Universidade do Minho
Escola de Engenharia
Departamento de Informática

$Ro_cme\mu$: orientação ao recurso na modelação de aplicações paralelas e exploração cooperativa de *clusters* multi-SAN

por

Albano Agostinho Gomes Alves

Dissertação apresentada à Universidade do Minho para
a obtenção do grau de Doutor em Informática

Orientador:
Prof. António Manuel Silva Pina

Guimarães
Junho de 2004

À Guida, à Matilde e ao Miguel.

Agradecimentos

Quero agradecer ao meu orientador, Prof. António Pina, pela sua orientação, disponibilidade e confiança que tem depositado em mim. Ao longo destes quatro anos, foram muitas as conversas que mantivemos e não posso deixar de mostrar o meu apreço pela possibilidade que sempre me foi dada de expor e confrontar, sem quaisquer limitações, as minhas ideias. À pessoa que sempre esteve disposta a ouvir e, nos momentos certos, soube mostrar que eu estava errado, o meu sincero agradecimento.

Um agradecimento especial aos que acompanharam mais de perto o meu trabalho e com quem pude trocar impressões e debater importantes questões técnicas, nomeadamente ao Rufino, ao Exposto e ao Prof. Joaquim Macedo.

Agradeço também ao Instituto Politécnico de Bragança, particularmente à Escola Superior de Tecnologia e de Gestão, onde me foram dadas todas as condições para desenvolver a minha actividade académica.

Não posso deixar de realçar todo o apoio e compreensão que recebi da Guida, que conseguiu ser mãe e pai de dois bebés gémeos nos momentos em que este trabalho mais exigiu de mim. Agradeço ainda aos meus pais, aos meus sogros e aos meus amigos.

Resumo

O desenvolvimento de soluções paralelas para problemas com requisitos computacionais elevados tem estado limitado à exploração de sistemas de computação específicos e à utilização de abstracções altamente conotadas com a arquitectura desses sistemas. Estes condicionalismos têm um impacto altamente desencorajador na utilização de *clusters* heterogéneos – que integram múltiplas tecnologias de interligação – quando se pretende dar respostas capazes, tanto ao nível da produtividade, como do desempenho.

Esta dissertação apresenta a orientação ao recurso como uma nova abordagem à programação paralela, unificando no conceito de recurso as entidades lógicas dispersas pelos nós de um *cluster*, criadas pelas aplicações em execução, e os recursos físicos que constituem o potencial de computação e comunicação da arquitectura alvo. O paradigma introduz novas abstracções para *(i)* a comunicação entre recursos lógicos e *(ii)* a manipulação de recursos físicos a partir das aplicações. As primeiras garantem um interface mais conveniente ao programador, sem comprometerem o desempenho intrínseco das modernas tecnologias de comunicação SAN. As segundas permitem que o programador estabeleça, explicitamente, uma correspondência efectiva entre as entidades lógicas e os recursos físicos, por forma a explorar os diferentes padrões de localidade existentes na hierarquia de recursos que resulta da utilização de múltiplas tecnologias SAN e múltiplos nós SMP.

O paradigma proposto traduz-se numa metodologia de programação concretizada na plataforma $m_{\varepsilon}\mu$, que visa a integração do desenho/desenvolvimento de aplicações paralelas e do processo de selecção/alocação de recursos físicos em tempo de execução, em ambientes multi-aplicação e multi-utilizador. Na base desta plataforma está o R_OCl, uma outra plataforma, desenvolvido com o intuito de oferecer uma imagem de sistema uno. Na arquitectura resultante, o primeiro nível, suportado pelo R_OCl, garante a conectividade entre recursos lógicos dispersos pelos diferentes nós do *cluster*, enquanto o segundo, da responsabilidade do $m_{\varepsilon}\mu$, permite a organização e manipulação desses recursos lógicos, a partir de uma especificação inicial, administrativa, dos recursos físicos disponíveis.

Do ponto de vista da programação paralela/distribuída, o $m_{\varepsilon}\mu$ integra adaptações e extensões dos paradigmas da programação por memória partilhada, passagem de mensagens e memória global. Numa outra vertente, estão disponíveis capacidades básicas para a manipulação de recursos físicos em conjunto com facilidades para a criação e localização de entidades que suportam a interoperabilidade e a cooperação entre aplicações.

Abstract

The development of parallel solutions for high demanding computational problems has been limited to the exploitation of specific computer systems and to the use of abstractions closely related to the architecture of these systems. These limitations are a strong obstacle to the use of heterogeneous clusters – clusters that integrate multiple interconnection technologies – when we intend to give capable answers to both productivity and performance.

This work presents the resource orientation as a new approach to parallel programming, unifying in the resource concept the logical entities spread through cluster nodes by applications and the physical resources that represent computation and communication power. The paradigm introduces new abstractions for *(i)* the communication among logical resources and *(ii)* the manipulation of physical resources from applications. The first ones guarantee a more convenient interface to the programmer, without compromising the intrinsic performance of modern SAN communication technologies. The second ones allow the programmer to explicitly establish the effective mapping between logical entities and physical resources, in order to exploit the different levels of locality that we can find in the hierarchy of resources that results from using distinct SAN technologies and multiple SMP nodes.

The proposed paradigm corresponds to a programming methodology materialized in the $m_{\varepsilon}\mu$ platform, which aims to integrate the design/development of parallel applications and the process of selecting/allocating physical resources at execution time in multi-application, multi-user environments. The basis for this platform is R_OCl, another platform, developed to offer a single system image. The first layer of the resultant architecture, which corresponds to R_OCl, guarantees the connectivity among logical resources instantiated at different cluster nodes, while the second, corresponding to $m_{\varepsilon}\mu$, allows to organize and manipulate these logical resources, starting from an initial administrative specification of the available physical resources.

In the context of parallel/distributed programming, $m_{\varepsilon}\mu$ integrates adaptations and extensions to the shared memory, message passing and global memory programming paradigms. Basic capabilities for the manipulation of physical resources along with facilities for the creation and discovery of entities that support the interoperability and cooperation between applications are also available.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A computação baseada em *clusters* tem vindo a assumir um papel preponderante na construção de aplicações paralelas que apresentem níveis elevados de desempenho. O aumento do poder de cálculo dos nós de um *cluster*, por via da vulgarização das máquinas SMP (Symmetric Multi-Processor), e a diminuição dos tempos de comunicação entre nós, por via do aparecimento das redes SAN (System Area Networks), fez com que os *clusters* se viessem a transformar nos mais poderosos supercomputadores actuais.

Do ponto de vista do hardware, salvo raras excepções, um *cluster* é idealizado como um conjunto de nós homogéneos interligados por uma única tecnologia de comunicação de elevado desempenho. No entanto, a aquisição faseada dos equipamentos usados na construção de um grande *cluster* ou a actualização parcial dos nós de um *cluster* existente levam à perca da homogeneidade inicial, principalmente devido à constante evolução das tecnologias de comunicação.

A possibilidade de se utilizarem múltiplas tecnologias de interligação num *cluster* traduz-se em dois cenários possíveis: *(i)* existência de múltiplas tecnologias de comunicação por máquina (nós multi-interface) e *(ii)* criação de *subclusters*, por via da partição da totalidade dos nós de acordo com as diferentes tecnologias disponíveis. A criação de vários *subclusters*, um por cada tecnologia, e a interligação destes através de nós multi-interface conduzem a um *cluster* heterogéneo, do ponto de vista da comunicação, aqui denominado de *cluster* multi-SAN. No caso de os nós do *cluster* serem máquinas SMP, utiliza-se a designação *cluster* SMP multi-SAN.

Um *cluster* SMP multi-SAN constitui um sistema hierárquico, onde podem ser identificados três níveis de paralelismo: interprocessador (intranó), internó (*intrasubcluster*) e *intersubcluster*. A exploração eficiente deste tipo de sistemas obriga à utilização de metodologias específicas, que possibilitem explorar os diversos níveis de localidade presentes.

Do ponto de vista do software, os programadores codificam algoritmos paralelos para tirar

partido da totalidade dos recursos físicos (hardware) do *cluster*, de maneira a minimizar o tempo necessário para efectuar um determinado cálculo científico ou de engenharia. No entanto, sistemas de informação complexos apontam no sentido da integração, numa vertente cooperativa, de múltiplos componentes, potencialmente desenvolvidos em diferentes estágios e executados por diferentes utilizadores. Nestes casos, vários programadores contribuem para o desenvolvimento de um sistema de aplicações, não sendo possível, na construção de um dado componente, assumir a disponibilidade da totalidade dos recursos do *cluster*.

Ao contrário de uma aplicação codificada para a resolução de um dado problema de cálculo, a qual se espera que termine no espaço de tempo mais curto possível, produzindo os resultados desejados, uma classe importante de aplicações, como por exemplo um sistema de indexação *Web*, executa de forma permanente. O carácter permanente de um sistema de aplicações, ou de uma simples aplicação, reforça a ênfase na disponibilidade parcial dos recursos do *cluster*; os recursos devem poder ser partilhados no tempo e no espaço, de modo a que várias aplicações possam coexistir. A execução em lotes, tradicionalmente utilizada na exploração de *clusters*, não se coaduna com a noção de execução permanente.

Tradicionalmente, o suporte ao desenvolvimento de aplicações para ambientes *cluster* passa pela concepção de uma arquitectura de, pelo menos, duas camadas. A camada de baixo-nível uniformiza o acesso ao hardware variado do *cluster* e permite um controlo estrito da forma como são utilizados os recursos físicos, oferecendo, nomeadamente, meios para selecção e alocação de: tecnologias de comunicação, sistemas de armazenamento, máquinas, processadores, etc. A camada de alto-nível oferece uma imagem de sistema uno, materializada numa biblioteca de programação e num sistema de apoio à execução, escondendo toda a complexidade associada à selecção e alocação de recursos.

1.1 Identificação do problema

Ultimamente, alguns autores têm chamado a atenção para as vantagens de se usarem metodologias específicas dos sistemas NUMA (Non Uniform Memory Access) na programação de *clusters* SMP. Apareceram também alguns sistemas de comunicação que integram múltiplas tecnologias SAN. No entanto, a programação de *clusters* SMP multi-SAN, de forma integrada, não tem sido alvo da atenção dos investigadores.

Habitualmente, o programador dispõe de uma imagem de sistema uno poderosa, mas que não permite explorar as várias localidades presentes num *cluster* SMP multi-SAN. Deste modo, é vulgar assumir-se que um *cluster* encerra um alto nível de homogeneidade e que deve ser visto como uma máquina com um grande número de processadores.

Numa outra dimensão, as abstracções oferecidas pelas plataformas tradicionais são alta-

mente dependentes do hardware do sistema de computação alvo e, como tal, são pouco convenientes para o programador. No entanto, a concepção de novas camadas, como acréscimo às arquitecturas destas plataformas, com o intuito de garantir níveis de abstracção mais elevados, tem um impacto negativo no desempenho, ou seja, a eficiência é sacrificada em prol da conveniência.

Neste contexto, afigura-se essencial a definição de novas abordagens para a exploração de *clusters*, que permitam atingir, simultaneamente, os dois objectivos seguintes:

- exploração eficiente de *clusters* que integrem máquinas multiprocessador e múltiplas tecnologias de interligação de elevado desempenho;
- programação de aplicações através de paradigmas bem conhecidos, enriquecidos com facilidades para a selecção/alocação dinâmica de recursos físicos, em ambientes multi--aplicação e multi-utilizador, numa perspectiva cooperativa.

1.2 Contribuições

A abordagem aqui exposta, para a exploração de *clusters* SMP multi-SAN, parte do princípio que o programador tem interesse no conhecimento das particularidades do sistema de computação e que, com base nesse conhecimento, as aplicações são desenvolvidas de forma a explorar mais eficientemente os recursos físicos disponíveis. Assim, a camada de alto-nível do sistema de exploração expõe os mecanismos de selecção e alocação de componentes hardware, não tendo, portanto, a pretensão de fornecer uma imagem de sistema uno. A camada de baixo-nível deixa, deste modo, de garantir os meios para a selecção e alocação de recursos, passando antes a assegurar uma imagem de sistema uno minimalista (de baixo-nível). Esta imagem assegura que a possível inabilidade do programador, no que diz respeito ao manuseamento das particularidades do sistema de computação, não compromete o funcionamento das aplicações, embora sem garantias de níveis elevados de desempenho.

Os resultados alcançados ao longo deste trabalho de investigação traduzem-se numa nova metodologia que integra, através do paradigma da orientação ao recurso, a modelação de aplicações e a exploração cooperativa do hardware, sem comprometer os níveis de desempenho característicos dos *clusters* SMP multi-SAN.

Em concreto foram criados dois novos modelos, um para a comunicação e outro para a computação em *clusters*, e desenvolvidas duas bibliotecas para a exploração desses modelos. O primeiro introduz o conceito de comunicação orientada ao recurso, baseando-se na definição de entidades comunicantes genéricas e na exploração de um sistema de directório de baixo-nível próprio. O segundo, destinado à programação de aplicações pa-

ralelas cooperativas, introduz uma metodologia de programação que visa a integração do desenho/desenvolvimento de aplicações paralelas e do processo de selecção/alocação de recursos físicos em tempo de execução, em ambientes multi-aplicação e multi-utilizador.

Os principais resultados obtidos em cada etapa deste trabalho foram sendo sucessivamente publicados e validados em conferências internacionais. Os artigos publicados, que traduzem a cronologia das contribuições científicas mais relevantes, são os seguintes:

1. CoR's Faster Route over Myrinet [Pina 00];
   ↪ Primeira abordagem à exploração de uma tecnologia de comunicação de elevado desempenho numa plataforma de programação com um elevado nível de abstracção.

2. Scalable Multithreading in a Low Latency Myrinet Cluster [Alves 02b];
   ↪ Adequação de um protótipo de passagem de mensagens ao desenvolvimento de aplicações baseadas em múltiplos fios-de-execução.

3. High Performance Multithreaded Message Passing on a Myrinet Cluster [Alves 02a];
   ↪ Concepção de sistemas de despacho de mensagens baseados em múltiplos fios-de--execução e definição de políticas para interacção com subsistemas de comunicação.

4. ToCL: a Thread Oriented Communication Library to Interface VIA and GM Protocols [Alves 03c];
   ↪ Definição de uma arquitectura e de um interface de comunicação para exploração conjunta dos protocolos VIA e GM.

5. RoCL: a Resource oriented Communication Library [Alves 03b];
   ↪ Introdução do conceito de comunicação orientada ao recurso e apresentação da biblioteca R_OCl.

6. Evaluating Applications Performance in a Multi-networked Cluster [Alves 03a];
   ↪ Avaliação exaustiva do desempenho da biblioteca R_OCl.

7. Mapping application-level components into hierarchical system resources [Alves 04b];
   ↪ Definição de metodologias e modelos para a representação de recursos físicos, criação de entidades lógicas e correspondência entre ambos.

8. Deploying Applications in Multi-SAN SMP Clusters [Alves 04a];
   ↪ Apresentação de uma metodologia para programação de aplicações em *clusters* SMP multi-SAN.

9. $m_\varepsilon\mu$: unifying application modeling and cluster exploitation [Alves 04c].
   ↪ Apresentação da biblioteca $m_\varepsilon\mu$ e do conceito de recurso como abstracção para a unificação da modelação de aplicações e exploração de *clusters*.

1.3 Organização da dissertação

Os artigos associados a este trabalho, atrás enumerados, constituem uma alternativa a esta dissertação, para aqueles que, através de uma leitura rápida, pretendem tomar conhecimento das principais ideias preconizadas. No que concerne à dissertação, a organização adoptada tem como propósito uma apresentação de conceitos mais coerente e integradora, remetendo para segundo plano a cronologia dos trabalhos.

O capítulo 2 contextualiza o trabalho desenvolvido e mostra o posicionamento do autor face à problemática da computação baseada em *clusters*. São apresentados alguns conceitos importantes à compreensão do restante texto, é feito um levantamento dos trabalhos que, decididamente, marcaram algumas das opções efectuadas ao longo do percurso que conduziu à escrita desta dissertação e, finalmente, conclui-se com a constatação de que existem carências ao nível da exploração cooperativa de *clusters* SMP multi-SAN.

O capítulo 3 apresenta, na sua globalidade, a metodologia de programação de aplicações preconizada e o modelo de computação orientada ao recurso subjacente. São expostos, sucintamente, os mecanismos de selecção/alocação de recursos físicos em tempo de execução e a sua integração com o processo de modelação da aplicação. No capítulo 4 dá-se continuidade à exposição efectuada no capítulo anterior, apresentando a forma como os paradigmas da programação por memória partilhada, passagem de mensagens e memória global são suportados no âmbito da computação orientada ao recurso. Mostra-se ainda a conveniência da utilização do $m_{\varepsilon}\mu$, no que diz respeito à programação de aplicações de acordo com metodologias e técnicas convencionais

O capítulo 5 apresenta a biblioteca R_OCl e o modelo de comunicação orientada ao recurso. São explicados os vários componentes do sistema e é efectuada uma breve exposição do interface disponível para o programador. No capítulo 6 são apresentados detalhes de implementação relativos à exploração das tecnologias Myrinet e Gigabit.

