resposta é registada pelo

protocolo PRIDE e enviada para o cliente.

O último exemplo, ilustrado na Figura 6.14, mostra a concretização de um esquema do tipo primário-secundário, no qual as actualizações devem ser asseguradas nas diversas réplicas antes da resposta ser disseminada. Pedidos de leitura são executados localmente (em qualquer réplica). Pedidos de escrita são executados apenas na réplica primária; se um secundário recebe um pedido de escrita ecoa-o para o primário[2]. O primário assegura cada actualização usando a primitiva `p_uniforme` do serviço PRIDE, garantindo deste modo que o pedido não é reexecutado e que todas as cópias armazenam a actualização. De seguida, o primário encaminha a resposta para o cliente.

Apesar de serem apenas apresentados alguns exemplos, é possível elaborar outros de modo semelhante. Note-se também que as aplicações que não necessitam dos serviços da camada PRIDE podem usar a camada RAP directamente.

6.7 Discussão e trabalho relacionado

Tanto quanto é do conhecimento do autor, existem poucos exemplos na literatura concebidos para inter-operar com diversas estratégias de replicação. A grande maioria dos protocolos de invocação remota são concebidos para operar de acordo com um algoritmo específico de gestão da replicação e não têm em consideração os problemas que o protocolo GRIP tenta resolver. Relaciona-se a aproximação aqui descrita com alguns dos mais relevantes exemplos de protocolos de invocação remota de serviços replicados.

Um dos primeiros protocolos na área foi apresentado por Cooper (1984). Esse protocolo permite a invocação de um "*bando*´´ de réplicas embora de um modo mais restritivo do que o protocolo GRIP; não só requer programas determinísticos mas assume também que todas as réplicas recebem e processam todos os pedidos.

O protocolo apresentado por Ladin *et al.* (1990) é semelhante ao GRIP no que se refere ao uso de comunicação ponto-a-ponto entre o cliente e o serviço replicado.

[2]Nalguns casos, pode ser desejável fazer com que o contacto do cliente seja o primário (embora nos casos em que o cliente faça maioritariamente leituras esta migração possa não ser óptima).

```
quando  p_pedido (id, pedido, estado) invocado e o pedido é uma op de leitura
execute-se
    início
       resposta := executar (pedido);
       r_resposta (id, resposta)// não é necessário activar o PRIDE
    fim ;

quando  p_pedido (id, pedido, estado) invocado e
    o pedido é uma op de escrita mas eu não sou o PRIMÁRIO
execute-se
    se estado = original então
       p_registe (id);
       p_propagado (id, ECOA [id, pedido, estado], PRIMÁRIO, delegar := VERDADEIRO);
    caso-contrário se (estado = desconhecido) então
       p_busque (id);
    fim-se ;

quando  p_pedido (id, pedido, estado) invocado e pedido é uma op de escrita e eu sou o PRIMÁRIO ou
quando  ECOA[id,pedido, estado] recebido
execute-se
    se estado = original então
       p_registe (id);
       resposta := executar (pedido);
       p_resposta (id, resposta, acção := registe);
       p_uniforme (id, actualização, todas_as_replicas, delega := FALSO);
       // espera até que a propagação esteja assegurada
       p_resposta (id, resposta, acção := envia, troca := processo_local);
    caso-contrário se (estado = desconhecido) então
       p_busque (id);
    fim-se ;

quando  actualização recebida do PRIMÁRIO
execute-se
    se sou o PRIMÁRIO então
       ignorar
    caso-contrário
       armazenar a actualização
    fim-se ;

quando  PRIMÁRIO falha execute-se
    seleccionar novo PRIMÁRIO;
```

Figura 6.14: Concretização de primário-secundário

Propriedades de coerência global são obtidas coleccionando e trocando estampilhas semelhantes a relógios vectoriais. O protocolo GRIP obtém o mesmo efeito ao confiar nos serviços da camada de comunicação em grupo. No entanto, a aproximação GRIP não se restringe a um único modelo de replicação: o recurso ao serviço de mensagens retidas permite obter os mesmos benefícios mas preserva uma maior transparência em relação aos mecanismos usados para concretizar o serviço. Para além do mais, a aproximação GRIP permite que o sub-sistema de comunicação use os seus próprios mecanismos de compressão da informação causal, libertando o protocolo de invocação remota dessa tarefa.

Mais recentemente foi apresentado por Wood (1993) um protocolo de chamada a procedimentos remotos executando-se sobre o sistema Amoeba. Apesar do Amoeba fornecer serviços de difusão fiável bastante eficientes, o protocolo recorre a comunicação ponto-a-ponto para que os clientes comuniquem com uma réplica seleccionada, designada por coordenador. Esta escolha deveu-se sobretudo ao facto da comunicação em grupos no Amoeba ser fechada (isto é, apenas membros do grupo podem comunicar para o grupo) embora os autores também sublinhem as vantagens do ponto de vista de transparência inerentes à aproximação. No entanto, e ao contrário da aproximação genérica seguida pelo GRIP, o coordenador assume que as réplicas são mantidas coerentes através de uma aproximação tipo máquina-de-estados e obriga a que os pedidos sejam sempre difundidos de modo atómico para o grupo de réplicas. Os protocolos do GRIP permitem que a aplicação decida quando essa propagação deve ser feita e permite também a selecção de um coordenador distinto para cada pedido, evitando a formação de pontos de estrangulamento.

6.8 Sumário

Os protocolos aqui apresentados fornecem a flexibilidade necessária para suportar a invocação remota de componentes replicados. Ao contrário da funcionalidade fornecida pela maioria dos serviços desenvolvidos anteriormente, o GRIP é independente da estratégia usada para manter a coerência das cópias e fornece suporte para

a reassociação dinâmica entre o cliente e o servidor. É fornecido suporte explícito para o desenvolvimento de modelos de coerência de memória enfraquecidos e fornece-se, discriminadamente, detecção distribuída de reinvocações. Ao isolar o cliente da concretização do serviço e ao fornecer os mecanismos para uma rápida adaptação a mudanças na filiação do grupo de servidores, o protocolo tem em consideração a evolução dinâmica dos serviços e a grande escala do sistema.

Notas

O trabalho aqui descrito foi realizado em colaboração com a Dr. Ellen Siegel. Partes seleccionadas dos resultados aqui apresentados foram publicadas nas actas do 13º "IEEE Symposium On Reliable Distributed Systems", Dana Point, California, Outubro 1994.

7

Conclusões e trabalho futuro

Nesta tese estudou-se o suporte à computação orientada aos grupos em sistemas distribuídos tolerantes a faltas. Designa-se por computação orientada aos grupos uma aproximação ao desenvolvimento de programas distribuídos que consiste em estruturar as aplicações em torno de serviços de filiação e comunicação em grupo.