O capítulo 7 pode ser visto como um complemento aos capítulos 3 e 4. É apresentado o interface da biblioteca $m_{\varepsilon}\mu$, que materializa as metodologias e modelos apresentados nos capítulos 3 e 4, e explica-se de que forma a funcionalidade do R_OCl serve de base a esta biblioteca.

No capítulo 8 analisa-se o desempenho do $m_{\varepsilon}\mu$, sendo também apresentados alguns modelos de avaliação próprios. Há ainda a preocupação de mostrar de que forma o sistema se comporta em duas aplicações concretas.

No último capítulo são referidas as principais contribuições deste trabalho e são apontadas algumas direcções para trabalho futuro.

Capítulo 2

Enquadramento

O uso de *clusters*, para execução de aplicações paralelas/distribuídas, tem vindo a aumentar, especialmente devido à vulgarização das estações de trabalho multiprocessador e das tecnologias de comunicação de elevado desempenho. O crescente interesse nos *clusters* esteve na origem da criação da Computer Society Task Force on Cluster Computing (TFCC), em 1999, por parte do IEEE, a qual produziu um *white paper* [Baker 00].

De acordo com a lista TOP500 [TOP500 04], revista em Junho de 2004, de entre os 500 supercomputadores mais poderosos, 291 são *clusters*, enquanto que em 2001 se contavam apenas 43 e em 1999 tão somente 7. Em consonância com os fortes desenvolvimentos ao nível do hardware de suporte à computação baseada em *clusters*, uma grande variedade de componentes de software foram redesenhados e melhorados, por forma a garantir níveis elevados de desempenho e disponibilidade.

Neste capítulo é feita uma síntese do estado da arte, de acordo com as seguintes vertentes: interligação de nós, gestão de recursos e paradigmas de programação. Fundamentalmente, é apresentada uma visão própria da computação baseada em *clusters*, tendo por objectivo enquadrar o trabalho desenvolvido, nomeadamente no que diz respeito à identificação dos problemas que motivaram a investigação subjacente a esta dissertação.

2.1 Interligação de nós

O desempenho de um sistema *cluster* depende, em larga medida, da largura de banda e do tempo de resposta das tecnologias usadas na interligação de nós. Uma elevada largura de banda e um reduzido tempo de resposta são apenas alcançáveis através de hardware de comunicação apropriado e de protocolos e modelos de comunicação eficientes.

2.1.1 Tecnologias de comunicação

A simples interligação de máquinas – nós de um *cluster* – através de redes de comunicação tradicionais, como por exemplo Fast Ethernet, não permite obter os desempenhos necessários à generalidade das aplicações paralelas. No entanto, com a utilização de novas tecnologias de comunicação, torna-se inclusivamente possível executar, em ambiente *cluster*, aplicações com um elevado rácio comunicação/computação [Langendoen 98].

A interligação de nós, num *cluster* moderno, faz-se por meio de comutadores de elevado desempenho, em virtude de se ter concluído que as tecnologias baseadas em barramentos não eram escaláveis. Esta constatação levou a equipa responsável pelo projecto Futurebus a desenvolver a tecnologia SCI (Scalable Coherent Interface) [Dolphin 96], a qual permite que os nós de um *cluster* partilhem partes do seu espaço de endereçamento físico. A ligação de nós aos comutadores faz-se por controladores de rede, desenhados para barramentos PCI (64bits, 66MHz) ou PCI-X (64bits, 133MHz), que dispõem de processadores próprios para libertar os processadores das máquinas para computação útil.

Os comutadores usados em redes *cluster* têm sido alvo de grandes desenvolvimentos, a partir das redes Clos, por forma a conseguir a interligação de um número cada vez maior de nós [Seitz 01]. Essencialmente, têm sido apresentadas soluções com o intuito de eliminar, do processo de desenvolvimento de software, a problemática da topologia. De facto, a utilização de múltiplos comutadores num *cluster*, criando topologias complexas, implica considerações particulares ao nível das bibliotecas de comunicação, de maneira a explorar em condições óptimas o hardware disponível [Etsion 99, Tam 00].

Uma das tecnologias de referência usada, actualmente, para a interligação dos nós de um *cluster*, é a tecnologia Myrinet [Boden 95], que permite alcançar débitos da ordem dos 2Gbits/s, em cada sentido, entre dois nós, e tempos de resposta de cerca de 6.3μs, no envio de mensagens de tamanho reduzido. Dado o seu elevado potencial, a tecnologia Myrinet foi a seleccionada pelo CERN para a construção de um sistema de aquisição de dados num acelerador de partículas [Meijers 02].

Alternativas mais económicas, mas ainda com bons níveis de desempenho, passam pela utilização de redes Gigabit Ethernet, desde que exploradas através de software conveniente, conforme se pode constatar em [Farrell 00a]. Nesta tecnologia, é necessário ter em conta a variedade de oferta, quer ao nível dos comutadores quer ao nível dos controladores, devido à existência de numerosos fabricantes. Em [Betz 02] é efectuada uma comparação sustentada de diversos controladores Gigabit, de diferentes fabricantes, concluindo-se que os SysKonnect SK9821 são os que apresentam o melhor desempenho. Estes controladores foram também os seleccionados para experiências levadas a cabo no CERN [Meijers 03].

Em cenários reais de execução de algumas aplicações, a tecnologia Gigabit Ethernet provou

ser uma alternativa à altura da Myrinet, desde que sejam usados comutadores (Gigabit) de qualidade superior [Chen 00]. Acresce ainda que, desde 2002, existe uma norma para desenvolver tecnologia 10Gigabit Ethernet, a qual poderá vir a ter um impacto importante na construção de sistemas de elevado desempenho [10GEA 02].

Outras tecnologias relevantes, no contexto da interligação de *clusters*, são a Sun Fire Link Interconnect [Sistare 02], com débitos da ordem dos 18Gbits/s (entre máquinas específicas da SUN), a Infiniband [IBTA 02], uma norma para a interligação de nós computacionais e sistemas de armazenamento, para a qual a Mellanox foi um dos primeiros fabricantes a comercializar equipamentos, com débitos superiores a 6.5Gbits/s [Mellanox 03], e a Quadrics [Petrini 02], que acrescenta um sistema operativo de rede capaz de garantir tolerância a faltas e oferece débitos superiores a 2.5Gbits/s.

2.1.2 Bibliotecas de comunicação de baixo-nível

A instalação de uma tecnologia de comunicação de elevado desempenho não garante, por si só, desempenhos elevados ao nível das aplicações. De facto, os procedimentos impostos pelos sistemas operativos, para a interacção com periféricos, não são adequados para a utilização deste tipo de tecnologias. Se for usada a pilha TCP/IP, por exemplo, será necessário, ao nível das aplicações, recorrer a técnicas que permitam mascarar os atrasos da comunicação [Strumpen 96] (sobrepondo comunicação e computação), técnicas essas que não têm uma aplicação directa na maior parte dos casos.

Com o intuito de explorar eficientemente as tecnologias SAN e trazer para as aplicações o potencial do hardware de interligação, foram definidas arquitecturas de comunicação, materializadas em bibliotecas de comunicação de nível utilizador, que permitem o acesso directo ao controlador de rede e evitam a cópia de dados [Warschko 99b]. Estas bibliotecas, de baixo-nível pelo facto de oferecerem funcionalidade mínima, fazem com que o sistema operativo não tenha qualquer intervenção na comunicação e permitem que os controladores de rede manipulem directamente os dados no espaço de endereçamento das aplicações.

A inclusão de funcionalidades específicas no hardware de comunicação, para reduzir os tempos de reposta na troca de informação, tinha já sido considerada no modelo de mensagens activas [von Eicken 92] e no interface Tempest [Reinhardt 94], para o qual é especificada a plataforma de hardware Thyphoon. No contexto das tecnologias de comunicação actuais, os processadores existentes nos controladores de rede são programados para aceder a zonas específicas da memória das aplicações, sendo usadas diversas técnicas para garantir a protecção dos dados de cada aplicação [Dubnicki 97, Welsh 97]. Esses processadores são também usados para executar parte do protocolo de comunicação, existindo a preocupação, em determinadas bibliotecas, de rentabilizar os múltiplos processadores de alguns controladores [Shivam 02] ou dividir pelos vários processadores de uma máquina SMP as

tarefas que o processador do controlador não assume [Wong 99].

Para tornar possível a uniformização da exploração de equipamentos de diferentes fabricantes e permitir uma maior portabilidade das aplicações, surgiu a norma VIA (Virtual Interface Architecture) [Compaq 97], que especifica uma arquitectura para a interacção entre sistemas de computação e hardware de comunicação de elevado desempenho. As vantagens desta arquitectura foram amplamente enfatizadas, quer do ponto de vista comercial [Clariion 98], quer do ponto de vista do desempenho, visto que muitas tecnologias até então exploradas via TCP/IP puderam ser exploradas de forma mais eficiente através de implementações VIA [Farrell 00b, Speight 99].

De entre as implementações VIA conhecidas, pelo facto de serem de uso livre, destacam-se o M-VIA (Modular VIA) [Bozeman 99], que inclui suporte para alguns controladores Fast Ethernet, Gigabit Ethernet e outros, e o MyVIA [Chen 02], que permite explorar controladores Myrinet. Com o intuito de validar uma dada implementação VIA, desenvolvida para suportar uma nova tecnologia ou como uma alternativa de maior desempenho, foram também criadas ferramentas para verificar a conformidade de uma implementação para com a norma [Intel 98]. Também foram desenhados sistemas para avaliação de desempenho capazes de avaliar o impacto das diferentes estratégias usadas na implementação da funcionalidade descrita na especificação VIA, entre os quais se destaca o VIBe (Virtual Interface Benchmark) [Kutluğ 01].

Algumas bibliotecas de comunicação, como é o caso do M-VIA, do GAMMA [Ciaccio 99], que permite explorar a tecnologia Fast Ethernet, e do GigaE-PM [Sumimoto 99], que permite explorar a tecnologia Gigabit, não se baseiam em funcionalidades específicas do hardware programável dos controladores de rede, com o objectivo de suportar equipamentos variados e de gama baixa. No entanto, tem-se assistido ao desenvolvimento de bibliotecas que apenas suportam equipamentos específicos, normalmente de gama alta, tirando o máximo partido dos processadores existentes nos controladores de rede, como acontece no sistema EMP [Shivam 01] desenvolvido para o controlador Gigabit Alteon e em variadas bibliotecas [dos Santos 99] desenvolvidas para a tecnologia Myrinet.

Dada a elevada flexibilidade do hardware Myrinet, têm sido investigados variados modelos de comunicação com base em bibliotecas desenvolvidas para essa tecnologia. Entre os vários casos de sucesso destacam-se o PM [Tezuka 97], o LFC [Bhoedjang 98b], o BIP [Geoffray 99], possuindo este a particularidade de ter sido desenhado para a operação em nós SMP, o protocolo usado no ParaStation2 [Warschko 99a] e o GM [Myricom 00], desenvolvido e periodicamente actualizado pela Myricom (o fabricante do equipamento Myrinet). Por motivos óbvios, o GM é actualmente o sistema mais utilizado, possuindo algumas semelhanças com a arquitectura VIA, que podem ser constatadas pela análise da implementação VI-GM [Myricom 02] (uma implementação VIA sobre GM).

Do ponto de vista da recepção, as bibliotecas de comunicação de baixo-nível têm que suportar eficientemente a chegada assíncrona de mensagens. Neste âmbito, foram propostas estratégias que combinam a sondagem e o tratamento de interrupções, minimizando o tempo de resposta e o impacto na computação [Damianakis 97]. Em alguns casos, estas estratégias são implementadas ao nível do escalonador de fios-de-execução, quer seja pela construção de um escalonador de nível utilizador específico [Langendoen 96], quer seja pela alteração do escalonador oferecido pelo sistema operativo [Hansen 00].

A recepção assíncrona levanta, ainda, dificuldades na programação de aplicações, pelo que algumas abordagens não impõem a utilização explícita de primitivas de recepção nas aplicações. Nestas abordagens, o envio de uma mensagem desencadeia a execução automática de uma rotina no destino [Bhoedjang 96] ou, em alternativa, são oferecidas operações de leitura e escrita remotas – comunicação com um só interveniente – como acontece nas bibliotecas ARMCI [Nieplocha 01] e FM [Giannini 98].

2.1.3 Integração em sistemas de alto-nível

A generalidade das bibliotecas de comunicação de baixo-nível têm sido usadas para reduzir o peso da comunicação em aplicações desenvolvidas com base em plataformas de programação paralela tradicionais. A integração de bibliotecas de comunicação de baixo-nível em plataformas já existentes permite alcançar melhores desempenhos nas aplicações, sem que seja necessário recorrer a técnicas específicas para esconder os tempos de resposta elevados associados à evocação de primitivas de comunicação [Strumpen 95], técnicas essas que não se adequam a todas as aplicações.

Algumas bibliotecas de comunicação exploram de forma eficiente o hardware disponível e, por via de um interface com abstracções mais elaboradas, adequam-se à programação de aplicações, como é o caso da BSPlib [Hill 98] e do Directed Point [Wang 02]. No entanto, as bibliotecas de comunicação de baixo-nível oferecem, normalmente, funcionalidades muito básicas, pouco convenientes à programação de aplicações paralelas. A definição de uma camada individualizada, para interface com o hardware de comunicação, permite que as plataformas de programação paralela sejam facilmente adaptadas a novas tecnologias e que quaisquer melhoramentos ao nível da comunicação sejam perfeitamente localizados.

Neste sentido, surgiram versões do MPICH para tirar partido do BIP [Geoffray 01], do GM [Geoffray 02] ou do PM [O'Carroll 98] e assim explorar a tecnologia de comunicação Myrinet. Também em relação ao VIA foram realizados estudos no sentido de comprovar a sua adequabilidade à computação paralela/distribuída [Begel 02, Brightwell 00] e fizeram-se adaptações do MPICH e do PVM sobre M-VIA [Pant 99, Espenica 02]. Em plataformas menos vulgares, no contexto da programação paralela, também tem havido esforços no sentido de suportar tecnologias de comunicação de elevado desempenho, como se pode

confirmar pelas implementações DECK sobre SCI [de Oliveira 01] e TreadMarks sobre VIA [Banikazemi 01] e por algumas tentativas de adequar o CORBA ao funcionamento das bibliotecas de comunicação de baixo-nível [Kurmann 03, Kuhns 99].

Numa outra vertente, alguns investigadores propuseram a implementação de serviços específicos sobre bibliotecas de baixo nível, serviços esses que podem ser usados pelas aplicações. São exemplos deste tipo de abordagem algumas implementações de estruturas de dados distribuídas [Gribble 00, Bhoedjang 98a] e de barreiras [Buntinas 01].

Dado que a generalidade das aplicações e plataformas de computação paralela foram, no passado, desenvolvidas sobre TCP/IP, tem havido também algumas tentativas de explorar, a partir da própria pilha TCP/IP, tecnologias de comunicação de elevado desempenho, como acontece no SOVIA [Kim 01] e no Sockets-GM [Fischer 02]. Tal abordagem permite aumentar, significativamente, a portabilidade das aplicações, no que diz respeito às tecnologias de comunicação, dado que existe uma uniformização do interface. No entanto, o modelo de comunicação do TCP/IP, apesar de conveniente, não permite explorar de forma eficiente tecnologias de comunicação de elevado desempenho [Barak 99].

Neste contexto, alguns autores propuseram a criação de bibliotecas/sistemas de comunicação de nível intermédio, capazes de integrar vários subsistemas de comunicação – bibliotecas de comunicação de baixo-nível e respectivas tecnologias de comunicação – e oferecer um interface único, para o desenvolvimento de aplicações ou plataformas de alto-nível. Estas bibliotecas, das quais o Madeleine [Bougé 98] e o Panda [Rühl 96] são dois bons exemplos, aumentam significativamente a portabilidade das aplicações.

2.1.4 Suporte multitecnologia

Os nós de um *cluster* são, na generalidade dos casos, interligados por uma única tecnologia de comunicação de elevado desempenho – nós homogéneos, do ponto de vista comunicacional. A instalação de vários *clusters*, cada um constituído por nós homogéneos, numa dada instituição, é também vulgar, fundamentalmente para efeitos de comparação de tecnologias: Myrinet versus SCI [Seifert 00], VIA versus SCI [de Oliveira 00], etc.