A necessidade deste tipo de serviços foi motivada pelo estudo de uma técnica fundamental para o desenvolvimento de sistemas distribuídos eficientes e tolerantes a faltas: a gestão de dados replicados. Através de uma panorâmica sobre os mais utilizados modelos de coerência de memória e sobre os algoritmos que os permitem concretizar, identificaram-se requisitos de filiação e disseminação de informação. Estes requisitos são satisfeitos por plataformas de grupos, as quais foram também revistas.

No âmbito do desenvolvimento, pelo Grupo de Automatização Industrial e Sistemas Distribuídos do INESC, de uma plataforma de grupos denominada NAVTECH, apresentaram-se diferentes algoritmos para filiação e comunicação em grupo adequados, respectivamente, a duas grandes classes de infra-estruturas físicas de comunicação: as redes locais e as redes de grande escala.

O potencial dos serviços propostos foi ilustrado através do desenvolvimento de um protocolo genérico de invocação remota. Este protocolo demonstra como é possível suportar diversos algoritmos de gestão da replicação, de um modo integrado, através da composição dos serviços de filiação e comunicação anteriormente apresentados.

Estes resultados enquadram-se nos objectivos inicialmente propostos: o protocolo de invocação remota suporta o desenvolvimento e o uso de objectos replicados; sendo genérico, permite utilizar a técnica de gestão da replicação mais adequada ao problema

que pretende resolver; trata-se de uma solução elegante e modular baseada na utilização de uma plataforma de grupos; mostrou-se que os serviços de filiação e comunicação em grupo podem ser oferecidos com eficácia sobre redes locais; finalmente, mostrou-se que através da escolha de serviços mais fracos, em combinação com a utilização de algoritmos adaptativos e de algoritmos que têm a topologia em consideração, é possível obter desempenhos satisfatórios sobre redes de grande escala.

Como se sublinhou na introdução, os objectivos atingidos nesta tese não são mais que passos intermédios de um objectivo maior, almejado a longo prazo, materializado na arquitectura NAVTECH. Sendo um objectivo ambicioso, possui a capacidade de albergar diverso trabalho de investigação, o qual poderá ser executado no âmbito de trabalhos finais de curso, teses de mestrado e de doutoramento. No curto prazo, espera-se a conclusão de duas teses de mestrado, uma na área da rede abstracta para grande escala e outra na área da filiação em grande escala. Nos parágrafos que se seguem, enumeram-se alguns aspectos em aberto cuja abordagem se considera relevante:

- Actualização dos componentes para rede local. Estes componentes foram cronologicamente os primeiros a serem desenvolvidos na arquitectura NAVTECH. A experiência entretanto obtida, não só através do esforço de concepção dos componentes para grande escala, mas também por outros grupos de investigação na área da comunicação em rede local (Amir *et al.*, 1992; Amir *et al.*, 1993) pode ser aproveitada para melhorar estes componentes. Nomeadamente, justifica-se a introdução de um mecanismo de maioria no serviço de filiação, o desenvolvimento de uma primitiva de difusão uniforme, o eliminar a dependência da propriedade de ordem na rede, e a integração das histórias causais estendidas.

- Concretização dos protocolos desenvolvidos para rede de grande escala. Estes protocolos, já programados mas apenas executados no simulador, poderão ser facilmente transportados para execução na arquitectura NAVTECH assim que os componentes correspondentes à rede abstracta e o serviço de filiação se encontrem disponíveis. Após esse passo, será possível desenvolver uma versão para distribuição à semelhança do que já acontece para os componentes para rede

local[1].

- Verificação da correcção dos algoritmos desenvolvidos recorrendo a técnicas de especificação e validação formal. Entretanto, esperam-se vir a desenvolver provas informais para os algoritmos apresentados (uma prova informal para o algoritmo causal foi já iniciada mas não se encontra concluída (Rodrigues & Veríssimo, 1995a)).

- Utilização de um serviço de filiação parcial. Os algoritmos apresentados assumem a existência de um serviço de filiação linear, que garante a animação numa única partição do sistema. O componente de filiação que se encontra em desenvolvimento irá oferecer, para além de um serviço linear, um serviço de filiação parcial forte, o qual deverá ser tido em consideração pelos restantes serviços.

- Estudo da aplicabilidade destas soluções a novas classes de redes como, por exemplo, redes móveis. Existem suficientes pontos de contacto entre as redes de grande escala e as redes móveis para justificar a adaptação de algumas das técnicas aqui propostas para estes tipos de redes (Rodrigues *et al.*, 1995b).

- Desenvolvimento da camada de integração. Tal como se referiu anteriormente, o desfasamento temporal entre a concepção das duas versões da arquitectura originou alguma desadaptação entre as respectivas interfaces. Referiu-se anteriormente que é possível remodelar os componentes orientados à rede local para eliminar estes problemas. Dado que os componentes para a rede de grande escala já foram concebidos tendo em vista a sua posterior integração com os componentes orientados à rede local, o desenvolvimento da camada de integração não se afigura particularmente difícil. No entanto, a experiência mostra que no momento da concretização se revelam sempre dificuldades inesperadas, pelo que só um trabalho mais apurado nesta área permitirá avaliar correctamente a dificuldade do problema.

- Desenvolvimento de ambientes de programação, em particular, o retomar da

[1] Neste momento, o código de uma versão dos componentes para rede local encontra-se disponível por FTP anónimo (e por WWW, no URL "*http://pandora.inesc.pt*"), sendo fornecido sem qualquer encargo a todos os interessados (mediante uma declaração de utilização para fins não comerciais).

arquitectura Romance. Este aspecto é certamente um elemento chave para promover a utilização das tecnologias desenvolvidas, cuja importância é confirmada por trabalhos paralelos (Maffeis, 1995). Trata-se no entanto de um esforço multidisciplinar, que para ter sucesso deve abranger outras áreas, tais como o suporte sistema operativo e as técnicas de desenvolvimento de programas, pelo que se espera poder vir a realizar em colaboração com outros grupos do INESC.

- Utilização da arquitectura no desenvolvimento de aplicações, de modo a recolher experiência dos programadores e utilizadores finais. No âmbito do projecto Broadcast, como resultado da colaboração do Grupo de Sistemas Distribuídos e Automatização Industrial com o Grupo de Técnicas de Interacção Multimédia, desenvolveu-se um protótipo de uma aplicação cooperativa utilizando a arquitectura NAVTECH (Cosquer, 1995a). Este protótipo, que demonstra as vantagens da utilização de detectores de falhas configuráveis, pode ser estendido de modo a utilizar os restantes serviços da arquitectura.