Neste contexto, poder-se-á considerar a constituição de um *cluster* de nós heterogéneos, a partir de *subclusters* que integram apenas nós homogéneos. Estes *subclusters* serão interligados por nós que disponham de um controlador de rede por cada tecnologia envolvida – nós multiconectados. Um *cluster* deste tipo deverá ser explorado como se fosse um sistema de computação único e, eventualmente, as aplicações (ou partes das aplicações) em execução nos nós multiconectados deverão poder tirar partido das várias tecnologias disponíveis. No entanto, este tipo de exploração requer que as bibliotecas de comunicação suportem múltiplas tecnologias.

As bibliotecas de nível intermédio podem ser consideradas uma primeira aproximação ao suporte multitecnologia, atendendo a que a definição de uma camada de uniformização, do género da pilha TCP/IP, não é uma solução desejável. No sistema Nexus [Foster 94], o suporte à execução em ambientes heterogéneos, em especial a utilização simultânea de múltiplos protocolos de comunicação, é fixado como objectivo essencial. No entanto, este sistema não é apropriado para a operação num ambiente *cluster* onde, apesar de existirem múltiplos *subclusters*, está em causa unicamente tecnologia de elevado desempenho. Noutros sistemas, caso do Panda, por exemplo, o objectivo é a portabilidade, isto é, a possibilidade de compilar uma aplicação para diferentes arquitecturas alvo, não sendo suportada a execução de uma aplicação num ambiente heterogéneo.

Deste modo, têm vindo a surgir soluções específicas, como é o caso da biblioteca de comunicação de nível intermédio Madeleine II [Aumage 00], a qual integra SCI, Myrinet e Ethernet, efectuando o reencaminhamento de mensagens ao nível dos nós multiconectados, e do sistema DECK [Barreto 00], ao qual foi adicionada funcionalidade para o reencaminhamento entre tecnologias SCI e Myrinet. Recorrendo à implementação MPICH sobre Madeleine II, por exemplo, os programadores poderão utilizar, numa aplicação, processadores dispersos por nós de diferentes *subclusters*, sendo automaticamente seleccionada, por cada par de processadores, a tecnologia de comunicação mais adequada. A biblioteca Proteus [Chiu 02] vai ainda mais longe, permitindo que duas entidades troquem de protocolo de comunicação em tempo de execução.

Para a operação de *clusters* geograficamente distantes, um outro nível de reencaminhamento, mais oneroso, baseado no TCP, pode ainda ser considerado. O PM^2 [Planquelle 99], por exemplo, inclui essa funcionalidade e, no projecto Albatross [Bal 99], o Panda foi adaptado também nesse sentido. Para efeitos de interoperabilidade de diferentes implementações MPI, o IMPI (Interoperable MPI) [George 00] usa uma abordagem similar.

2.2 Gestão de recursos

A variedade de recursos físicos existentes num sistema de computação, nomeadamente num *cluster* composto por vários *subclusters*, levanta dificuldades ao nível da administração e, principalmente, da utilização ao nível das aplicações. Tais dificuldades serão maiores se no *cluster* estiverem a ser executadas várias aplicações sob controlo de múltiplos utilizadores.

2.2.1 Administração e monitorização

A exploração de um *cluster* implica a instalação, em cada nó, de um sistema operativo, normalmente o Linux por ser um sistema aberto, e de uma ou mais plataformas de ex-

ploração. Para tornar o procedimento de instalação mais conveniente, principalmente em *clusters* com um elevado número de nós, têm surgido ferramentas, com é o caso do NPACI Rocks [Papadopou. 01] e do C3 Tools Suite [Flanery 00], que exploram a clonagem de discos e as facilidades de configuração remota para instalar, de forma rápida, o sistema Linux nos vários nós de um *cluster*. Esta abordagem é válida essencialmente em *clusters* Beowulf [Becker 95], nos quais pode ser identificado um sistema principal e múltiplos nós computacionais com características idênticas (do ponto de vista do hardware).

As tarefas relacionadas com a gestão dos vários nós de um *cluster*, como por exemplo a gestão de versões do software instalado, gestão de processos, etc., são também motivo de preocupação dos administradores. Algumas plataformas, como por exemplo o MSC.Linux [MSC 01] e o Scyld Beowulf [Scyld 01], incluem versões modificadas do Linux, tendo-lhes sido acrescentados comandos e utilitários que visam facilitar a administração.

A monitorização do estado de cada nó é também uma funcionalidade geralmente oferecida aos administradores de um *cluster* ou mesmo aos simples utilizadores, para que estes possam tomar decisões relativamente ao arranque de aplicações. Geralmente, a partir de um nó do *cluster*, pretende-se saber quais os nós que se encontram activos, a carga computacional de cada um, a temperatura dos processadores, etc. Ferramentas como o Ganglia [RegentsUC 02] e o M3C [Geist 00] permitem visualizar este tipo de informações através de interfaces *Web* ou de aplicações cliente específicas, consultando, de forma transparente, serviços remotos em execução em cada nó computacional.

A um nível mais baixo, pode ainda efectuar-se a monitorização da tecnologia de interligação dos nós do *cluster*. Este procedimento visa, essencialmente, detectar falhas do equipamento e pontos de congestionamento. No caso da tecnologia Myrinet, estão disponíveis para este efeito algumas funcionalidades, ao nível do hardware e do software [Finucane 02].

Por forma a facilitar tanto a instalação como a monitorização e a normal utilização de um *cluster*, isto é, o desenvolvimento e execução de aplicações, foram criados pacotes de software que incluem: sistema operativo, normalmente o Linux com alterações e acréscimos pontuais, ferramentas para instalação, utilitários para gestão centralizada, serviços de monitorização e plataformas para desenvolvimento. O OSCAR [OCG 01] e o SCE [ParGrpKU 01] são dois exemplos deste tipo de pacotes de software.

2.2.2 Imagem de sistema uno

A natureza distribuída e heterogénea dos recursos físicos de um *cluster* dificulta a tarefa dos programadores. Uma imagem de sistema uno corresponde a uma ilusão, criada por hardware ou software, que permite explorar a totalidade de recursos como um único recurso unificado mais poderoso. Esta imagem pode ser criada a diferentes níveis [Buyya 01], desde

o nível do sistema operativo ao nível da aplicação.

Ao nível do sistema operativo, foram feitos alguns esforços no sentido de adaptar soluções já existentes à operação em ambiente *cluster*, garantindo balanceamento de carga e elevada disponibilidade à custa de mecanismos de migração de processos. O GLUnix [Ghormley 98] e o MOSIX [Barak 98] são dois bons exemplos deste tipo de abordagem, os quais são vulgarmente designados por sistemas operativos de nível utilizador. O SCore-D [Hori 99] é também um sistema operativo de nível utilizador, mas, ao contrário do GLUnix e do MOSIX, distingue entre aplicações paralelas e aplicações normais, tratando de escalonar, no universo de todos os nós, apenas os processos das aplicações paralelas.

O STORM [Frachten. 02], embora não suporte a migração de processos, inclui mecanismos para o escalonamento em ambientes *cluster*, permitindo também que a totalidade dos recursos seja explorada como um todo. O escalonamento das tarefas de uma aplicação em sistemas *multicluster*, isto é, no contexto de múltiplos *clusters* interligados via Internet, também foi explorado em alguns trabalhos. Em [Nieuwpoort 01] é apresentado um escalonador que, com base em anotações colocadas numa aplicação Java, é capaz de rentabilizar os recursos de dois *clusters*, sem que a ligação de baixo desempenho, usada para chegar de um *cluster* a outro, prejudique significativamente o tempo de execução das aplicações.

Ao nível da aplicação, são bons exemplos de implementação de imagens de sistema uno os serviços *Web* desenvolvidos para suportar taxas de pedidos elevadas à custa da utilização de múltiplos servidores. Nestes sistemas, são usados mecanismos automáticos de distribuição dos pedidos, por parte de um servidor de interface, e os clientes não têm a noção da existência de um *cluster* de servidores *Web* (imagem de sistema uno), como é o caso do Linux Virtual Server [Zhang 00]. O Network Dispatcher [Hunt 98], o ONE-IP [Damani 97] e o NLB [Microsoft 00] são abordagens similares, embora implementadas a um nível mais baixo e, portanto, mais genéricas.

2.2.3 Selecção e alocação

A execução de aplicações num *cluster* requer meios para a selecção, reserva e alocação de recursos, principalmente se se tratar de um ambiente multi-aplicação e multi-utilizador. Alguns sistemas de escalonamento de tarefas, como é o caso do PBS [Jones 01], permitem que o utilizador indique, via linha de comando, a lista de recursos (processadores, memória, disco, etc.) que devem ser assegurados antes da execução da tarefa.

Para a especificação genérica dos recursos disponíveis num *cluster*, em [Brune 99] é proposta uma linguagem específica bem como um interface gráfico, o qual facilita a descrição de sistemas de computação complexos (em termos de heterogeneidade e topologia). Esta linguagem de descrição de recursos permite, também, descrever serviços e requisitos asso-

ciados a tarefas computacionais. O sistema de gestão de recursos do CCS [Keller 01] usa tais facilidades para tomar decisões de escalonamento.

Num ambiente altamente dinâmico, onde as aplicações ou processos/fios-de-execução das aplicações aparecem e desaparecem de forma assíncrona e a um ritmo elevado, são necessários mecanismos para tratar eficientemente a alocação e libertação de recursos. Em [Ramamoor. 97] é apresentado um mecanismo baseado em testemunhos e calendários que permite melhorar o desempenho das infra-estruturas de escalonamento.

Apesar de os escalonadores desempenharem um papel importante na selecção e alocação de recursos, muitas vezes é o programador quem acaba por decidir directamente a localização de processos, sem ter em consideração o sistema de gestão de recursos. Deste modo, o sistema PLUS [Brune 97] propõe algumas primitivas adicionais que permitem, num programa PVM ou MPI, interactuar com o sistema de gestão de recursos.

No sentido de contemplar a manipulação de recursos em larga escala, foi proposto, no âmbito do projecto Globus [Foster 97], um serviço de directório distribuído [Fitzgerald 97], que inclui um modelo de dados, para a representação da estrutura de recursos e do estado destes, e um interface de programação baseados em LDAP. Para além da infra-estrutura de representação e consulta, a manipulação de recursos a esta escala requer técnicas de co-alocação [Bucur 03], isto é, técnicas para alocação simultânea, em vários *clusters*, de processadores e outros recursos necessários a uma dada tarefa.

2.3 Paradigmas de programação

O desenvolvimento de aplicações paralelas está intimamente ligado à computação de elevado desempenho, mais especificamente à resolução de problemas de dimensão cada vez maior, já que, no âmbito de um problema de dimensão fixa, a lei de Amdahl impõe limitações quanto aos recursos computacionais utilizados [Worley 02, Gustafson 88].

Dependendo da natureza do problema a resolver, a escolha do modelo de programação é crucial, quer do ponto de vista da programabilidade quer do ponto de vista do desempenho. Os modelos/paradigmas de programação possíveis num *cluster* constituído por nós SMP podem ser classificados de acordo com 3 eixos [Gropp 95]: espaço de endereçamento, processos versus fios-de-execução e heterogeneidade. Neste contexto, são de seguida apresentados os quatro paradigmas mais relevantes.

2.3.1 Memória partilhada

O paradigma da memória partilhada baseia-se na utilização de múltiplos fios-de-execução, partilhando um espaço de endereçamento único. De entre as várias implementações de

sistemas de suporte a múltiplos fios-de-execução, para as quais [Kavi 99] faz um levantamento exaustivo, as mais divulgadas são as que seguem a norma POSIX, a qual estabelece um interface de programação sem especificar pormenores de implementação. De facto, essas implementações têm tido grande aceitação no mundo UNIX, tendo sido extensivamente analisadas e comparadas [Garcia 00].

A grande preocupação ao nível dos sistemas de suporte à programação com múltiplos fios-de-execução prende-se com os tempos de comutação de contexto entre fios-de-execução. Na verdade, tem havido alguma discussão relativamente ao papel do sistema operativo no escalonamento dos fios-de-execução e as duas implementações daí decorrentes – fios-de-execução de nível utilizador ou de nível sistema – têm sido amplamente confrontadas [Cohen 98]. Apesar de as implementações de nível utilizador garantirem, pressupostamente, tempos de comutação mais baixos, não permitem utilizar os múltiplos processadores de uma máquina SMP. Além disso, obrigam, em alguns casos, a alterações ao sistema operativo, como acontece na biblioteca Marcel [Danjean 00], colocando em causa a portabilidade. No caso do GNU Portable Threads [Engelschall 00] é usada uma abordagem que não recorre a tais alterações, mas em contrapartida as limitações desta biblioteca são mais acentuadas.

Com o aparecimento das implementações NGPT [Abt 02] e NPTL [Drepper 03], a problemática entre os dois tipos de abordagens atenuou-se. A primeira utiliza uma abordagem híbrida, isto é, permite que os fios-de-execução de uma aplicação se dividam por vários fios-de-execução de nível sistema, tal como acontece no Solaris, e garante tempos de execução muito baixos, apesar de ser uma abordagem híbrida. A segunda, que substituiu o LinuxThreads no RedHat 9.0, apesar de manter a filosofia original do LinuxThreads – fios-de-execução de nível sistema – consegue superar o desempenho da generalidade das implementações de nível utilizador.

No entanto, para suporte ao paralelismo de grão fino[1], tem havido a preocupação de desenvolver plataformas específicas [Price 03]. De facto, neste âmbito, não só o tempo de comutação de contexto deverá ser reduzido como também a criação e destruição de fios-de-execução deverão ser expeditas. Em particular, têm sido apresentadas várias abordagens em torno do conceito de Nanothreading [Nikolopou. 99a, Nikolopou. 99b].

Do ponto de vista da utilização de fios-de-execução, o programador deverá ter em atenção que, na generalidade dos casos, é necessário recorrer a mecanismos de sincronização e que a criação de um elevado número de fios-de-execução levará à inevitável degradação do desempenho. Em [Welsh 00] são apresentadas algumas técnicas para a construção de sistemas concorrentes, com o intuito de manter o número de fios-de-execução relativamente

[1]Note-se que qualquer sistema de fios-de-execução suporta o paralelismo de grão fino quando comparado com um sistema baseado em processos.

baixo, independentemente do número de tarefas a serem executadas.

Dado que a utilização explícita de fios-de-execução, recorrendo por exemplo ao interface POSIX, é considerada pouco conveniente para os programadores desenvolverem aplicações, foi criada uma norma para expressar o paralelismo das aplicações através de directivas de compilação – o OpenMP [Dagum 98]. Estas directivas são usadas pelo compilador para a criação e coordenação dos fios-de-execução necessários, sendo possível, em alguns casos, manter o mesmo desempenho que seria alcançado com a utilização directa de fios-de-execução [Dedu 00]. De igual forma, no sistema Cilk [Blumofe 95] são propostas extensões à linguagem C, juntamente com um escalonador próprio, garantindo-se níveis excepcionais de desempenho à custa de algumas imposições ao nível do modelo de programação.

2.3.2 Passagem de mensagens

O paradigma da passagem de mensagens é, sem dúvida, o mais utilizado na programação de aplicações paralelas, no contexto da computação baseada em *clusters*, por via da grande implantação do MPI (Message Passing Interface) [Snir 95] e do PVM (Parallel Virtual Machine) [Geist 94], duas especificações para bibliotecas de programação. No caso do MPI, é de realçar o MPICH [Gropp 96], a implementação mais divulgada devido a ser de uso livre e suportar uma grande variedade de tecnologias de comunicação.

Na programação por passagem de mensagens são normalmente usados processos para materializar as entidades envolvidas na troca de mensagens e, por definição, não há partilha do espaço de endereçamento. No entanto, ao nível da implementação do sistema de passagem de mensagens, podem ser explorados mecanismos para optimizar a comunicação entre entidades em execução num mesmo nó, os quais, habitualmente, se baseiam em memória partilhada ou abordagens similares. No MPICH-PM/CLUMP [Takahashi 99], por exemplo, foi acrescentada uma primitiva ao sistema operativo para possibilitar a leitura de dados entre processos num mesmo nó.

Apesar das diferenças de raíz do PVM e MPI [Gropp 98], pode ser estabelecido um paralelo entre ambos [Geist 96], daí resultando um conjunto de funcionalidades de referência no panorama da programação por passagem de mensagens. No entanto, os programadores, de acordo com a especificidade das aplicações, acabam por usar apenas um subconjunto das potencialidades disponíveis nestes dois sistemas.

Neste contexto, surgiram alguns sistemas de passagem de mensagens que não implementam determinadas funcionalidades relacionadas com contextos de comunicação, operações colectivas, serviço de identificação, diferentes modos de envio e recepção, tratamento de mensagens assíncronas, etc. Assim, estes sistemas são mais simples, facilitando a experimentação de novas técnicas, por parte dos investigadores, e reduzindo o tempo de apren-

dizagem daqueles que começam a programar aplicações paralelas. O MP_Lite [Turner 01] é um destes sistemas, implementando um subconjunto das primitivas do MPI. De igual forma, no Para++ [Coulaud 95] foram seleccionadas algumas funcionalidades comuns ao PVM e ao MPI, por forma a produzir uma biblioteca com um interface de programação único capaz de explorar ambos os sistemas.

A programação por passagem de mensagens recorre frequentemente a operações colectivas, que, na sua vertente mais simples, correspondem à difusão selectiva de mensagens, ou seja ao envio atómico de uma mensagem a um grupo de entidades. No contexto de um *cluster*, a optimização destas operações tem merecido especial atenção [Huse 99, Thakur 03]. Na verdade, as tecnologias de comunicação típicas de um *cluster* impõem a utilização de funcionalidades especiais implementadas ao nível dos subsistemas de comunicação [Verstoep 96, Sun 02] ou a utilização de árvores de dispersão, cuja construção, ao nível da plataforma de passagem de mensagens, pode recorrer a algoritmos distribuídos complexos [Singh 98].

Ainda no âmbito das operações colectivas, mais propriamente da difusão selectiva, os mecanismos de gestão de membros de um grupo poderão ir desde as abordagens estáticas, usadas no MPI, por exemplo, às abordagens dinâmicas centralizadas, caso do DECK [Cassali 00], do ALMI [Pendarakis 01] e do PVM, ou mesmo distribuídas, usando conceitos P2P (Peer-to-Peer), como acontece no Scribe [Castro 02].

Para fazer face à crescente heterogeneidade de determinados sistemas de computação, têm surgido adaptações de soluções tradicionais, caso do MPICH-G [Foster 98], uma implementação MPI para ambientes *Grid*, e inclusivamente novas soluções, como é o caso do DataExchange [Eisenhauer 98], o qual aposta na vertente cooperativa das aplicações, suportando diversidade aplicacional.

Algumas plataformas de programação paralela orientada por objectos (por exemplo, o DPC+ [Silveira 00]) e aquelas que oferecem a evocação remota como mecanismo de interacção entre entidades (caso do RMI [Philippsen 00] e do GMI [Maassen 02]) podem também ser consideradas casos particulares da passagem de mensagens.

2.3.3 Memória global

O paradigma da memória global, que é vulgarmente denominado de memória partilhada virtual, memória partilhada distribuída ou simplesmente memória partilhada, confundindo-se neste caso com a exploração de múltiplos fios-de-execução, baseia-se na criação de um espaço de endereçamento global, agregando a memória disponível nos vários nós de um *cluster*. Os dados neste espaço de endereçamento são lidos ou escritos, concorrentemente, pelos vários processos de uma aplicação dispersos pelos nós do *cluster*.

O grau de complexidade das várias implementações possíveis para suporte a este paradigma

dependem do modelo de coerência adoptado e do consequente nível de abstracção oferecido aos programadores [Adve 96]. De facto, num sistema com vários nós computacionais, garantir uma semântica correcta para as várias operações de leitura e escrita de memória, por natureza concorrentes, requer a constante troca de mensagens, sendo necessário relaxar os modelos de coerência, de forma a minimizar o número dessas mensagens [Keleher 92].

Em [Amza 96] são apresentados alguns pormenores relativos à plataforma TreadMarks, uma biblioteca de primitivas, para programação em memória partilhada distribuída, inteiramente construída com base nas tradicionais funcionalidades do sistema Unix. A elevada portabilidade desta abordagem permitiu que muitas aplicações sequenciais fossem paralelizadas mediante este paradigma. Por outro lado, o suporte à execução em larga escala – ambiente *Web*, por exemplo – levou à inclusão de mecanismos adicionais para lidar com a heterogeneidade e com questões de controlo de acesso e segurança [Baratloo 96]. No entanto, por questões de eficiência, algumas plataformas exploram características particulares dos sistemas de computação, como acontece no Brazos [Speight 97], onde são exploradas especificidades do Windows NT e, em particular, múltiplos fios-de-execução e técnicas de difusão selectiva para diminuir o número de mensagens trocadas.

Em algumas aplicações, quando é usado o paradigma da memória global, têm sido constatados níveis baixos de desempenho, por comparação com implementações, dessas mesmas aplicações, que recorrem à passagem de mensagens [Lu 95, Klaiber 94]. A principal desvantagem da maioria dos sistemas que criam a ilusão de um espaço de endereçamento global é o facto de esconderem a hierarquia de memória NUMA do hardware subjacente. De facto, ao estabelecerem uma vista horizontal (com um único nível) para a memória, estes sistemas dificultam o papel do programador no que diz respeito à exploração da localidade e à identificação do efeito dos padrões de acesso aos dados no desempenho da aplicação. Esta é, na realidade, a principal razão para o baixo desempenho de algumas aplicações desenvolvidas de acordo com o paradigma da memória global.

Neste sentido, surgiram abordagens que permitem a gestão explícita, por parte do programador, da hierarquia de memória de um *cluster*. Um exemplo paradigmático desta abordagem é a plataforma Global Arrays [Nieplocha 96b], na qual o programador movimenta explicitamente dados do espaço de endereçamento global (um vector ou uma matriz distribuídos) para estruturas de dados locais, tirando partido de operações de leitura e escrita remotas. Estas operações são oferecidas por uma biblioteca de comunicação específica denominada ARMCI [Nieplocha 99]. Uma abordagem similar foi adoptada na plataforma GAMESS [Olson 03].

Do ponto de vista do modelo de coerência, os sistemas de memória partilhada distribuída mais básicos, dos quais o sistema Midway [Bershad 93] é pioneiro, integram primitivas para, explicitamente, se adquirir acesso a uma zona de memória e, após a sua leitura ou

escrita, se libertar essa zona. O Global Arrays e o GAMESS apresentam abordagens semelhantes. No entanto, o facto de a hierarquia de memória ser exposta ao programador permite distinguir, no próprio programa, acessos locais de acessos remotos. No Global Arrays foi, inclusivamente, introduzido um terceiro nível de acesso, através do Disk Resident Arrays [Nieplocha 96a], que alarga o modelo original à movimentação de dados entre memória principal e memória secundária.

2.3.4 Modelos híbridos

A combinação de vários modelos de programação, no desenvolvimento de uma única aplicação, traduz a assunção da heterogeneidade do sistema de computação; em alguns casos, o sistema de computação propicia a escrita de programas heterogéneos, do ponto de vista do modelo de programação usado. Num *cluster* de nós multiprocessador, por exemplo, é natural reconhecer que, ao nível de cada nó, o paradigma da memória partilhada é o mais adequado, enquanto que, entre dois nós, a troca de informação deverá ser efectuada por via da passagem de mensagens. Tomando como referência os três modelos de programação apresentados – memória partilhada, passagem de mensagens e memória global – podem ser estabelecidos quatro modelos híbridos.

O primeiro, o modelo híbrido mais comum, combina passagem de mensagens e memória partilhada, numa tentativa de tirar partido dos nós multiprocessador de um *cluster*, como referido anteriormente, e/ou conseguir a sobreposição da computação com a comunicação [Felten 92]. Vulgarmente são usados o MPI e o OpenMP, em conjunto [He 02, Cappello 00], contornando-se, ao nível da aplicação, as deficiências que o MPI apresenta no suporte à utilização de fios-de-execução. Têm ainda surgido abordagens no sentido de acrescentar ao MPI suporte genérico para múltiplos fios-de-execução [Protopopov 01, Chowdappa 94], destacando-se, neste contexto, a nova especificação MPI-2 [Huss-Led. 97]. Em relação ao PVM, o TPVM [Ferrari 95] e o LPVM [Zhou 98] constituem abordagens similares.

Algumas plataformas foram desenhadas de raíz para suportar um modelo híbrido de passagem de mensagens e memória partilhada, como é o caso do Chant [Haines 94], onde fios-de-execução dispersos pelos nós de um *cluster* podem trocar mensagens entre si, e do PM^2 [Namyst 95], em que são suportadas evocações remotas em ambiente multi- fio- -de-execução. Outras, como sejam o SIMPLE [Bader 99], o Sputnik [Peisert 01] e o Kelp [Baden 00], para além de combinarem de raíz a passagem de mensagens e a memória partilhada, fazem uma abordagem mais metodológica à programação de *clusters* SMP, introduzindo abstracções próprias para lidar com o paralelismo multinível e com a hierarquia do hardware do sistema de computação.

O segundo modelo híbrido, um pouco mais invulgar, baseia-se na combinação do para-

digma da memória partilhada com o da memória global. No sistema Brazos [Speight 98b], por exemplo, é incluído suporte para múltiplos fios-de-execução, ao nível da aplicação, podendo-se considerar que é oferecido um modelo de programação híbrido. No entanto, apenas a abordagem DSA (Distributed Shared Array) [Xu 99], a qual expõe a organização hierárquica do *cluster* ao programador, permite o controlo explícito da distribuição dos dados e a consequente exploração da localidade por via do suporte multi-fio-de-execução em Java.

O terceiro modelo híbrido diz respeito à utilização conjunta da passagem de mensagens e da memória global. Vários autores têm argumentado que, dada a natureza do hardware de comunicação de um *cluster*, um modelo de programação baseado unicamente no paradigma da memória global não permite explorar eficientemente esse sistema de computação [Kranz 93, Frank 93]. O próprio Global Arrays inclui primitivas para o envio e recepção de mensagens [Nieplocha 02], tendo também sido testadas tais funcionalidades numa versão experimental do sistema Brazos [Speight 98a]. No sistema Charlotte [Karl 98] é proposto um esquema de anotações às rotinas escritas em Java, através do qual o programador indica em que circunstâncias o sistema deverá usar passagem de mensagens, em vez de memória partilhada distribuída, para o acesso a objectos distribuídos.

O quarto modelo híbrido, o modelo de excelência, será aquele que combina os três paradigmas, optando pelo mecanismo mais adequado, automaticamente ou com base em indicações directas ou indirectas do programador, consoante o hardware disponível e a aplicação em execução. Seguem esta abordagem os sistemas Athapascan [Roch 03] e MPC++ [Ishikawa 99]; no primeiro caso, o programador dispõe de primitivas para criação de fios-de-execução, troca de mensagens e partilha de dados, enquanto no segundo, com um nível de integração superior, são oferecidos mecanismos de evocação remota, apontadores para memória global e facilidades para a sincronização de objectos C++. Também a especificação CoR [Moreira 01] e o respectivo protótipo pCoR [Pina 02] têm por objectivo oferecer um modelo de programação deste tipo.

2.4 Epílogo

A integração de várias tecnologias de interligação, para a construção de *clusters* heterogéneos, tem motivado o desenvolvimento de bibliotecas de comunicação de nível intermédio que não introduzem abstracções relevantes, quando comparadas com as abstracções de mais baixo nível. Basicamente, é oferecida comunicação entre nós do *cluster* ou, em alguns casos, comunicação entre processos/processadores ou portas criadas para o efeito, cabendo aos programadores a responsabilidade pela associação entre entidades aplicacionais e pontos de comunicação. Esta situação cria condições para a investigação

de novos paradigmas adaptados às necessidades de comunicação entre entidades lógicas usadas para materializar abstracções de alto-nível.

O desenvolvimento de aplicações paralelas requer abstracções de alto-nível, que podem ser desenvolvidas sobre bibliotecas de comunicação de nível intermédio. Geralmente, o objectivo principal de tais abstracções é a exploração eficiente dos recursos disponíveis, ficando para segundo plano a simplificação do processo de modelação de uma aplicação, nomeadamente quando o programador pretende modelar a aplicação como uma colecção de fios-de-execução que partilham recursos. Em alguns casos, o suporte multi-fio-de-execução é específico (proprietário), o que dificulta a utilização de módulos que utilizam fios-de-execução POSIX, especialmente em sistemas Linux. Interessa pois criar abstracções que facilitem a modelação de aplicações, mas que não comprometam a utilização de soluções genéricas.

Para explorar os múltiplos níveis de localidade que é possível identificar num *cluster*, alguns autores propuseram modelos de programação que combinam vários paradigmas. No entanto, na generalidade dos casos, o nível do *subcluster* não é considerado, apesar de ser previsível o aparecimento de partições tecnológicas num *cluster* inicialmente homogéneo, do ponto de vista da tecnologia de comunicação. O estabelecimento da correspondência entre entidades lógicas e recursos físicos deverá contemplar a hierarquia de hardware de um *cluster* heterogéneo e permitir inclusivamente a manipulação de *clusters* geograficamente distantes.

O recurso a abstracções de alto-nível não pode comprometer o desempenho oferecido pelo hardware de comunicação. O suporte à combinação de múltiplos paradigmas de programação é importante para a obtenção de soluções eficientes, sem comprometer a conveniência. A criação de abstracções que possam servir os vários paradigmas e, simultaneamente, contemplar a selecção e alocação de recursos é essencial para garantir o grau de aceitação de novas contribuições no domínio da programação paralela.

A criação de mecanismos para a especificação quer dos elementos físicos constituintes de um *cluster* quer da sua própria organização constitui um contributo importante para a problemática da selecção e alocação de recursos. Com o crescente interesse em torno da *Grid* e da metacomputação, este aspecto assumiu extrema importância. No entanto, apesar do elevado grau de desenvolvimento das soluções disponíveis neste domínio, não foram ainda criadas abstracções e mecanismos que, de forma integrada, permitam manipular recursos físicos e entidades aplicacionais.

Capítulo 3

Computação orientada ao recurso

Num passado recente, os *clusters* assumiram um papel importante na execução de programas paralelos para resolução de problemas de índole científica. Essencialmente, os programadores procuravam ambientes de desenvolvimento e execução que permitissem a exploração do *cluster* como se se tratasse de uma colecção de processadores.

Actualmente, e à medida que novos modelos de programação e sistemas de exploração são desenvolvidos, os *clusters* começam a tornar-se uma opção interessante para o suporte de sistemas de informação complexos. Neste âmbito, há todo o interesse em integrar variados componentes de software, constituindo verdadeiros sistemas de aplicações cooperantes.

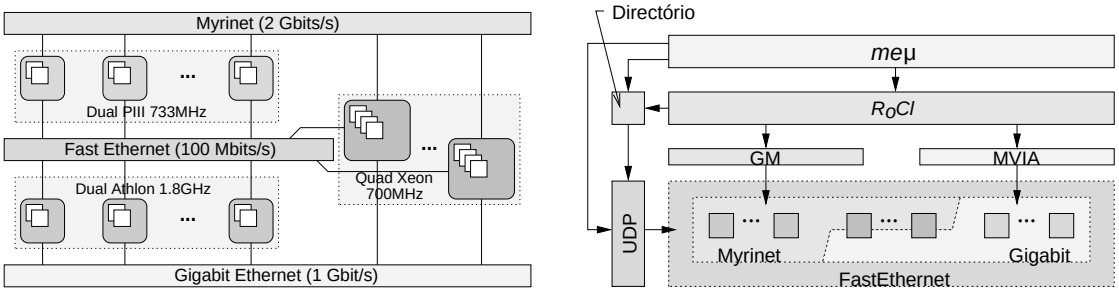
A natureza hierárquica dos *clusters* SMP tem motivado a investigação de modelos de programação apropriados à exploração das localidades resultantes dos dois níveis de paralelismo: interprocessador (intranó) e internó. No entanto, a utilização eficiente e conveniente dos *clusters* SMP multi-SAN e o suporte à execução de múltiplas aplicações cooperantes motivam, ainda, a investigação de novas abordagens.

Neste capítulo é introduzido um novo modelo para a exploração de *clusters* SMP multi-SAN, o qual se baseia no paradigma da orientação ao recurso.

3.1 Arquitectura do sistema de exploração

A utilização de múltiplas tecnologias SAN num *cluster* traduz-se na existência de nós multi-interface e partições tecnológicas (*subclusters*). Os nós multi-interface podem ser utilizados para interligar múltiplos *subclusters*, sendo o garante da conectividade total dos nós de um *cluster*. A figura 3.1(a) apresenta um exemplo de um *cluster* SMP multi-SAN, combinando as tecnologias Myrinet e Gigabit, que constitui a plataforma de experimentação subjacente a esta dissertação. Neste *cluster*, podem ser identificados dois

grupos de máquinas biprocessador e um de máquinas quadriprocessador. Os dois primeiros grupos constituem duas partições tecnológicas, uma Myrinet e outra Gigabit, enquanto que o último, por integrar máquinas conectadas aos comutadores das duas tecnologias de comunicação, serve de elo de ligação. Todas as máquinas do *cluster* estão ligadas por via de um comutador Fast Ethernet.



(a) (b)

Figura 3.1: Exploração de um *cluster* SMP multi-SAN.

3.1.1 Imagem de sistema uno de nível comunicacional

Com o intuito de explorar eficientemente um *cluster* do género do representado na figura 3.1(a), foi desenhada uma biblioteca de comunicação orientada ao recurso – o R_OCl (*Resource oriented Communication library*) – que combina as bibliotecas de comunicação de baixo-nível GM e M-VIA, de forma a oferecer serviços de passagem de mensagens e de leitura/escrita remotas entre entidades aplicacionais dispersas pelos vários nós computacionais. O conceito de comunicação orientada ao recurso deriva do facto de se considerar que as entidades dotadas de capacidade comunicacional constituem um conjunto de recursos aplicacionais que, uma vez registados (publicados) aquando da sua criação, podem ser localizados a partir de qualquer componente software a executar no *cluster*, por forma a que se estabeleçam parcerias comunicacionais.