Reserva-se para o fim uma achega para o aceso debate em volta das vantagens da "orientação para os grupos". A noção de grupo é intrínseca à especificação de muitas aplicações e, para além disso, constitui também uma ferramenta extremamente útil para lidar com os problemas de escala, nomeadamente para modelar hierarquias e agrupamentos. Mostrou-se que soluções para problema da filiação e comunicação em grupo são, não só extremamente complexas, mas também muito dependentes das propriedades do sub-sistema de comunicação utilizado. Por este motivo, defende-se que estes serviços sejam concretizados por plataformas concebidas para o efeito.

Notas

A utilização da arquitectura NAVTECH *sobre redes móveis encontra-se discutida em "Reliable Computing over Mobile Networks", Rodrigues, Fonseca e Veríssimo, Actas do 5º "IEEE Workshop on Future Trends of Distributed Computing Systems", Cheju Island, Korea, Agosto 1995.*

A

Glossário Português-Inglês

alguma vez	eventually
animação	liveness
atomicidade	atomicity
bando	troupe
cancelamento	abort
consenso	consensus
coerência atómica	atomic consistency
coerência causal	causal consistency
coerência forte	strong consistency
coerência híbrida	hybrid consistency
coerência na entrada	entry consistency
coerência na saída	release consistency
coerência por processador	processor consistency
coerência por processo	process consistency
coerência sequencial	sequencial consistency
confiança no funcionamento	dependability
confirmação	commit

confirmação atómica	atomic commit
confirmação em duas-fases	two-phase commit
conversor de protocolos	gateway
coutada	coterie
disponibilidade	availability
durabilidade	durability
equivalência a uma cópia	1-copy equivalence
embaixador	ambassador
encaminhador	router
estampilha	timestamp
exactidão	accuracy
fiabilidade	reliability
filiação	membership
gestor de periférico	device driver
grelha	grid
histórico	log
isolamento	isolation
linearizabilidade	linearizability
modelo de coerência	consistency model
plenitude	completeness
rotina de adaptação	stub
segurança	safety

serializabilidade	serializability
sincronia virtual	virtual syncrony
temporizador	timer
transacção	transaction
trinco	lock
vista	view
votação	voting
votação ponderada	weighted voting
votação maioritária	majority voting

B

Glossário Inglês-Português

1-copy equivalence	equivalência a uma cópia
abort	cancelamento
accuracy	exactidão
ambassador	embaixador
atomic commit	confirmação atómica
atomic consistency	coerência atómica
atomicity	atomicidade
availability	disponibilidade
causal consistency	coerência causal
commit	confirmação
completeness	plenitude
consensus	consenso
consistency model	modelo de coerência
coterie	coutada
dependability	confiança no funcionamento
device driver	gestor de periférico
durability	durabilidade

entry consistency	coerência na entrada
eventually	alguma vez
gateway	conversos de protocolos
grid	grelha
hybrid consistency	coerência híbrida
isolation	isolamento
linearizability	linearizabilidade
liveness	animação
lock	trinco
log	histórico
majority voting	votação maioritária
membership	filiação
process consistency	coerência por processo
processor consistency	coerência por processador
release consistency	coerência na saída
reliability	fiabilidade
router	encaminhador
safety	segurança
sequencial consistency	coerência sequencial
serializability	serializabilidade
strong consistency	coerência forte
stub	rotina de adaptação

timer	temporizador
timestamp	estampilha
transaction	transacção
troupe	bando
two-phase commit	confirmação em duas-fases
view	vista
virtual syncrony	sincronia virtual
voting	votação
weighted voting	votação ponderada

Bibliografia

AGRAWAL, D., & ABBADI, A. EL. 1990 (Novembro). Efficient Techniques for Replicated Data Management. *Páginas 48–52 de: Proceedings of the IEEE Workshop on the Management of Replicated Data*. Houston, Texas, USA.

AGRAWAL, D., & ABBADI, A. EL. 1991. An Efficient and Fault-Tolerant Solution for Distributed Mutual Exclusion. *ACM Transaction on Computer Systems*, **9**(1), 1–20.

AHAMAD, M., & AMMAR, M. 1991. Multidimensional Voting. *ACM Transactions on Computer Systems*, **9**(4), 339–431.

AHAMAD, M., HUTTO, P., & JOHN, R. 1991 (Maio). Implementing and Programming Causal Distributed Shared Memory. *Páginas 274–281 de: Proceedings of the 11th IEEE International Conference on Distributed Computing Systems*. Arlington, Texas, USA.

ALONSO, R., COVA, L., & BARBARA, D. 1990 (Outubro). Using Stashing to Increase Node Autonomy in Distributed File Systems. *Páginas 12–21 de: Proceedings of the 9th IEEE Symposium on Reliable Distributed Systems*. Huntsville, Alabama, USA.

ALSBERG, P., & DAY, J. 1976 (Outubro). A principle for resilient sharing of distributed resources. *Páginas 562–570 de: Proceedings of the Second International Conference on Software Engineering*.

AMIR, Y., DOLEV, D., KRAMER, S., & MALK, D. 1992. Transis: A Communication Sub-System for High-Availability. *Páginas 76–84 de: Digest of Papers, The 22nd IEEE International Symposium on Fault-Tolerant Computing Systems*.

AMIR, Y., MOSER, L., MELLIAR-SMITH, P., AGARWAL, D., & CIARFELLA, P. 1993 (Maio). Fast message ordering and membership using a logical token-passing ring. *Páginas 551–560 de: Proceedings of the 13th IEEE International Conference on Distributed Computing Systems*.

ANSA. 1987. *ANSA Reference Manual, Release 00.03*. ESPRIT technical week edn. Advanced Networked Systems Architecture.

ATTIYA, H., & FRIEDMAN, R. 1992. A correctness condition for high-performance multiprocessors. *Páginas 679–690 de: Proceedings of the 24th ACM Symposium on Theory of Computing*.

BABAOĞLU, Ö. 1987. On the Reliability of Consensus-Based Fault-Tolerant Distributed Computing Systems. *ACM Transactions on Computer Systems*, **5**(3), 394–416.

BABAOĞLU, Ö., & DRUMMOND, R. 1985. Streets of Byzantium: Network Architectures for Fast Reliable Broadcasts. *IEEE Transactions on Software Engineering*, **11**(6).

BABAOĞLU, Ö., & TOUEG, S. 1993. Understanding Non-Blocking Atomic Commitement. *Capítulo 6 de:* MULLENDER, S. (ed), *Distributed Systems (2nd edition)*. Addison-Wesley Publishing Company.