O conceito de comunicação orientada ao recurso impõe a utilização de um serviço de directório adequado à operação em ambiente *cluster*. Este serviço tira partido das topologias de interligação de um *cluster* multi-SAN e, principalmente, explora o facto de todos os nós de um *cluster* se encontrarem, normalmente, interligados por uma tecnologia de comunicação de base, como Fast Ethernet, vulgarmente usada para operações de instalação, manutenção e gestão.

No sentido de assegurar a execução de sistemas de aplicações complexos, esta biblioteca foi desenhada por forma a proporcionar uma plataforma de comunicação com suporte

nativo para múltiplos fios-de-execução. Desta forma, os recursos aplicacionais podem ser animados por fios-de-execução, cabendo ao R_OCl oferecer uma imagem de sistema uno, suportada pelo directório e pela integração de diferentes tecnologias de comunicação. A figura 3.1(b) ilustra a camada R_OCl sobre as bibliotecas de comunicação de baixo-nível GM e M-VIA e sobre o serviço de directório, escondendo a heterogeneidade comunicacional do *cluster* multi-SAN.

É importante referir que, apesar de esta imagem de sistema uno garantir a total conectividade de entidades criadas sem qualquer preocupação de organização, o desempenho de uma aplicação dependerá fortemente do local (nó do *cluster*) escolhido para cada uma das entidades aplicacionais.

3.1.2 Modelação e exploração unificadas

A organização das entidades que constituem uma aplicação deverá, na medida do possível, ir de encontro à própria organização dos recursos físicos do *cluster*. Neste sentido, foi desenhado um sistema de suporte à execução – o $m_\varepsilon\mu$ (*Modelling applications and Exploiting clusters through a Unified abstraction layer*) – que fornece mecanismos para a organização de entidades aplicacionais.

As abstracções oferecidas pelo $m_\varepsilon\mu$ destinam-se a auxiliar o programador na construção de sistemas de aplicações cooperantes, permitindo um controlo explícito dos recursos físicos disponíveis. Com base num conjunto reduzido de conceitos, o programador é capaz de modelar aplicações que facilmente se adequam à estrutura de recursos físicos do *cluster*. Basicamente, esta abordagem parte do princípio que o programador é capaz de interpretar a complexidade do hardware que compõe o *cluster* e que pode, portanto, ajudar o sistema de suporte à execução na criação de componentes lógicos.

Visto que a imagem de sistema uno oferecida pelo R_OCl não inclui quaisquer mecanismos para a criação de entidades lógicas, cabe integralmente ao $m_\varepsilon\mu$ o papel de suportar a selecção de recursos físicos com vista à materialização de um dado componente software. Para o efeito, o $m_\varepsilon\mu$ utiliza o serviço de directório para armazenar cada um dos recursos físicos do *cluster* e a informação relativa à forma como esses recursos estão organizados. Esta representação dos recursos físicos permite que o programador e o sistema de suporte à execução, em conjunto, tomem decisões quanto ao posicionamento de entidades lógicas, tendo em vista a exploração dos vários níveis de localidade de um *cluster* SMP multi-SAN.

Tendo em conta o percurso seguido na construção de uma aplicação, pode-se dizer que o $m_\varepsilon\mu$ constitui uma metodologia de programação que inclui as seguintes fases:

- especificação da hierarquia de recursos físicos que constituem o *cluster*;

- modelação da aplicação ou do conjunto de aplicações cooperantes do sistema;
- estabelecimento da correspondência entre componentes aplicacionais (entidades lógicas) e recursos físicos.

3.2 Recursos físicos e lógicos

As duas primeiras fases da metodologia de programação $m_{\varepsilon}\mu$ exigem abstracções que facilitem o desenho de aplicações e a manipulação eficiente dos recursos físicos de um *cluster*. A manipulação de recursos físicos, nomeadamente a selecção e a alocação de recursos, exige formalismos e mecanismos adequados para a sua representação e organização. O desenvolvimento de aplicações paralelas/distribuídas que tirem partido de um *cluster* SMP multi-SAN requer abstracções apropriadas à modelação de componentes software em consonância com a organização dos recursos hardware.

3.2.1 Antecedentes

O paradigma da orientação ao recurso é o resultado do desenvolvimento dos conceitos principais de célula, operão e gene propostos no Modelo de Computação Celular [Pina 97], MC^2, para explorar a computação paralela e distribuída de grão fino.

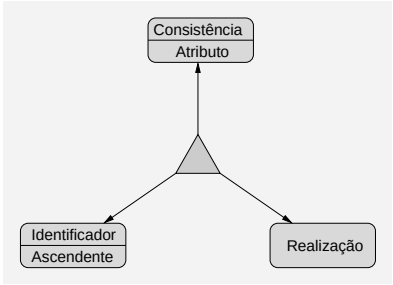


Figura 3.2: Constituição da abstracção recurso.

Na sua definição primordial, o recurso é uma entidade básica de computação e de programação cujo comportamento assenta nos elementos constitutivos visíveis na figura 3.2:

- ascendente – uma referência usada para ligar um recurso, no momento da sua criação, a um outro recurso seu ascendente;
- identificador – um índice e um nome (opcional) que identificam univocamente o recurso no contexto do recurso ascendente;

- atributo – uma referência a um objecto que regista o estado inicial do recurso, no momento da sua criação, e pode ser usado para inquirir ou alterar o seu comportamento e propriedades actuais;
- realização - uma referência ao objecto de computação que é a realização concreta do recurso no sistema hospedeiro;
- consistência – uma referência ao objecto usado para controlar os modos de acesso a um recurso e manter a coerência de estado entre todas as suas representações.

O desejo de animar o modelo de computação celular deu posteriormente lugar à realização do ambiente de programação DoTS [Oliveira 97], uma biblioteca implementada sobre PVM que combinava memória partilhada distribuída com passagem de mensagens e fios-de-execução.

Posteriormente, a necessidade de formalizar o paradigma da orientação ao recurso esteve na base da dissertação de mestrado com o título "CoRes: Computação orientada ao Recurso - uma especificação" [Moreira 01]. Este trabalho desenvolveu, enriqueceu e generalizou o conceito de recurso, dando lugar a um refinamento da sua constituição que doravante passa a contemplar os elementos constitutivos contentor, organizador, executor, valor, caixa postal, porto e sincronizador, que traduzem a funcionalidade das classes de recursos então usados para a construção de aplicações, ou seja, domínios, grupos, tarefas, dados, portos, guardas e barreiras.

Este trabalho prossegue o desenvolvimento do paradigma da orientação ao recurso através do refinamento da constituição da entidade básica, da criação de novos esquemas para a organização de recursos físicos/lógicos e da redefinição das classes de recursos pré-definidas.

Outras contribuições incluem a redefinição e a extensão da metodologia de programação implícita no paradigma da orientação ao recurso que passa a incluir o estabelecimento explícito da correspondência entre a hierarquia de recursos lógicos de uma aplicação e a hierarquia que representa a estrutura física da arquitectura alvo. Os melhoramentos introduzidos põem a ênfase na comunicação de elevado desempenho orientada ao recurso, para permitir a utilização mais eficiente do hardware e, desta forma, alcançar os melhores níveis possíveis de desempenho, a partir do conhecimento dos diferentes níveis de localidade existentes num *cluster* SMP multi-SAN.

3.2.2 Entidades para a modelação de recursos

No $m_{\varepsilon}\mu$, a modelação dos recursos físicos de um *cluster* e dos componentes lógicos das aplicações que nele executam – recursos lógicos – faz-se através de sete entidades, representadas graficamente na figura 3.3, cujo propósito/significado é o seguinte:

- domínio – usado para agrupar/delimitar entidades relacionadas;
- operão – usado para suportar o contexto de execução onde tarefas e blocos de memória são criados;
- tarefa – um fio-de-execução que suporta passagem de mensagens de grão-fino;
- caixa postal – um repositório para/de onde as tarefas enviam/retiram mensagens;
- bloco de memória – uma zona de memória contígua que suporta acessos de leitura/escrita remotos;
- agregador de memória – usado para encadear múltiplos blocos de memória;
- subtractor – usado para suportar a reserva de recursos.

Figura 3.3: Representação gráfica dos recursos.

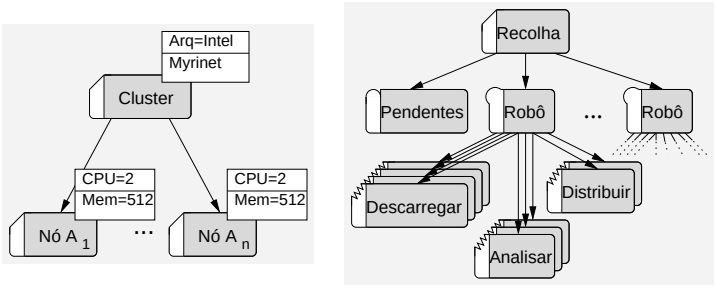
A cada entidade, para além de uma identificação, pode ser associada uma lista de propriedades, por forma a evidenciar a presença de características ou estipular valores que quantificam ou clarificam aspectos relevantes. A uma dada entidade tarefa, por exemplo, poderão ser associadas as propriedades `Resultado=histograma`, `Tempo=10` e `ComputIntensiva`; a primeira clarificará o objectivo da tarefa, a segunda quantificará a sua duração e a última indicará a presença de operações que exigem, essencialmente, tempo de processador.

Na modelação de aplicações, os domínios permitem organizar recursos lógicos representados pelas demais cinco entidades (operões, tarefas, caixas postal, blocos e agregadores), sendo portanto denominados domínios lógicos. Na modelação de recursos físicos são usados unicamente os domínios, ficando reservado às propriedades o papel de inventariação dos recursos físicos propriamente ditos, isto é, um domínio físico engloba um determinado conjunto de recursos hardware caracterizados por uma lista de propriedades.

3.2.3 Organização básica

Dada a natureza intrinsecamente hierárquica dos *clusters* SMP multi-SAN, uma árvore é a estrutura mais indicada para dispor os distintos recursos físicos. De igual forma, a modelação de uma aplicação compreende a definição de uma hierarquia de entidades, com o intuito de garantir uma fácil correspondência entre componentes aplicacionais e recursos

físicos. A figura 3.4 apresenta duas hierarquias de recursos, uma para a especificação dos recursos físicos de um *cluster* simples (figura 3.4(a)) e outra para a modelação de uma aplicação paralela básica destinada à recolha de páginas *Web* (figura 3.4(b)).



(a) (b)

Figura 3.4: Organização de recursos físicos e lógicos.

Numa hierarquia de domínios físicos, os domínios de nível mais elevado (topo da árvore) introduzem recursos gerais, isto é, recursos comuns a vários nós do *cluster*. Os domínios correspondentes às folhas da árvore englobam os recursos hardware mais específicos, mas ainda susceptíveis de serem manipulados pelo sistema de exploração.

Dependendo da capacidade do sistema de exploração para manipular (alocar, reservar, etc.) recursos a um nível mais fino, algumas propriedades podem ser transformadas em subdomínios, cabendo tal decisão ao administrador, que é o responsável pela criação da hierarquia de domínios físicos. Na figura 3.4(a), as propriedades `CPU=2`, usadas na caracterização de domínios que representam máquinas, podem tornar-se subdomínios – cada nó máquina passaria a ter tantos subdomínios *CPU* quantos os seus processadores – desde que o sistema de exploração seja capaz de manipular directamente o recurso processador.

As hierarquias usadas para representar componentes aplicacionais incluem múltiplas entidades de tipos distintos, as quais são encadeadas de acordo com regras preestabelecidas (ver secção 3.2.6). A modelação hierárquica de aplicações está em conformidade com o modelo *dividir para conquistar*, o qual tem sido utilizado em ambientes de programação que visam explorar múltiplos *clusters* [Nieuwpoort 02].

Cada nó de uma árvore de recursos herda as propriedades do seu ascendente, as quais são reunidas com as que directamente lhe foram associadas. Desta forma, é possível atribuir uma determinada propriedade a todos os nós de uma subárvore através da simples associação da propriedade em causa à raíz dessa subárvore. O domínio *Nó* A_1 da figura 3.4(a) reunirá, portanto, as propriedades `Arq=Intel`, `Myrinet`, `CPU=2` e `Mem=512`.

A localização de recursos (físicos ou lógicos) numa dada hierarquia faz-se mediante a espe-

cificação da lista de propriedades que os caracterizam. O sistema de exploração percorre a árvore de recursos e localiza a entidade cujas propriedades acumuladas satisfaçam os requisitos indicados pelo programador. Na figura 3.4(a), qualquer um dos domínios *Nó A* será resposta aos requisitos (`Myrinet`) $\wedge$ (`CPU=2`).

Se as propriedades especificadas pelo programador, para a localização de recursos, se encontram dispersas pelas várias entidades de uma dada subárvore, a estratégia de descoberta devolve a raíz dessa subárvore. Para combinar as propriedades de todos os nós de uma subárvore na sua raíz, o sistema recorre à sintetização. Assim, o domínio *Cluster* da figura 3.4(a) preenche os requisitos (`Myrinet`) $\wedge$ (`CPU=2*n`)[1].

3.2.4 Vistas

A herança e a sintetização não permitem reunir os recursos dispersos por duas ou mais subárvores distintas, a não ser que as raízes dessas subárvores correspondam ao grupo de descendentes de um dado nó. Ainda em relação à figura 3.4(a), se o programador pretender individualizar dois nós do *cluster* em particular, totalizando quatro processadores, não tem qualquer domínio disponível para o efeito, isto é, nenhum domínio preenche cabalmente os requisitos (`CPU=4`).

Na verdade, as relações normais de ascendência/descendência de uma estrutura em árvore não permitem a constituição de vistas alternativas, tendo como objectivo a interpretação dos recursos envolvidos de acordo com os interesses do programador num dado momento. Na prática, seria útil a introdução de domínios virtuais, simbolizando vistas diferentes, mas sem comprometer as vistas inerentes à organização de base. A abordagem aqui apresentada introduz a relação original/pseudónimo e a partilha de propriedades, como resposta à necessidade de criação dinâmica de vistas.

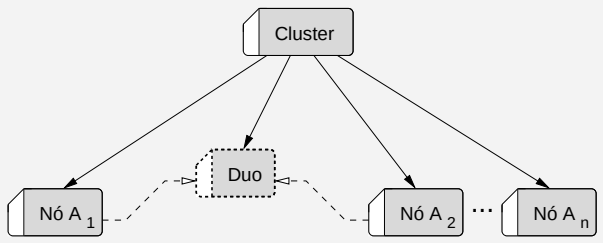


Figura 3.5: Definição de uma vista através de um pseudónimo.

A criação de um pseudónimo implica designar um ascendente e um ou mais nós originais. Na figura 3.5, o domínio *Duo*, representado a tracejado, é um pseudónimo cujos originais,

[1] n diz respeito ao número de nós do *cluster*.

ligados a si por setas a tracejado, são os domínios *Nó* A_1 e *Nó* A_2. Este pseudónimo partilhará as propriedades dos seus originais e herdará as propriedades do seu ascendente (o domínio *Cluster*). Por regra, para além das propriedades obtidas por via da herança (a partir do seu ascendente) e da sintetização (a partir do seus descendentes), um pseudónimo reunirá as propriedades directamente associadas aos seus originais bem como as previamente herdadas ou sintetizadas por estes.

Combinando relações de original/pseudónimo e ascendente/descendente, é possível representar plataformas de hardware complexas e oferecer aos programadores mecanismos para criação dinâmica de vistas, de acordo com os requisitos das aplicações. Esta flexibilidade (para definição de vistas) constitui uma inovação importante, relativamente às abordagens mais comuns para especificação de recursos de um sistema de computação, como por exemplo a plataforma RSD [Brune 99].

3.2.5 Cálculo de propriedades

O processo de obtenção das propriedades de uma entidade terá que levar em conta, para além dos mecanismos de herança, sintetização e partilha de propriedades, a natureza de cada uma das propriedades. Na verdade, as propriedades que incluem um nome e um valor poderão incluir ainda uma função de acumulação destinada ao cálculo de uma nova propriedade, num determinado nó de uma árvore de recursos, a partir de duas ou mais propriedades que possuam o mesmo nome. Os passos associados à obtenção das propriedades de uma entidade são os representados pela operação *P()*, descrita em linguagem algorítmica na figura 3.6.