BABAOĞLU, Ö., DAVOLI, R., GIACHINI, L.-A., & SABATTINI, P. 1995 (Abril). *The Inherent Cost of Strong-Partial View-Synchronous Communication*. Tech. rept. UBLCS-95-11. University of Bologna.

BAPTISTA, M. 1991 (Julho). *Especificação Formal e Verificação de Protocolos Complexos*. Tese de Mestrado, Instituto Superior Técnico, Universidade Técnica de Lisboa.

BAPTISTA, M., RODRIGUES, L., VERÍSSIMO, P., GRAF, S., RICHIER, J.L., RODRIGUEZ, C., & VOIRON, J. 1991. Formal Specification and Verification of a Network Independent Atomic Multicast Protocol. *Páginas 345–352 de:* QUEMADA, J., MAÑAS, J., & VAZQUES, E. (eds), *FORMAL DESCRIPTION TECHNIQUES, III*. IFIP. North-Holland.

BARBARA, D., & GARCIA-MOLINA, H. 1990 (Novembro). The Case for Controlled Inconsistency in Replicated Data. *Páginas 35–38 de: Proceedings of the IEEE Workshop on the Management of Replicated Data*. Houston, Texas, USA.

BARGHOUTI, N., & KAISER, G. 1991. Concurrency Control in Advanced Database Applications. *ACM Computing Surveys*, **23**(3), 269–317.

BERNSTEIN, P., & GOODMAN, N. 1984. An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases. *ACM Transactions on Database Systems*, **9**(4), 596–615.

BERNSTEIN, P., HADZILACOS, V., & GOODMAN, N. 1987. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Publishing Company.

BERSHAD, B., & ZEKAUSKAS, M. 1991. *Midway: Shared memory paralel programming with entry consistency for distributed memory multiprocessors*. Tech. rept. CMU-CS-91-170. Carnegie-Mellon University.

BIRMAN, K. 1994 (Janeiro). *A Response to Cheriton and Skeen's Criticism of Causal and Totally Ordered Communication*. *ACM Operating Systems Review*, **28**(1), 11–21.

BIRMAN, K., & JOSEPH, T. 1987. Reliable Communication in the Presence of Failures. *ACM Transactions on Computer Systems*, **5**(1).

BIRMAN, K., & JOSEPH, T. 1989. Exploiting replication in distributed systems. *Páginas 319–366 de:* MULLENDER, S. (ed), *Distributed Systems*. Frontier Series. ACM Press.

BIRMAN, K., & RENESSE, R. (eds). 1994. *Reliable Distributed Computing With the ISIS Toolkit*. IEEE Computer Society Press.

BIRMAN, K., SCHIPER, A., & STEPHENSON, P. 1991a. Lightweight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems*, **9**(3).

BIRMAN, K., COOPER, R., & GLEESON, B. 1991b (Janeiro). *Programming with Process Groups: Group and Multicast Semantics*. Tech. rept. TR-91-1185. Cornell University, Ithaca, New York, USA.

BLACK, A., HUTCHINSON, N., JUL, E., LEVY, H., & CARTER, L. 1986. Distribution and Abstract Types in Emerald. *IEEE Transactions on Software Eng., Special Issue on Distributed Computing*, Dezembro

BLOCH, J. 1989 (Junho). The CAMELOT library: A C language extension for programming a general purpose distributed transaction system. *Páginas 172–180 de: Proceedings of the 9th IEEE International Conference on Distributed Computing Systems*. Newport Beach, California, USA.

BORR, A. 1981. Transaction Monitoring in Encompass: Reliable Distributed Transaction Processing. *Em: Proceedings of VLDB*.

CHANDRA, T., & TOUEG, S. 1991 (Julho). *Unreliable Failure Detectors for Asynchronous Systems (Preliminary Version)*. Tech. rept. Department of Computer Science, Cornell University, Ithaca, New York, USA.

CHANG, J., & MAXEMCHUCK, N. 1984. Reliable Broadcast Protocols. *ACM Transactions on Computer Systems*, **2**(3), 251–273.

CHARRON-BOST, B. 1991. Concerning the Size of Logical Clocks in Distributed Systems. *Information Processing Letters*, **39**(1), 11–16.

CHERITON, D. 1992 (Setembro). Problem oriented shared memory revisited. *Em: Proceedings of the 5th ACM SIGOPS European workshop*. Le Mont Saint-Michel, France.

CHERITON, D., & MANN, T. 1989. Decentralizing a Global Naming Service for Improved Performance and Fault Tolerance. *ACM Transactions on Computer Systems*, **7**(2), 147–183.

CHERITON, D., & SKEEN, D. 1993 (Dezembro). Understanding the limitations of causally and totally ordered communication. *Páginas 44-57 de: Proceedings of the 14th ACM Symposium on Operating Systems Principles*.

CHEUNG, S., AMMAR, M., & AHAMAD, M. 1990. The grid protocol: A high performance scheme for maintaining replicated data. *Páginas 438–445 de: Proceedings of the 6th Internation Conference on Data Engineering*.

COOPER, E. 1984 (Agosto). Replicated Procedure Call. *Páginas 220-232 de: Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*.

COPLIEN, J. 1992. *Advanced C++ Programming Styles and Idioms*. Addison-Wesley Publishing Company.

COSQUER, F., & VERÍSSIMO, P. 1994 (Setembro). *Survey of Selected Groupware Applications and Supporting Platforms*. Broadcast Tech. rept. 2nd Year, Vol. 1.

COSQUER, F. 1995 (Agosto). *Cooperative Work over Large Scale Networks: INESC Broadcast Workshop Demo*. Relatório Técnico INESC.

COSQUER, F., RODRIGUES, L., & VERÍSSIMO, P. 1995 (Outubro). Using Tailored Failure Suspectors to Support Distributed Cooperative Applications. *Em: Proceedings of the 7th ISATED/ISMM International Conference on Parallel and Distributed Computing and Systems*. Washington (DC), USA.

CRISTIAN, F., AGHILI, H., STRONG, R., & DOLEV, D. 1985 (Junho). Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement. *Páginas 200-206 de: Digest of Papers, The 15th IEEE International Symposium on Fault-Tolerant Computing*. Ann Arbor, USA.

CRISTIAN, F., DANCEY, R., & DEHN, J. 1990 (Junho). Fault-Tolerance in the Advanced Automation System. *Páginas 6–17 de: Digest of Papers, The 20th IEEE International Symposium on Fault-Tolerant Computing*. Newcastle, UK.

CSMA/CD. 1985. *ISO DIS 8802/3-85, Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*.