No algoritmo apresentado, a operação $\bigtriangledown$ representa a acumulação de propriedades (com base nas operações de acumulação associadas), enquanto que a operação *OP()* designa a obtenção, a partir do directório R_oCl, das propriedades directamente associadas a uma entidade e as operações recursivas *IP()*, *SiP()* e *ShP()* implementam, respectivamente, os mecanismos de herança, sintetização e partilha. A figura 3.7 apresenta o funcionamento da operação *P()*, com base num exemplo em que se assume que as propriedades com valores especificados são acumuláveis, tendo associada, como operação de acumulação, a adição de inteiros.

Dado que a existência de pseudónimos pode levar a que uma entidade obtenha mais que uma vez a mesma propriedade, a aplicação das operações de acumulação apenas poderá ter lugar quando todas as propriedades tiverem sido reunidas. Além disso, as propriedades acumuláveis deverão ser etiquetadas com a respectiva origem, por forma a evitar a eliminação de falsas repetições, visto que a normal reunião de propriedades elimina as que possuem nomes e valores idênticos.

$\underline{P(x)}$
$\quad return(\bigtriangledown(OP(x) \cup IP(x) \cup SiP(x) \cup ShP(x)));$

$\underline{IP(x)}$
$\quad y \leftarrow ancestor(x);$
$\quad foreach\ z \in (\{y\} \cup originals(y))$
$\quad\quad p \leftarrow p \cup OP(z) \cup IP(z);$
$\quad return(p);$

$\underline{SiP(x)}$
$\quad foreach\ y \in descendants(x)$
$\quad\quad origs \leftarrow origs \cup originals(y);$
$\quad foreach\ y \in (descendants(x) \cup origs)$
$\quad\quad p \leftarrow p \cup OP(y) \cup SiP(y);$
$\quad return(p);$

$\underline{ShP(x)}$
$\quad foreach\ y \in originals(x)$
$\quad\quad p \leftarrow p \cup P(y);$
$\quad return(p);$

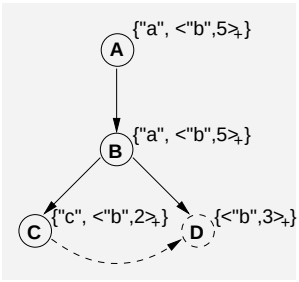
Figura 3.6: Cálculo das propriedades de uma entidade.

3.2.6 Encadeamento de entidades

O domínio é o principal elemento estruturador de aplicações. Na verdade, as tarefas, os blocos de memória e as caixas postal são sempre nós terminais (folhas) da árvore subjacente à hierarquia representativa de uma aplicação, enquanto que os operões e os agregadores de memória, embora não sejam terminais, impõem limitações quanto às subárvores que a partir deles se podem construir.

Basicamente, as restrições ao encadeamento de entidades são as seguintes:

- as tarefas, os blocos de memória e as caixas postal são sempre nós terminais;
- na cadeia de ascendência, isto é, na cadeia de nós até à raíz da árvore, de uma tarefa ou de um bloco de memória terá que existir um operão;
- na cadeia de ascendência de um operão, agregador de memória ou caixa postal terá que existir um domínio que represente um sistema de computação;
- na cadeia de ascendência de um operão não pode existir qualquer outro operão;
- um agregador de memória terá como descendentes apenas blocos de memória.



$$
\begin{aligned}
P(D) &= \bigtriangledown(OP(D) \cup IP(D) \cup ShP(D)) \\
&= \bigtriangledown(\{\langle "b", 3\rangle^{D}_{+}\} \cup (OP(B) \cup OP(A)) \cup P(C)) \\
&= \bigtriangledown(\{\langle "b", 3\rangle^{D}_{+}\} \cup (\{"a", \langle "b", 5\rangle^{B}_{+}\} \cup \{"a", \langle "b", 5\rangle^{A}_{+}\}) \cup \\
&\qquad (OP(C) \cup OP(B) \cup OP(A))) \\
&= \bigtriangledown(\{\langle "b", 3\rangle^{D}_{+}\} \cup \{"a", \langle "b", 5\rangle^{B}_{+}, \langle "b", 5\rangle^{A}_{+}\} \cup \\
&\qquad (\{"c", \langle "b", 2\rangle^{C}_{+}\} \cup \{"a", \langle "b", 5\rangle^{B}_{+}\} \cup \{"a", \langle "b", 5\rangle^{A}_{+}\})) \\
&= \bigtriangledown(\{"a", \langle "b", 3\rangle^{D}_{+}, \langle "b", 5\rangle^{B}_{+}, \langle "b", 5\rangle^{A}_{+}\} \cup \\
&\qquad \{"c", "a", \langle "b", 2\rangle^{C}_{+}, \langle "b", 5\rangle^{B}_{+}, \langle "b", 5\rangle^{A}_{+}\}) \\
&= \bigtriangledown(\{"a", "c", \langle "b", 3\rangle^{D}_{+}, \langle "b", 2\rangle^{C}_{+}, \langle "b", 5\rangle^{B}_{+}, \langle "b", 5\rangle^{A}_{+}\}) \\
&= \{"a", "c", \langle "b", 15\rangle\}
\end{aligned}
$$

Figura 3.7: Reunião de propriedades.

Se se considerarem, também, as relações de pseudonímia presentes numa hierarquia $m_{\varepsilon}\mu$, os mecanismos de verificação das restrições apresentadas terão que refinar o conceito de cadeia de ascendência. É então introduzido o conceito de cadeia de pseudo-ascendência, que traduz as várias cadeias de nós desde um dado ponto até à raíz da árvore, tendo em consideração, para além da relação descendente-ascendente, as múltiplas relações pseudónimo--original de cada entidade.

Na criação de pseudónimos existe ainda uma outra restrição importante: um pseudónimo admite como original uma única entidade e obrigatoriamente do seu tipo. Os pseudónimos domínio constituem excepção, aceitando múltiplos originais e de tipos distintos, isto é, para além de outros domínios, são admitidos, como originais, caixas postal, operões e tarefas. Note-se que a criação de entidades pseudónimo não obriga à verificação das segunda e terceira restrições ao encadeamento anteriormente indicadas.

A criação de entidades é sempre efectuada a partir de tarefas. Assim, é sempre possível obter o identificador da tarefa responsável pela sua criação, o qual é mantido como uma propriedade especial da entidade. O ascendente de cada entidade e os originais dos pseudónimos são mantidos também nos mesmos moldes. No que diz respeito aos descendentes e pseudónimos das entidades, não há qualquer armazenamento explícito. Por conseguinte, a obtenção dos descendentes ou pseudónimos de uma entidade efectua-se de forma inversa.

3.2.7 Orientação ao recurso

A execução de uma aplicação obriga à associação de recursos lógicos a recursos físicos e à interacção entre recursos lógicos. Os domínios físicos que representam o hardware do

cluster são registados no directório com o intuito de colocar à disposição das aplicações a totalidade dos recursos físicos do *cluster*, enquanto que as entidades registadas por uma aplicação podem também ser vistas como um conjunto de recursos lógicos que a aplicação coloca ao dispor de outros componentes aplicacionais.

É nesta perspectiva que o $m_{\varepsilon}\mu$ pode ser entendido como uma abordagem orientada ao recurso, pois tanto o hardware como o software aplicacional de um *cluster* são manipulados através do mesmo conceito – o recurso (físico ou lógico). Note-se que o $R_{O}Cl$ – sistema responsável por garantir os meios necessários ao registo e à localização de recursos físicos e lógicos – é também orientado ao recurso e não faz qualquer distinção entre os dois tipos de recursos.

Formalmente, do ponto de vista sintáctico, um recurso é caracterizado por um nome, um identificador global (atribuído pelo $R_{O}Cl$), uma lista de propriedades, um ascendente, uma lista de descendentes, uma lista de pseudónimos, uma lista de originais, uma lista de membros e uma lista de identificações de membro.

O conceito de membro surge da necessidade de manipular descendentes e originais de forma não diferenciada. A reunião dos descendentes com os originais de uma entidade constitui um conjunto, cujos elementos são numerados pela ordem de criação. Cada um dos membros de uma dada entidade poderá ser univocamente referenciado através do identificador global dessa entidade e de um identificador de membro. Dado que uma determinada entidade é membro não só do seu ascendente mas também de todos os seus pseudónimos, terá associada uma lista de identificadores de membro.

Do ponto de vista semântico, à excepção do subtractor, um recurso é caracterizado pela existência de pelo menos um dos seguintes elementos constitutivos:

- um organizador – um mecanismo para a gestão de membros;
- um contentor – um contexto para a execução de rotinas e armazenamento de dados;
- um porto – um ponto de acesso ao sistema de comunicação, usado para envio/recepção de mensagens e leitura/escrita de memória remota;
- um reencaminhador – um mecanismo para o reenvio de mensagens aos descendentes e originais do recurso;
- um repositório – estruturas de dados que possibilitam o armazenamento de mensagens e a sua recuperação com base em critérios elaborados de selecção;
- um executor – um fio-de-execução que executa o código da função associada a uma tarefa;

- um encaminhador – um mecanismo para direccionar leituras/escritas de memória remota;
- uma sequência de valores – uma zona de memória contígua;
- um sincronizador – um mecanismo para a coordenação de acessos concorrentes.

Tabela 3.1: Caracterização de recursos

Recurso		Elementos constitutivos
domínio	::	NM · [AS · or∗] · ds∗ · PD∗ · oz · pt · re
operão	::	NM · AS · PD∗ · (or · re · ds∗) | (ct · rp · ds+) · oz · pt
tarefa	::	NM · AS · PD∗ · (or · re) | (ex · rp) · pt
caixa postal	::	NM · AS · PD∗ · (or · re) | rp · pt
bloco	::	NM · AS · PD∗ · (or · ec) | sq · pt
agregador	::	NM · AS · PD∗ · (or · ec) | (ds+ · oz · sz) · pt
subtractor	::	NM · AS
NM	::	nm · id · pr∗
AS	::	as · im
PD	::	pd · im

nm=nome, id=identificador global, pr=propriedade, as=ascendente, ds=descendente, or=original, im=identif. de membro, pd=pseudónimo, oz=organizador, ct=contentor, ex=executor, pt=porto, rp=repositório, re=reencaminhador, sq=sequência de valores, sz=sincronizador, ec=encaminhador

A tabela 3.1 apresenta, usando uma sintaxe próxima do EBNF, a caracterização formal dos vários tipos de recursos suportados no $m_{\varepsilon}\mu$, considerando quer os elementos sintácticos quer os semânticos.

3.3 Correspondência entre recursos lógicos e físicos

A última fase da programação de aplicações, de acordo com a metodologia oferecida pelo $m_{\varepsilon}\mu$, consiste na fusão da hierarquia representativa do hardware do *cluster* com a hierarquia representativa da aplicação (ou do sistema de aplicações). O objectivo é produzir uma única hierarquia que estabeleça a correspondência entre recursos lógicos e físicos.

3.3.1 Disposição de recursos lógicos

A figura 3.8 apresenta uma hipótese de integração das entidades das duas hierarquias ilustradas na figura 3.4. A árvore de entidades $m_{\varepsilon}\mu$ resultante combina domínios correspondentes a recursos físicos com entidades de tipos variados (domínios, operões, etc.) correspondentes a componentes lógicos.

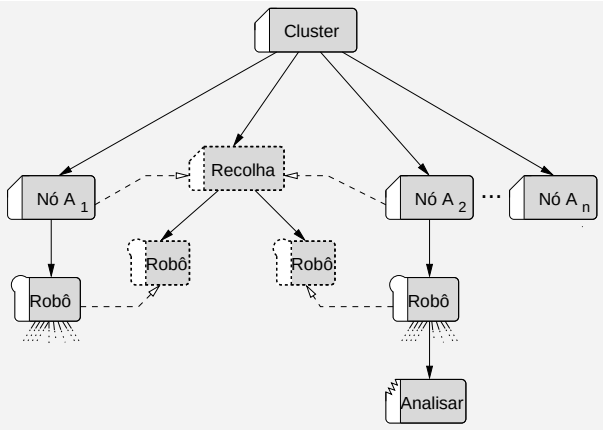


Figura 3.8: Correspondência entre hierarquias lógica e física.

A exploração efectiva dos recursos físicos de um *cluster* passa pela criação de operões, caixas postal e agregadores de memória sob domínios não pseudónimos, da hierarquia de recursos físicos, que representam sistemas de computação. As tarefas e os blocos de memória, dado que são criados no contexto de operões, não têm qualquer papel relevante no que diz respeito à apropriação de hardware.

No sentido de tornar o mecanismo de criação de entidades lógicas mais conveniente, um domínio lógico – um domínio da hierarquia representativa da aplicação – deverá ser usado para delimitar os domínios físicos – domínios da hierarquia representativa do hardware do *cluster* – que um dado conjunto de entidades aplicacionais pode explorar, bastando que esse domínio lógico se apresente como um pseudónimo e constitua uma vista. Na figura 3.8, o domínio lógico *Recolha* estabelece os recursos físicos usados pela aplicação de recolha, uma vez que os operões *Robô* são automaticamente distribuídos pelos nós do *cluster* dispostos sob os originais desse domínio. Na verdade, o domínio *Recolha* é um pseudónimo que constitui uma vista particular dos recursos físicos do *cluster*, delimitando um subconjunto dos recursos disponíveis.

Para preservar a hierarquia concebida pelo programador para a aplicação, o sistema de exploração procede à criação automática de pseudónimos para as entidades lógicas (operões, caixas postal e agregadores de memória) posicionadas sob domínios físicos. Deste modo, estão sempre presentes pelos menos duas perspectivas distintas: a do programador e a do sistema. Por um lado, o programador tem interesse numa visão própria dos componentes de software da sua responsabilidade bem como dos recursos hardware que num dado momento utiliza. Por outro lado, o $m_{\varepsilon}\mu$ interpreta o sistema de computação e as aplicações que nele executam como um conjunto de máquinas e fios-de-execução, sem qualquer estrutura adicional. A visão de sistema é ainda adequada para efeitos de administração, uma

vez que o administrador terá interesse numa perspectiva única do sistema que, no caso da figura 3.8, corresponde à árvore constituída pelas entidades não pseudónimo.

A tarefa *Analisar* da figura 3.8, por exemplo, poderá ser alcançada através de dois percursos distintos: *Cluster* → *Nó* A_2 → *Robô* → *Analisar* – perspectiva do sistema – e *Cluster* → *Recolha* → $Robô^{(Pseudo)}$ → *Analisar* – perspectiva do programador. Note-se que nenhum pseudónimo é criado para a tarefa *Analisar*, porque as duas perspectivas estão já integradas pelo domínio pseudónimo *Robô*. Na prática, os pseudónimos permitem cruzar perspectivas.

3.3.2 Criação dinâmica de recursos

Os recursos lógicos são, praticamente na sua totalidade, criados durante a execução das aplicações, de acordo com o código escrito pelo programador; à excepção dos operões e tarefas automaticamente criados no arranque de uma aplicação (ver secção 7.3.3), todas as entidades são criadas através de primitivas específicas evocadas a partir de tarefas.

O procedimento para a criação de um recurso lógico, para além de uma lista de propriedades, exige a especificação do identificador da entidade sob a qual o recurso vai ser criado (o ascendente pretendido) e, no caso de se tratar da criação de um pseudónimo, dos identificadores de todos os originais envolvidos. A criação do domínio pseudónimo *Recolha*, na figura 3.8, por exemplo, exigirá o conhecimento dos identificadores do domínio lógico *Cluster* e dos domínios físicos *Nó* A_1 e *Nó* A_2. A obtenção dos identificadores necessários à especificação do ascendente e dos originais pressupõe, eventualmente, a descoberta dos recursos alvo numa hierarquia $m_{\varepsilon}\mu$, com base em propriedades previamente conhecidas.

Quando as aplicações solicitam a criação de operões, caixas postal e agregadores de memória, o sistema de exploração é responsável por descobrir um domínio que represente um sistema de computação (um nó do *cluster*). De facto, os programadores podem especificar um domínio de alto-nível que englobe múltiplos domínios representativos de nós do *cluster*.

Considerando o exemplo da figura 3.8, um operão *Robô* será criado partindo da indicação que o ascendente é o domínio pseudónimo *Recolha*. Com base neste domínio, o sistema de exploração facilmente descobrirá os domínios originais, bastando escolher um de entre os dois nós computacionais em causa. No entanto, numa hierarquia de recursos mais elaborada, como é o caso da apresentada na figura 3.9, a descoberta de um domínio representativo de uma máquina será mais complicada.

O sistema de exploração deverá, portanto, percorrer a hierarquia $m_{\varepsilon}\mu$ existente num dado momento da execução de uma aplicação, por forma a localizar domínios adequados para a criação efectiva de operões, caixas postal e agregadores. Depois de descoberto o domínio

alvo, o sistema de exploração encarrega-se de criar e registar no directório o recurso lógico em causa. Haverá ainda lugar à criação de um pseudónimo para esse recurso, conforme já referido.