DASGUPTA, P., LEBLANC, R., & APPELBE, W. 1988 (Junho). The Clouds Distributed System: Functional Description, Implementation Details and Related Work. *Páginas 2–8 de: Proceedings of the 8th IEEE International Conference on Distributed Computing Systems*. San Jose, California, USA.

DAVCEV, D. 1989. A Dynamic Voting Scheme in Distributed Systems. *IEEE Transactions on Software Engineering*, **15**(1), 93–97.

DEERING, S. 1989 (Agosto). *Host Extensions for IP MUlticasting*. Tech. rept. RFC 1112. Stanford University, Stanford, California, USA.

DEMERS, A., GREENE, D., HAUSTER, C., IRISH, W., LARSON, J., & TERRY, D. 1987 (Agosto). Epidemic Algorithms for Replicated Database Maitenance. *Páginas 1–12 de: Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*.

DIETZEN, S., & SPECTOR, A. 1992. Distributed Transaction Systems. *Em:* CERUTTI, D., & PIERSON, D. (eds), *Distributed Computer Environements*. McGraw-Hill Publishers.

DOLEV, D., MALKI, D., & STRONG, R. 1993a. *An Asynchronous Membership Protocol that Tolerates Partitions*. Tech. rept. The Hebrew University of Jerusalem.

DOLEV, D., KRAMER, S., & MALKI, D. 1993b (Junho). Early Delivery Totally Ordered Multicast in Asynchronous Environments. *Páginas 544–553 de: Digest of Papers, The 23th IEEE International Symposium on Fault-Tolerant Computing*. Toulouse, France.

ELLIS, M., & STROUSTRUP, B. 1990. *The Annotated C++ Reference Manual*. Addison-Wesley Publishing Company.

EVEN, S. 1979. *Graph Algorithms*. Computer Science Press.

EZHILCHELVAN, P., MACEDO, R., & SHRIVASTAVA, S. 1995 (Maio). Newtop: A Fault-Tolerant Group Communication Protocol. *Páginas 296–306 de: Proceedings of the 15th IEEE International Conference on Distributed Computing Systems*. Vancouver, British Columbia, Canada.

FISCHER, M., LYNCH, N., & PATERSON, M. 1985. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the Association for Computing Machinery*, **32**(2), 374–382.

FONSECA, H. 1994 (Junho). *Ambientes de Suporte para a Modularização, Concretização e Execução de Protocolos de Comunicação*. Tese de Mestrado, Instituto Superior Técnico, Universidade Técnica de Lisboa.

FONSECA, H., RODRIGUES, L., RUFINO, J., & VERÍSSIMO, P. 1990 (Agosto). *Local Support Environment: User Specification*. Tech. rept. RT/50-90. INESC, Lisboa, Portugal.

FRAZÃO, J. 1995. *Novos Protocolos para Interligacao de Redes Grande-escala*. Tese de Mestrado, Instituto Superior Técnico, Universidade Técnica de Lisboa. (Em preparação.)

FRIEDMAN, R., & RENESSE, R. 1995 (Março). *Strong and Weak Virtual Synchrony in Horus*. Tech. rept. Department of Computer Science, Cornell University.

GARCIA-MOLINA, H., & BARBARA, D. 1985. How to assign votes in distributed system. *Journal of the ACM*, **32**(4), 841–860.

GARCIA-MOLINA, H., & SPAUSTER, A. 1991. Ordered and Reliable Multicast Communication. *ACM Transactions on Computers Systems*, **9**(3), 242–271.

GARCIA-MOLINA, H., KAO, B., & BARBARA, D. 1991 (Maio). Agressive Transmissions over Redundant Paths. *Páginas 198–207 de: Proceedings of the 11th IEEE International Conference on Distributed Computing Systems*. Arlington, Texas, USA.

GHARACHORLOO, K., LENOSKI, D., LAUDON, J., GIBSONS, P., GUPTA, A., & HENESSY, J. 1990 (Maio). Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. *Páginas 15–26 de: Proceedings of the 17th International Symposium on Computer Architecture*.

GHEZZI, C., JAZAYERI, M., & MANDRIOLI, D. 1991. *Fundamentals of Software Engineering*. Prentice-Hall.

GIFFORD, D. 1979 (Dezembro). Weighted Voting for Replicated Data. *Páginas 150–162 de: Proceedings of the 7th ACM Symposium on Operating System Principles*.

GLADE, B., BIRMAN, K., COOPER, R., & RENESSE, R. 1993. Light-weight process groups in the ISIS system. *Distributed System Engineering*, 29–36.

GOODMAN, J. 1989 (Março). *Cache consistency and sequencial consistency*. Tech. rept. 61. SCI Commitee.

GUEDES, P., & CASTRO, M. 1993 (Outubro). Distributed Shared Object Memory. *Páginas 142–149 de: Proceedings of the 4th IEEE Workshop on Workstation Operating Systems*. Napa, California, USA.

HAGSAND, O., HERZOG, H., BIRMAN, K., & COOPER, R. 1992. Object-oriented Reliable Distributed Programming. *Em: Proceedings of 2nd International Workshop on Object-Orientation in Operating Systems*.

HERLIHY, M., & WING, J. 1990. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, **12**(3), 463–492.

HEYBEY, A. 1990 (Setembro). *The Network Simulator Version 2.1*. Tech. rept. M.I.T.

HILTUNEN, M., & SCHLICHTING, R. 1994a (Agosto). *A Configurable Membership Service*. Tech. rept. TR 94-37. University of Arizona, Department of Computer Science, Tucson, Arizona, USA.

HILTUNEN, M., & SCHLICHTING, R. 1994b (Agosto). *Properties of Membership Services*. Tech. rept. University of Arizona, Department of Computer Science, Tucson, Arizona, USA.

HISGEN, A., BIRREL, A., JERIAN, C., MANN, T., SHROEDER, M., & SWART, G. 1990 (Novembro). Granularity and Semantic Level of Replication in the Echo Distributed File System. *Páginas 2–4 de: Proceedings of the IEEE Workshop on the Management of Replicated Data*. Houston, Texas, USA.

HUTCHINSON, N., & PETERSON, L. 1988 (Agosto). Design of the x-Kernel. *Em: Proceedings of the ACM SIGCOMM'88: Communications Architectures and Protocols*.

JAHANIAN, F., FAKHOURI, S., & RAJKUMAR, R. 1993 (Outubro). Processor Group Membership Protocols: Specification, design and Implementation. *Páginas 2–11 de: Proceedings of the 12th IEEE Symposium on Reliable Distributed Systems*. Princeton, New Jersey, USA.

JOHN, P. 1990. *Statistical Methods in Engineering and Quality Assurance*. John Wiley & Sons Inc.

KAASHOEK, M., & TANENBAUM, A. 1991 (Maio). Group Communication in the Amoeba Distributed Operating System. *Páginas 222–230 de: Proceedings of the 11th IEEE International Conference on Distributed Computing Systems*. Arlington, Texas, USA.

KOPETZ, H., & SCHWABL, W. 1989 (Outubro). *Global Time in Distributed Real-Time Systems*. Tech. rept. 15/89. Technische Universitat Wien, Wien,, Austria.

KUMAR, A., & CHEUNG, S. 1991. A high availability $\sqrt{N}$ hierarquical grid algorithm for replicated data. *Information Processing Letters*, Dezembro, 311–316.

LADIN, R., LISKOV, B., & SHRIRA, L. 1990 (Agosto). Lazy Replication: Exploiting the Semantics of Distributed Services. *Páginas 43–57 de: Proceedings of the 9th Annual ACM Symposium of Principles of Distributed Computing*.

LAMPORT, L. 1978. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM*, **7**(21).

LAMPORT, L. 1979. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, **28**(9), 690–691.

LAMPORT, L., SHOSTAK, R., & PEASE, M. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, **4**(3).

LISKOV, B. 1985. The ARGUS Language and System. *Em: Distributed Systems, Methods and Tools for Specification*. Lecture Notes in Computer Science, vol. 190. Springer-Verlag.

LISKOV, B., & SCHEIFLER, R. 1982 (Janeiro). Guardians and Actions: Linguistic Support for Robust, Distributed Programs. *Páginas 7–19 de: Proceedings of the 9th ACM Symposium on Principles of Programming Languages*.

LISKOV, B., GRUBE, R., JOHNSON, P., & SHRIRA, L. 1990 (Novembro). A Replicated Unix File System. *Páginas 11–14 de: Proceedings of the IEEE Workshop on the Management of Replicated Data*. Houston, Texas, USA.

LITTLE, M. 1991 (Setembro). *Object Replication in a Distributed System*. Ph.D. thesis, Newcastle University, Computer Science Department.

LLOYD, W., & KEARNS, P. 1990. Bounding Sequence Numbers in Distributed Systems: a General Approach. *Páginas 312–319 de: Proceedings of the 10th IEEE International Conference on Distributed Computing Systems*. Paris, France.

MACEDO, R., EZHILCHELVAN, P., & SHRIVASTAVA, S. 1995 (Abril). *Flow Control Schemes for a Fault-Tolerant Multicast Protocol*. Tech. rept. University of Newcastle upon Tyne.

MAFFEIS, S. 1995. *Run-Time Support for Object-Oriented Distributed Programming*. Ph.D. thesis, Fakultat der Universitat Zurich.

MAHLIS, L., SANDERS, W., & SCHLICHTING, R. 1992. *Analytic Performability Evaluation of a Group-Oriented Multicast Protocol*. Tech. rept. University of Arizona.

MAKPANGOU, M., GOURHANT, Y., NARZUL, J.-P., & SHAPIRO, M. 1992. Fragmented Objects for Distributed Abstractions. *Em: Advances in Distributed Systems*. IEEE.

MARZULLO, K., & SCHMUCK, F. 1988 (Junho). Supplying High Availability with a Standard Network File System. *Páginas 447–453 de: Proceedings of the 8th IEEE International Conference on Distributed Computing Systems*.

MATOS, D. 1994 (Setembro). *Técnicas para a Programação de Aplicações Distribuídas e Persistentes num Modelo de Objectos Uniforme e Transparente*. Tese de Mestrado, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, Portugal.

MELDAL, S., SANKAR, S., & VERA, J. 1991. Exploiting Locality in Maitaining Potential Causality. *Páginas 231–239 de: Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*.

MELLIAR-SMITH, P. MICHAEL, & SCHWARTZ, RICHARD L. 1982. Formal Specification and Mechanical Verification of SIFT: A Fault-Tolerant Flight Control System. *IEEE Transactions on Computers*, **31**(7), 616–630.

MINOURA, T., & WIEDERHOLD, G. 1982. Resilient extented true-copy token scheme for a distributed database system. *IEEE Transactions on Software Engineering*, **8**(3), 173–189.

MISHRA, S., PETERSON, L., & SCHLICHTING, D. 1993. Consul: A Communication Substracte for Fault-Tolerant Distributed Programs. *Distributed Systems Engineering*, **1**(2), 87–103.

MOSTEFAOUI, A., & RAYNAL, M. 1993 (Setembro). Causal Multicast in Overlapping Groups: Towards a Low Cist Approach. *Páginas 136–142 de: Proceedings of the 4th IEEE Workshop on Future Trends of Distributed Computing Systems*. Lisboa, Portugal.

MULLENDER, S. 1989. Introduction. *Em:* MULLENDER, S. (ed), *Distributed Systems*. ACM Press - Frontier Series. Addison-Wesley Publishing Company.

MULLENDER, S., ROSSUM, G., TANENBAUM, A., RENESSE, R., & STAVEREN, H. 1990. Amoeba : A Distributed Operating System for the 1990s. *IEEE Computer Magazine*, **23**(Maio).

NEVES, N., CASTRO, M., & GUEDES, P. 1994 (Agosto). A Checkpoint Protocol for an Entry Consistent Shared Memory Model. *Em: Proceedings of the 13th ACM Symposium on Principles of Distributed Computing*.

NITZBERG, B., & LO, V. 1991. Distributed Shared Memory: A Survey of Issues and Algorithms. *IEEE Computer*, **24**(8), 52–60.

ODP. 1987. *Proposed Revised Text for the New Work Item on the Basic Reference Model for Open Distributed Processing*. ISO/TC97/SC21 N1889.

OMG, OBJECT MANAGEMENT GROUP. 1991 (Agosto). *The Common Object Request Broker Architecture and Specification*. OMG.

PARIS, J.F., & SLOOPE, P. 1992 (Outubro). Dynamic Management of Highly Replicated Data. *Páginas 20–28 de: Proceedings of the 11th IEEE Symposium on Reliable Distributed Systems*.

PEREIRA, J. 1994. *Enriquecimento com Controlo de Concorrência de Especificações Orientadas por Objectos*. Tese de Mestrado, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, Portugal.

PETERSON, L., BUCHHOLZ, N., & SCHLICHTING, R. 1989. Preserving and Using Context Information in Interprocess Communication. *ACM Transactions on Computer Systems*, **7**(3).

POWELL, D. (ed). 1991. *Delta-4 - A Generic Architecture for Dependable Distributed Computing*. ESPRIT Research Reports. Springer Verlag.