Tendo em conta que as tarefas das aplicações $m_{\varepsilon}\mu$ executam concorrentemente e dispersas pelos nós do *cluster* e dado que são as tarefas quem despoleta a criação de outras entidades lógicas, facilmente se conclui que a criação e registo de recursos lógicos se faz de forma completamente distribuída e assíncrona. Nestas circunstâncias, um serviço de directório distribuído que suporte o registo local de todas as entidades criadas num dado nó do *cluster* contribuirá, decisivamente, para o bom desempenho deste sistema.

3.4 Sistemas de aplicações

Numa pequena aplicação paralela/distribuída é natural que todos os componentes sejam desenhados pelo mesmo programador. No entanto, algumas aplicações, por via da sua elevada complexidade, poderão motivar o envolvimento de vários programadores, sendo necessário integrar múltiplas aplicações mais pequenas, por forma a constituir um sistema de aplicações.

3.4.1 Cooperação interaplicação

No contexto de um sistema de aplicações, vários programadores deverão ser capazes de desenvolver aplicações cooperantes, tendo conhecimento, unicamente, das características e da funcionalidade básica das aplicações desenvolvidas por outros. As distintas aplicações poderão, ainda, no âmbito de um sistema operativo multi-utilizador, executar sob o controlo de vários utilizadores. No $m_{\varepsilon}\mu$, o sistema de aplicações, como um todo, é suportado por um única hierarquia. Com efeito, quaisquer aplicações em execução num *cluster*, num dado momento, serão traduzidas numa única hierarquia de recursos, independentemente de integrarem o mesmo sistema de aplicações ou não, isto é, independentemente do seu nível de cooperação.

Aplicações independentes podem cooperar entre si desde que sejam capazes de localizar pontos de entrada, no universo de componentes software que executam na totalidade dos nós de um *cluster*, que facultem algum tipo interacção. Estes pontos de entrada serão as entidades que permitem a troca de mensagens e o acesso a zonas de memória remotas. Dado que todas as entidades lógicas se encontram registadas no directório, podendo ser facilmente localizadas de acordo com propriedades previamente definidas e divulgadas, será apenas necessário dar a conhecer, como base em mecanismos externos ao $m_{\varepsilon}\mu$, a forma como esses pontos de entrada devem ser utilizados (que tipo de mensagens um

componente espera receber, que género de informação é disponibilizada numa zona de memória, etc.).

Os paradigmas para a cooperação interaplicação são, portanto, a passagem de mensagens e a memória global. Assim, uma aplicação deverá solicitar a descoberta de domínios, operões, tarefas e caixas postal específicos, pertencentes a outra aplicação, por forma a iniciar uma relação de cooperação baseada na troca de mensagens, com essa aplicação. Do mesmo modo, com base nos identificadores de blocos de memória e agregadores de memória externos, uma aplicação poderá ler e escrever dados de outras aplicações.

Formas de cooperação mais complexas podem ainda ser conseguidas, com base na possibilidade de criação de pseudónimos cujos originais pertencem a outros componentes aplicacionais. A criação de entidades sob domínios, operões e agregadores de memória pertencentes a outras aplicações permite que uma aplicação misture a sua hierarquia com outras, para obter uma outra forma de cooperação. Na secção 4.3 são apresentadas, com maior detalhe, algumas abordagens que tiram partido desta facilidade. A título de exemplo, se uma aplicação criar um pseudónimo de uma sua tarefa no contexto de um domínio externo (pertencente a uma outra aplicação), obterá uma cópia de todas as mensagens endereçadas a esse domínio, criando assim condições para a cooperação.

3.4.2 Partilha de recursos físicos

A execução simultânea de múltiplas aplicações requer metodologias para controlar a forma como os recursos físicos do *cluster* são partilhados.

Quando uma hierarquia $m_{\varepsilon}\mu$ compreende entidades lógicas, de uma ou mais aplicações, de alguma forma criadas sob o controlo de um único utilizador/programador, a alocação de recursos ou domínios de recursos a entidades lógicas está, em certa medida, facilitada. No entanto, se vários utilizadores executam aplicações desenvolvidas de forma independente, e que, provavelmente, não se enquadram num único sistema de aplicações, torna-se necessário dispor de mecanismos de alocação e reserva de recursos.

O $m_{\varepsilon}\mu$ usa um mecanismo simples de alocação de recursos; o acto de criar uma entidade lógica sob um determinado domínio físico corresponde, automaticamente, à apropriação de recursos. Para o efeito, a criação de uma entidade lógica implica a associação de uma propriedade especial relacionada com a propriedade que efectivamente descreve o recurso a consumir. A localização de recursos poderá depois entrar em consideração com ambas as propriedades.

Relativamente à figura 3.9, a criação de uma vista com garantia de 8192 unidades de memória deverá ter por base o requisito (`Mem - Mem' ≥ 8192`), significando que a totalidade da memória deverá estar disponível. Deste modo, o domínio *Vista X* não cumprirá

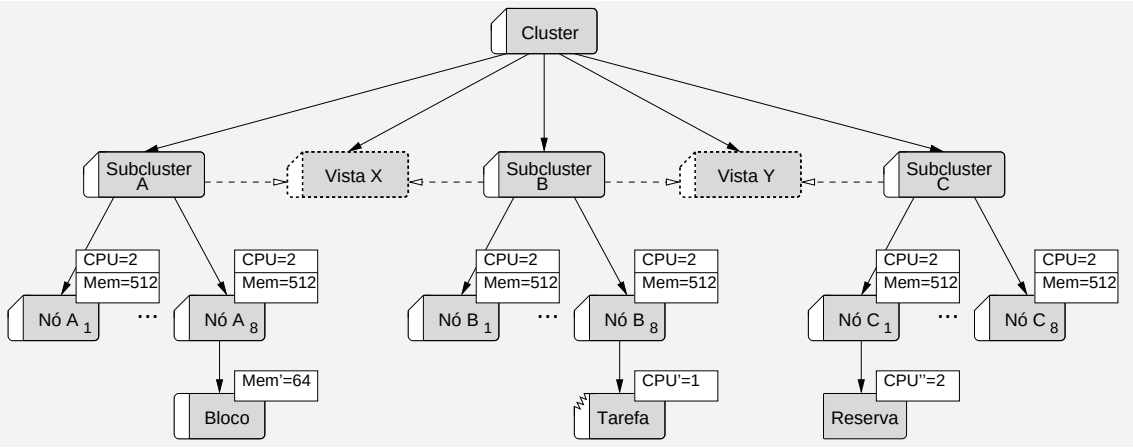


Figura 3.9: Alocação e reserva de recursos físicos.

tais requisitos, em virtude de o bloco descendente do domínio *Nó* A_8 incluir uma propriedade implícita (`Mem'=64`) que representa a apropriação de 64 unidades de memória.

O mecanismo de reserva é ligeiramente mais complexo, na medida em que é necessário marcar os recursos como ocupados, sem que as entidades lógicas que efectivamente irão usar esses recursos estejam ainda criadas. O $m_{\varepsilon}\mu$ utiliza uma entidade virtual – o subtractor – que, ao ser criada como descendente de um determinado domínio físico, conduz à subtracção de recursos, de acordo com as propriedades que lhe forem associadas.

Ainda em relação à figura 3.9, a criação de uma vista com 31 processadores disponíveis deverá ter por base o requisito (`CPU - CPU' - CPU'' ≥ 31`), significando que nenhum processador poderá estar ocupado ou reservado. Assim, o domínio *Vista Y* não irá satisfazer tais requisitos, apesar de apenas um dos 32 processadores estar ocupado por via da tarefa *Tarefa* (e da sua propriedade implícita `CPU'=1`), dado que dois processadores do domínio *Nó* A_1 estão reservados por intermédio do subtractor *Reserva* (e da propriedade associada `CPU''=2`).

Uma funcionalidade fundamental do $m_{\varepsilon}\mu$ prende-se com a possibilidade de, numa única operação, solicitar a criação de uma vista com determinados requisitos e simultaneamente proceder à reserva de recursos com base numa ou mais propriedades.

3.4.3 Controlo do acesso a recursos

A interoperação de componentes aplicacionais bem como a utilização dos recursos físicos de um *cluster* carecem de mecanismos de controlo de permissões. De facto, por um lado, as aplicações em execução num *cluster* num dado momento não têm, obrigatoriamente, que

integrar um só sistema aplicacional. Por conseguinte, um programador/utilizador poderá ter interesse em vedar o acesso de outras aplicações às entidades lógicas que constituem a sua aplicação. Por outro lado, determinados recursos físicos do *cluster* poderão, por iniciativa do administrador, estar disponíveis apenas para alguns utilizadores.

O mecanismo de controlo de permissões do $m_{\varepsilon}\mu$ baseia-se, em primeiro lugar, na associação de listas de controlo de acesso a cada uma das entidades de uma hierarquia. Basicamente, logo na criação das entidades físicas, o administrador indica, para cada entidade, uma lista de controlo de acesso, por forma a determinar que utilizadores (ou grupos de utilizadores) terão acesso a cada um dos recursos físicos do *cluster*. Posteriormente, quando uma aplicação solicita a criação de uma qualquer entidade lógica como descendente de um domínio físico, o identificador do utilizador que governa a execução da aplicação é usado para averiguar, perante a lista de controlo de acesso do ascendente pretendido, se a operação é permitida. O mesmo se passa relativamente aos originais pretendidos para a criação de um pseudónimo (criação de vistas para o hardware do *cluster*).

A criação de uma entidade lógica também engloba a especificação de uma lista de controlo de acesso. Desta forma, a criação de entidades, por parte de uma aplicação de um dado utilizador, que referenciem como ascendente ou como originais entidades lógicas de aplicações pertencentes a outros utilizadores, ficará sujeita ao mesmo mecanismo de validação usado na apropriação de recursos físicos. O envio de mensagens e a leitura/escrita de zonas de memória remota também seguem o mesmo sistema de controlo, no que diz respeito ao destinatário das operações em causa.

3.5 Epílogo

No contexto dos sistemas de computação mais comuns, a execução de aplicações com fortes exigências ao nível do desempenho é geralmente suportada por imagens de sistema uno poderosas, como é o caso do Gobelins [Gallard 02]. A gestão de recursos é feita de forma transparente às aplicações, com o intuito de garantir uma elevada disponibilidade e esconder os detalhes (baixo-nível) da arquitectura do sistema de computação.

A abordagem aqui apresentada assenta numa imagem de sistema uno minimalista, ao nível da camada de comunicação. Com base na funcionalidade fornecida a esse nível, são implementadas abstracções de alto-nível, no sentido de dotar os programadores de mecanismos para uma gestão consciente dos recursos físicos do *cluster*. Isto significa que, ao contrário do que acontece nas abordagens tradicionais, os programadores podem optar por uma gestão de recursos mais próxima do hardware, por forma a potenciar níveis mais elevados de desempenho nas aplicações.

No âmbito da metacomputação, têm sido propostas arquitecturas genéricas para a gestão

de recursos, como se pode constatar em alguns trabalhos desenvolvidos no âmbito da *Grid* [Czajkowski 98, Czajkowski 01]. O $m_{\varepsilon}\mu$ foi desenhado para *clusters* SMP multi--SAN, que são sistemas consideravelmente mais simples. No entanto, pela exploração do conceito de recurso, visando englobar as entidades físicas e lógicas que traduzem a operação de um *cluster*, e com a integração, num conjunto básico de abstracções, da *(i)* representação de recursos físicos, da *(ii)* modelação de aplicações e da *(iii)* forma de fazer corresponder componentes aplicacionais a recursos físicos, a abordagem aqui apresentada está longe de ser uma mera adequação de modelos e técnicas mais genéricas.

Capítulo 4

Modelação de aplicações

O modelo de programação do $m_{\varepsilon}\mu$ introduz uma nova abordagem, para o desenho de sistemas de aplicações, totalmente compatível com os conceitos contemplados em abordagens anteriores. Na verdade, as abstracções do $m_{\varepsilon}\mu$ foram desenhadas tendo em conta os aspectos mais notáveis de soluções amplamente divulgadas no panorama da programação de aplicações paralelas/distribuídas.

Neste capítulo mostra-se de que forma o $m_{\varepsilon}\mu$ permite desenvolver aplicações que tiram partido dos paradigmas de programação tradicionais. A vantagem de se optar pela utilização do $m_{\varepsilon}\mu$, em alternativa a outras plataformas de uso generalizado, prende-se com o facto de o $m_{\varepsilon}\mu$ expandir os mecanismos convencionais da memória partilhada, passagem de mensagens e memória global, integrando-os no paradigma da orientação ao recurso.

4.1 Suporte aos paradigmas convencionais

A aprendizagem de uma nova linguagem ou paradigma de programação justifica-se apenas quando tal investimento puder ser imediatamente rentabilizado. Por isso, qualquer nova abordagem deverá adequar-se às capacidades adquiridas pelos programadores ao longo de vários anos de utilização de tecnologias convencionais. No $m_{\varepsilon}\mu$ são oferecidas abstracções e funcionalidades que se enquadram nos três paradigmas de programação paralela mais utilizados.

4.1.1 Memória partilhada

Os vários componentes hardware de um sistema de computação têm um funcionamento inerentemente paralelo, para além de que as arquitecturas actuais exploram o paralelismo ao nível do processador através de técnicas tais como o encadeamento, a execução si-

multânea e o *hiperthreading*. No desenvolvimento de uma aplicação paralela, o primeiro nível de paralelismo que pode ser identificado num *cluster* SMP multi-SAN emerge da existência de múltiplos processadores em cada nó.

O paralelismo intranó pode ser explorado pela criação de múltiplas tarefas no contexto de um recurso operão, em que cada tarefa corresponde a um fio-de-execução de nível sistema. Os fios-de-execução de nível sistema, ao contrário dos de nível utilizador, estavam conotados com baixo desempenho, motivado pelos elevados tempos de comutação de contexto, apesar de possibilitarem a utilização dos vários processadores de um sistema SMP e suportarem a execução assíncrona de primitivas de sistema. Mais recentemente, com o aparecimento da biblioteca NPTL no RedHat Linux 9.0, os fios-de-execução de nível sistema tornaram-se uma alternativa viável devido aos significativos melhoramentos ao nível do desempenho [Drepper 03].

A cooperação entre as tarefas de um operão faz-se através de estruturas de dados partilhadas, criadas no contexto do operão, o qual é materializado, de forma transparente, num processo. A sincronização entre as tarefas, no que diz respeito ao acesso aos dados partilhados, é feita por meio das primitivas POSIX de manipulação de fios-de-execução.

O suporte para múltiplos fios-de-execução oferecido pelo R_OCl permite que os programadores combinem técnicas bem conhecidas de programação de aplicações para ambientes SMP – fios-de-execução POSIX – com os restantes paradigmas suportados pelo $m_\varepsilon\mu$, nomeadamente a passagem de mensagens e a memória global.

4.1.2 Passagem de mensagens

A abordagem do $m_\varepsilon\mu$ amplia o modelo tradicional da passagem de mensagens, integrando, no processo de comunicação, diferentes tipos de recursos usados na modelação de aplicações, criando a noção de recurso comunicante. De facto, apesar de todas as mensagens serem geradas por tarefas, que podem usar os seus identificadores ou identificadores correspondentes a caixas postal como origem da mensagem, o destino de uma mensagem pode ser uma tarefa, um operão, um domínio ou uma caixa postal.

Quando o destino de uma mensagem é uma tarefa, é suficiente o mecanismo de comunicação oferecido pelo R_OCl (ver figura 4.1-*a)*). Se o destino da mensagem for um pseudónimo de uma tarefa, a mensagem deverá ser encaminhada para o original dessa tarefa, o que requer funcionalidade acrescida (ver figura 4.1-*b)*). Se o original for, por sua vez, um pseudónimo, então será necessário um novo encaminhamento (ver figura 4.1-*c)*).

Quando o destino de uma mensagem é um operão, então qualquer tarefa sua descendente que não seja um pseudónimo poderá competir pelo acesso à mensagem. Assim, o operão pode ser visto como um repositório da cópia única da mensagem, que será consumida por

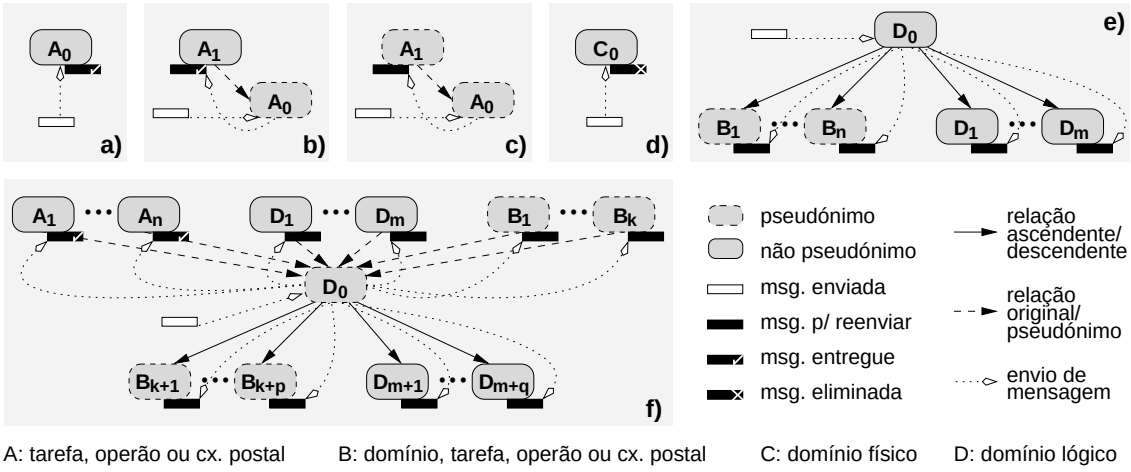


Figura 4.1: Cenários possíveis para a entrega de mensagens.

uma das tarefas descendentes. Se o operão for um pseudónimo, a mensagem terá que viajar até ao destino da forma descrita para as tarefas.