POWELL, D. 1994. Distributed Fault Tolerance: Lessons from Delta-4. *IEEE Micro*, Fevereiro, 36–47.

PU, C., NOE, J., & PROUDFOOT, A. 1988. Regeneration of Replicated Objects, a technique and its Eden Implementations. *IEEE Transactions on Software Engineering*, **14**(7), 936–945.

RAYNAL, M., SCHIPER, A., & TOUEG, S. 1991. The causal ordering abstraction and a simple way to implement it. *Information processing letters*, **39**(6), 343–350.

RDO. 1993. *Preliminary User Documentation for RDO/C++, Release 1.0.2*.

RENESSE, R. 1993. Causal Controversy at Le Mont St.-Michel. *ACM Operating Systems Review*, **27**(2), 44–53.

RENESSE, R., BIRMAN, K., COOPER, R., GLADE, B., & STEPHENSON, P. 1992 (Abril). Reliable Multicast between Microkernels. *Páginas 269–283 de: Proceedings of the USENIX Workshop on Micro-Kernels and Other Architectures*.

RICCIARDI, A., & BIRMAN, K. 1991 (Agosto). Using Process Groups to Implement Failure Detection in Asynchronous Environemnts. *Páginas 341–351 de: Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*.

RICCIARDI, A., SCHIPER, A., & BIRMAN, K. 1993 (Setembro). Understanding Partitions and the No Partition Assumption. *Páginas 354–360 de: Proceedings of the 4th IEEE Workshop on Future Trends of Distributed Computing Systems*. Lisboa, Portugal.

RODRIGUES, L. 1990 (Outubro). *Mecanismos de comunicação eficientes para sistemas de tempo-real e tolerantes a faltas*. Tese de Mestrado, Instituto Superior Técnico, Universidade Técnica de Lisboa.

RODRIGUES, L. 1994. *A light-weight group mechanism for NavTech (Short memo)*. Tech. rept. INESC.

RODRIGUES, L., & VERÍSSIMO, P. 1992a (Março). *A posteriori Agreement for Clock Synchronization on Broadcast Networks*. Tech. rept. RT/62-92. INESC.

RODRIGUES, L., & VERÍSSIMO, P. 1992b (Outubro). xAMp: a Multi-primitive Group Communications Service. *Páginas 112–121 de: Proceedings of the 11th IEEE Symposium on Reliable Distributed Systems*. Houston, Texas, USA.

RODRIGUES, L., & VERÍSSIMO, P. 1993a (Setembro). Replicated Object Management Using Group Technology. *Páginas 54–61 de: Proceedings of the 4th IEEE Workshop on Future Trends of Distributed Computing Systems*. Lisboa, Portugal.

RODRIGUES, L., & VERÍSSIMO, P. 1993b. *The ROMANCE Approach to Replicated Object Management*. Tech. rept. TR/57-93. INESC, Lisboa, Portugal.

RODRIGUES, L., & VERÍSSIMO, P. 1994 (Setembro). How to avoid the cost of causal communication in large-scale systems. *Páginas 106–111 de: Proceedings of the 6th ACM-SIGOPS Europe Workshop*.

RODRIGUES, L., & VERÍSSIMO, P. 1995a (Fevereiro). *Causal separators and topological timestamping: an approach to support causal multicast in large-scale systems* Tech. rept. TR/5-95. INESC, Lisboa, Portugal.

RODRIGUES, L., & VERÍSSIMO, P. 1995b (Maio). Causal separators for large-scale multicast communication. *Páginas 83–91 de: Proceedings of the 15th IEEE International Conference on Distributed Computing Systems*. Vancouver, British Columbia, Canada.

RODRIGUES, L., VERÍSSIMO, P., & RUFINO, J. 1993a (Maio). A low-level processor group membership protocol for LANs. *Páginas 541–550 de: Proceedings of the 13th IEEE International Conference on Distributed Computing Systems*. Pittsburgh, Pennsylvania, USA.

RODRIGUES, L., VERÍSSIMO, P., & CASIMIRO, A. 1993b (Outubro). Using atomic broadcast to implement *a posteriori* agreement for clock synchronization. *Páginas 115–124 de: Proceedings of the 12th IEEE Symposium on Reliable Distributed Systems*. Princeton, New Jersey, USA.

RODRIGUES, L., FONSECA, H., & VERÍSSIMO, P. 1994. *A dynamic hybrid protocol for total order in large-scale systems*. Tech. rept. TR/34-94. INESC, Lisboa, Portugal.

RODRIGUES, L., CASIMIRO, A., & VERÍSSIMO, P. 1995a (Maio). Priority-based totally ordered multicast. *Páginas 375–383 de: Proceedings of the 3rd IFAC/IFIP workshop on Algorithms and Architectures for Real-Time Control (AARTC'95)*.

RODRIGUES, L., FONSECA, H., & VERÍSSIMO, P. 1995b (Agosto). Reliable Computing over Mobile Networks. *Páginas 488-494 de: Proceedings of the 5th IEEE Workshop on Future Trends of Distributed Computing Systems*.

RUFINO, J. 1993 (Abril). *Rede de Difusão Abstracta: um modelo para construção de protocolos portáteis sobre rede local*. Tese de Mestrado, Instituto Superior Técnico, Universidade Técnica de Lisboa. Cheju Island, Korea.

RUSINKIEWICK, M., SHETH, A., & KARABATIS, G. 1991. Specifying Interdatabase Dependencies in a Multidatabase Environment. *IEEE Computer*, **24**(12), 46,53.

SANDERS, R. 1990 (Janeiro). *The Xpress Transfer Protocol (XTP)- A Tutorial*. Tech. rept. Computer Science Report TR-89-10. University of Virginia.

SARGENTO, A. 1995. *Filiação em redes de grande escala*. Tese de Mestrado, Instituto Superior Técnico, Universidade Técnica de Lisboa. (Em preparação.)

SCHIPER, A., & RICCIARDI, A. 1993 (Junho). Virtually-Synchronous Communication Based on a Weak Failure Suspector. *Páginas 534–543 de: Digest of Papers, The 23th IEEE International Symposium on Fault-Tolerant Computing*. Toulouse, France.

SCHIPER, A., & SANDOZ, A. 1993 (Maio). Uniform Reliable Multicast in a Virtually Synchronous Environement. *Páginas 561–568 de: Proceedings of the 13th IEEE International Conference on Distributed Computing Systems*.

SCHIPER, A., & SANDOZ, A. 1994 (Setembro). Primary Partition Virtually-Synchronous Communication harder than Consensus. *Páginas 38–52 de: Proceedings of the 8th International Workshop on Distributed Algorithms (WDAG-94)*.

SCHNEIDER, F. 1987 (Agosto). *Understanding protocols for byzantine clock synchronization*. Tech. rept. Cornell University, Ithaca, New York, USA.

SCHNEIDER, F. 1990. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys*, **22**(4), 290–319.

SCHNEIDER, F., GRIES, D., & SCHLICHTING, R. 1984. Fault-tolerant broadcasts. *Science of Computer Programming*, 1–15.

SCHWARZ, A., & MATTERN, F. 1991. *Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail*. Tech. rept. Departement of Computer Science, University of Kaiserlautern.

SHAPIRO, M. 1989. SOS: An Object-Oriented Operating System - Assessment and Perspectives. *Computing Systems*, **2**(4), 287–338.

SHRIVASTAVA, S., DIXON, G., & PARRINGTON, G. 1991. An Overview of the Arjuna Distributed Programming System. *IEEE Software*, Janeiro

SILVA, A., PEREIRA, J., & MARQUES, J. 1995a (Julho). *A Framework for Heterogeneous Concurrency Control Policies in Distributed Applications*. Tech. rept. IST/INESC.

SILVA, A., SOUSA, P., & MARQUES, J. 1995b (Dezembro). Development of Distributed Applications with Separation of Concerns. *Em: Proceedings of the 1995 IEEE Asia-Pacific Software Engineering Conference (APSEC'95)*. Brisbane, Australia.

SINGHAL, M., & KSHEMKALYANI, A. 1990 (Outubro). *An Efficient Implementation of Vector Clocks*. Tech. rept. Ohio State University, Departement of Computer and Information Science.

SKEEN, D. 1982 (Maio). *Crash recovery in a distributed database system*. Ph.D. thesis, University of California at Berkeley.

SKEEN, D. 1985. Determining the Last Process to Fail. *ACM Transactions on Computer Systems*, **3**(1).

SKEEN, D., & STONEBRAKER, M. 1983. A Formal Model of Crash Recovery in a Distributed System. *IEEE Transactions on Software Engineering*, **9**(3), 219–228.

SOUSA, P., SEQUEIRA, M., ZÚQUETE, A., FERREIRA, P., LOPES, C., GUEDES, P., & MARQUES, J. 1993. The IK Distributed and Persistent Platform: Overview and Evaluation. *Usenix Computing Systems*, **6**(4).

SPECTOR, A., DANIELS, D., DUCHAMP, D., EPPINGER, J., & PAUSCH, R. 1985 (Dezembro). Distributed Transactions for Reliable Systems. *Páginas 127–146 de: Proceedings of the 10th ACM Symposium on Operating Systems Principles*.

STEPHENSON, P. 1991 (Fevereiro). *Fast Causal Multicast*. Ph.D. thesis, Cornell University.

STONEBRAKER, M. 1979. Concurrency Control and Consistency of Multiple Copies in Distributed INGRES. *IEEE Transactions on Software Engineering*, **5**(3), 188–194.

TANENBAUM, A. 1989. *Computer Networks*. Second edition. Prentice-Hall.

TANENBAUM, A., KAASHOEK, M., & BAL, H. 1992. Parallel Programming Using Shared Objects and Broadcasting. *IEEE Computer*, **25**(8), 10–19.

THOMAS, R. 1979. A Majority Concensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transactions on Database Systems*, **4**(2), 180–209.

TOKEN-BUS. 1985. *ISO DIS 8802/4-85, Token Passing Bus Access Method and Physical Layer Specifications*.

TOKEN-RING. 1985. *ISO DP 8802/5-85, Token Ring Access Method and Physical Layer Specifications*.

TRANCOSO, P., & SEQUEIRA, M. 1993 (Março). *INESC's CORBA IDL Compiler*. Tech. rept. HARNESS/PRI/WP/INE/201/1.0. INESC/ ESPRIT Project 5279.

VERÍSSIMO, P. 1989 (Dezembro). *Comunicação em Grupo Fiável, em Sistemas Distribuidos sobre Rede Local*. Tese de Doutoramento, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, Portugal.

VERÍSSIMO, P. 1995. Causal Delivery Protocols in Real-time Systems: a Generic Model. *Journal of Real-Time Systems*.

VERÍSSIMO, P., & RODRIGUES, L. 1992a (Abril). Group Orientation: a Paradigm for Distributed Systems of the Nineties. *Páginas 57–63 de: Proceedings of the 3rd IEEE Workshop on Future Trends of Distributed Computing Systems*. Taipe, Taiwan.

VERÍSSIMO, P., & RODRIGUES, L. 1992b (Julho). A posteriori Agreement for Fault-tolerant Clock Synchronization on Broadcast Networks. *Páginas 115–124 de: Digest of Papers, The 22nd IEEE International Symposium on Fault-Tolerant Computing*. Boston, USA.

VERÍSSIMO, P., & RODRIGUES, L. 1995 (Agosto). *The NavTech Large-Scale Distributed Computing Platform*. Tech. rept. INESC, Lisboa, Portugal. (Em preparação.)

VERÍSSIMO, P., & VOGELS, W. 1993 (Setembro). The Changing Face of Technology in Distributed Systems. *Páginas 119–127 de: Proceedings of the 4th IEEE Workshop on Future Trends of Distributed Computing Systems*. Lisboa, Portugal.

VERÍSSIMO, P., RODRIGUES, L., & BAPTISTA, M. 1989 (Setembro). AMp: A Highly Parallel Atomic Multicast Protocol. *Páginas 83–93 de: Proceedings of the ACM SIGCOM'89 Symposium*. Austin, Texas, USA.

VERÍSSIMO, P., RODRIGUES, L., & CASIMIRO, A. 1995 (Março). *CesiumSpray: a Precise and Accurate Global Clock Service for Large-scale Systems*. Tech. rept. INESC, Lisboa, Portugal.

VOGELS, W., RODRIGUES, L., & VERÍSSIMO, P. 1992 (Novembro). Fast Group Communication for Standard Workstations. *Páginas 337-363 de: Proceedings of the OpenForum'92 Technical Conference*. EurOpen, UniForum, Utrecht, the Netherlands.

WILHELM, U., & SCHIPER, A. 1995 (Outubro). A hierarchy of totally ordered multicasts. *Em: Proceedings of the 14th IEEE Symposium on Reliable Distributed Systems*.

WOOD, M. 1993 (Maio). Replicated RPC Using Amoeba Closed Group Communication. *Páginas 499–507 de: Proceedings of the 13th IEEE International Conference on Distributed Computing Systems*. Pittsburgh, Pennsylvania, USA.

Índice Remissivo