A caixa postal, quando usada como destino de uma mensagem, tal como o operão, armazena a cópia única da mensagem. Neste caso, porém, o universo de tarefas que podem reclamar a mensagem é mais alargado que no caso do operão; qualquer tarefa incluída na subárvore definida pelo ascendente da caixa postal poderá reclamar a mensagem.

Quando uma mensagem é endereçada a um domínio, é enviada uma cópia a cada um dos seus descendentes com capacidade para recepção de mensagens (tarefas, operões, caixas postal e domínios) e, se o domínio for um pseudónimo, a mensagem também é enviada a cada um dos seus originais (ver figuras 4.1-*e),f)*). Qualquer mensagem endereçada a um domínio que represente um recurso físico será descartada (ver figura 4.1-*d)*).

4.1.3 Memória global

A criação de um espaço de endereçamento global é essencial para a execução de aplicações com elevados requisitos de memória, para além de constituir um mecanismo de exploração do paralelismo no domínio dos dados. A utilização de sistemas de memória partilhada distribuída, para esse efeito, acarreta habitualmente algoritmos de sincronização onerosos. Os mecanismos de memória global, baseados nas operações de leitura/escrita remota das bibliotecas de comunicação de baixo-nível, têm vindo a ganhar especial relevância, tornando-se uma alternativa à memória distribuída partilhada.

No $m_{\varepsilon}\mu$ os blocos de memória criados ao nível dos operões podem ser acedidos remotamente, de forma individualizada, e podem ainda ser unificados através de um agregador

de memória. O mecanismo de agregação implica a criação de um pseudónimo descendente do agregador, para cada bloco de memória (ver figura 4.2).

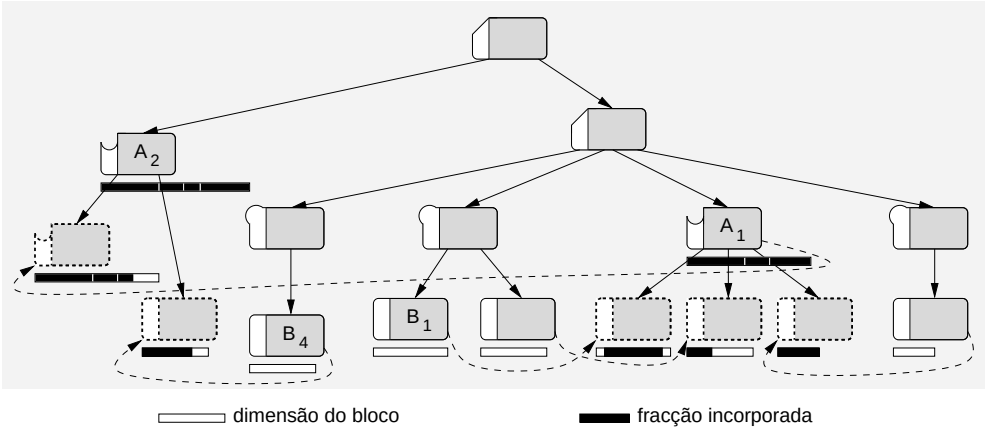


Figura 4.2: Exemplo de agregação de blocos de memória.

Aquando da adição de um bloco de memória a um agregador, ou seja, na criação de um pseudónimo de um bloco sob um agregador, por omissão, é criado um tuplo do tipo $\langle início, tamanho, identificador \rangle$, onde *início* diz respeito ao tamanho actual do espaço de endereçamento global, iniciado a zero no momento da criação do agregador, *tamanho* corresponde à dimensão do bloco e $identificador$ é o identificador global do bloco. Neste mecanismo, a ordem pela qual os blocos são adicionados ao agregador é decisiva para a sequenciação do espaço de endereçamento global. Opcionalmente, a posição que o bloco deve ocupar pode ser especificada, sendo também possível definir a fracção do bloco que deve ser incorporada no agregador. Neste caso, o programador é responsável por preencher a totalidade do espaço de endereçamento global. Na figura 4.2, apenas uma fracção do bloco B_1, por exemplo, é adicionada ao agregador A_1.

Um grupo de blocos representado por um agregador pode também ser integrado, de forma atómica, num outro agregador. Neste caso, é imprescindível especificar a fracção da sequência de blocos – primeiro agregador – que se pretende acrescentar ao segundo agregador, por forma a evitar que a expansão do primeiro interfira com o posicionamento de outros blocos no segundo agregador. Na figura 4.2, se ao agregador A_1, integrado no agregador A_2, for posteriormente adicionado um novo bloco, tal não deverá interferir com a posição ocupada pelo bloco B_4 no contexto do agregador A_2.

Para obter acesso a uma zona da memória global, o programador deverá começar por obter um apontador para uma zona de memória local contígua, através da indicação do identificador do agregador e dos limites inferior e superior que definem o fragmento da memória global desejado. De seguida, é possível actualizar, total ou parcialmente, de

acordo com os dados armazenados na memória global, a zona de memória local, através de operações *get*, que desencadearão as leituras remotas dos vários blocos de memória envolvidos. A actualização complementar é efectuada através de operações *put*.

O acesso em regime de exclusão mútua a uma dada zona da memória global pode ser efectuado através de operações *lock* e *unlock*, as quais implementam um mecanismo básico de sincronização, à custa do agregador, que constitui um elemento centralizador.

4.2 Um exemplo de aplicação

Com o intuito de mostrar a aplicação prática dos conceitos e abstracções $m_{\varepsilon}\mu$, apresenta-se aqui um exemplo de modelação de um sistema de aplicações para suporte a um ambiente escalável de recuperação de informação da *Web*. Este sistema de aplicações é uma simplificação do sistema proposto no projecto SIRe[1], visando exclusivamente a apresentação das abstracções propostas no $m_{\varepsilon}\mu$; abordagens específicas, e naturalmente mais elaboradas, para a recuperação de informação da *Web* podem ser encontradas em [Brin 98, Cho 02].

4.2.1 Especificação do hardware envolvido

A modelação estrita de uma aplicação $m_{\varepsilon}\mu$ é independente do sistema de computação disponível para a execução dessa aplicação. De igual forma, o processo de correspondência entre recursos lógicas e físicos poderá, de acordo com a estratégia traçada pelo programador, contemplar diferentes configurações de hardware. No entanto, a exploração eficiente de um *cluster* SMP multi-SAN requer o conhecimento da forma como se encontram estruturados os respectivos recursos físicos.

Na figura 4.3 é apresentada uma hierarquia de recursos, que representa o *cluster* ilustrado na figura 3.1(a), o qual irá servir de plataforma alvo para a execução da aplicação.

Na hierarquia, o segundo nível corresponde aos *subclusters* que compõem o sistema, os quais resultam da utilização de diferentes tecnologias de interligação de nós. Para além das propriedades usadas para a caracterização individual de cada um dos nós computacionais ($A_1...A_n$, $B_1...B_m$ ou $C_1...C_k$) e para a classificação dos *subclusters* de acordo com as tecnologias de comunicação, são ainda usadas propriedades `GFS=...` para a definição da forma de acesso a sistemas de ficheiros globais. A existência de uma rede Fast Ethernet, que interliga todos os nós do *cluster*, permite que seja criado um sistema de ficheiros global, comum a todos os nós, descrito pela propriedade `GFS=/ethfs`. Em cada *subcluster* poderão, também, ser instalados sistemas de ficheiros globais para explorar as tecnologias de comunicação aí existentes (propriedades `GFS=/myrifs` e `GFS=/gigfs`). Cada nó do

[1]Um projecto de investigação financiado pela FCT/MCT, sob o contrato POSI/CHS/41739/2001.

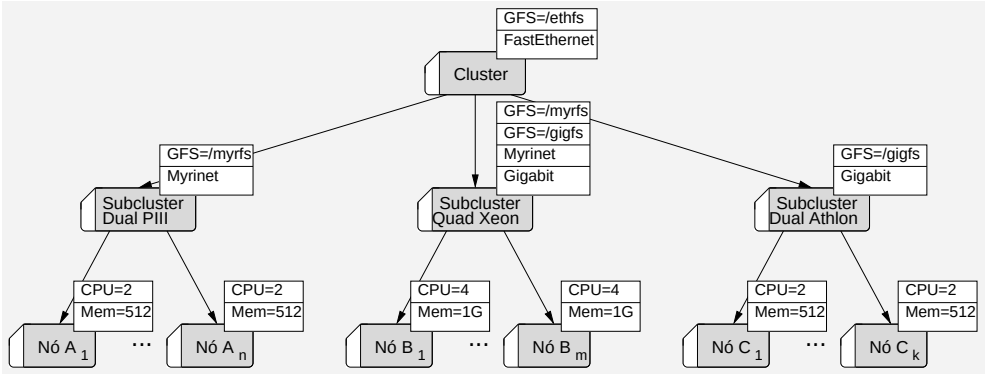


Figura 4.3: Hierarquia de recursos de um *cluster*.

cluster herdará os pontos de entrada dos sistemas de ficheiros que lhe dizem respeito.

4.2.2 Modelação da aplicação

A figura 4.4 apresenta a hierarquia de entidades lógicas usada para modelar a plataforma de recuperação de informação da *Web*. O sistema de aplicações em causa é representado pelo domínio lógico *SIRe*, que compreende três subsistemas representados pelos domínios *Recolha*, *Indexação* e *Resposta*.

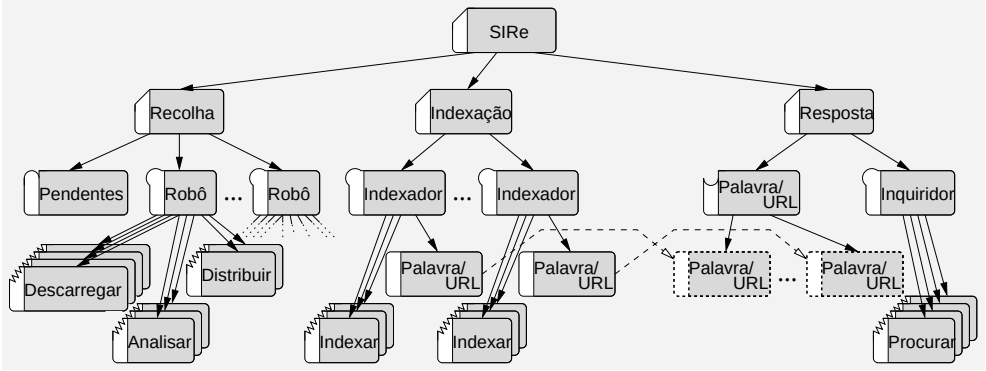


Figura 4.4: Exemplo de modelação do sistema SIRe.

Recolha

Cada operão *Robô* representa uma réplica de um robô em execução numa única máquina (um nó do *cluster*). Cada um dos estágios da recolha *Web* é suportado por uma bateria

de tarefas, por forma a explorar as potencialidades de cada nó SMP. Obviamente, estágios fortemente dependentes de operações de entrada/saída, como acontece com a descarga de páginas, comportam um maior número de tarefas.

Em cada estágio, as várias tarefas competem entre si por trabalho. As tarefas do estágio que corresponde à análise do conteúdo das páginas, por exemplo, partem do estado de repouso e, assim que existir uma página descarregada, uma das tarefas, de acordo com a estratégia de escalonamento do sistema operativo, inicia o processamento da página.

As tarefas correspondentes aos diferentes estágios de um robô são sincronizados através de estruturas de dados partilhadas ao nível do contexto do operão *Robô* e através de primitivas POSIX de sincronização de fios-de-execução. Desta forma, cada réplica do robô explora uma estação de trabalho SMP através do paradigma da memória partilhada.

No âmbito do domínio *Recolha*, os vários robôs cooperam com base na partição do universo de URLs. Esta partição pode ser efectuada por via de uma função de dispersão[2], a aplicar a cada um dos URLs. Esta estratégia requer protocolos adicionais, para suportar, por exemplo, a variação do número de entidades cooperantes, que naturalmente não são aqui analisados.

Para distribuir os URLs, após a análise do conteúdo de cada uma das páginas descarregadas, o estágio *Distribuir* envia a cada operão *Robô* uma mensagem com os URLs atribuídos de acordo com a função de dispersão. Assim, às tarefas *Descarregar* de um dado operão *Robô* cabe o papel de processarem, concorrentemente, as várias mensagens que vão chegando, por forma a determinarem que URLs devem ser descarregados.

Devido ao facto de nenhum esquema de partição do género do descrito garantir, por si só, o perfeito balanceamento dos operões, no que diz respeito ao número de URLs a descarregar, as tarefas *Descarregar* podem enviar URLs excedentários para a caixa postal *Pendentes*. Esta caixa postal pode ser acedida por qualquer tarefa *Descarregar* que se encontre ociosa.

A cooperação entre robôs é pois conseguida através da passagem de mensagens, quer sejam mensagens trocadas de forma directa – mensagens enviadas por tarefas *Distribuir* a operões *Robô* – ou mensagens trocadas indirectamente através da caixa postal *Pendentes*.

Indexação

O sistema de indexação, representado pelo domínio *Indexação*, tem como propósito manter uma matriz cruzando palavras chave e URLs, tendo em vista a determinação expedita dos URLs relevantes para um dado conjunto de palavras chave. A elevada quantidade de memória necessária ao armazenamento de tal matriz obriga à reunião das disponibilidades de memória de um conjunto alargado de nós do *cluster*.

[2]Função de *hash*.

Com o intuito de reunir múltiplos fragmentos de memória, são criados alguns operões *Indexador*, cada um com um bloco de memória. Cada indexador manipula uma colecção de URLs, de acordo com um esquema de partição básico. Essa colecção de URLs corresponderá a um conjunto contíguo de linhas da matriz referida, a armazenar no bloco de memória local ao indexador. Deste modo, os indexadores evitarão referências a posições de memória respeitantes a blocos remotos.

A integração do sistema de recolha com o sistema de indexação poderia fazer-se através de mensagens enviadas pelas tarefas *Analisar* aos operões *Indexador*. Tais mensagens, contendo o resultado da análise do conteúdo de páginas *Web*, seriam endereçadas com base no esquema de partição definido para divisão de trabalho entre indexadores.

No entanto, dado que as aplicações $m_{\varepsilon}\mu$ não estão limitadas aos mecanismos aqui apresentados, a integração dos dois sistemas poderá fazer-se por via de um sistema de ficheiros global. Neste cenário, as tarefas *Analisar* armazenariam os resultados da análise de páginas *Web* em ficheiros globais, os quais seriam lidos e processados pelas tarefas *Indexar* dos vários operões *Indexador*, de acordo com o esquema de partição definido.

Resposta

O sistema de resposta determina os URLs relevantes para uma dada palavra chave, utilizando a totalidade dos blocos de memória criados ao nível dos operões *Indexador* como se se tratasse de um único espaço de endereçamento global. Os blocos de memória são sequenciados através da criação de pseudónimos no contexto do agregador *Palavra/URL*, o qual será responsável por redireccionar referências de memória e por garantir exclusão, impedindo leituras, por parte das tarefas *Procurar*, quando as tarefas *Indexar* procedem, simultaneamente, a actualizações.

A consulta da matriz criada pelo sistema de indexação, com a finalidade de determinar os URLs para uma dada palavra chave, resultará em leituras remotas transparentes, ao longo de uma coluna da matriz. Assim, o sistema de resposta explora múltiplos nós do *cluster* através do paradigma da memória global.

4.2.3 Exploração do hardware disponível

A figura 4.5 mostra a integração da hierarquia obtida com a modelação da aplicação (figura 4.4) com a hierarquia representativa do hardware do *cluster* (figura 4.3). A árvore de recursos resultante será fruto do código escrito pelo programador para criação de entidades lógicas. De facto, no momento do arranque da aplicação existirá apenas a hierarquia de recursos físicos, criada previamente pelo administrador do *cluster*, cabendo à aplicação a tarefa de proceder à criação de recursos lógicos.

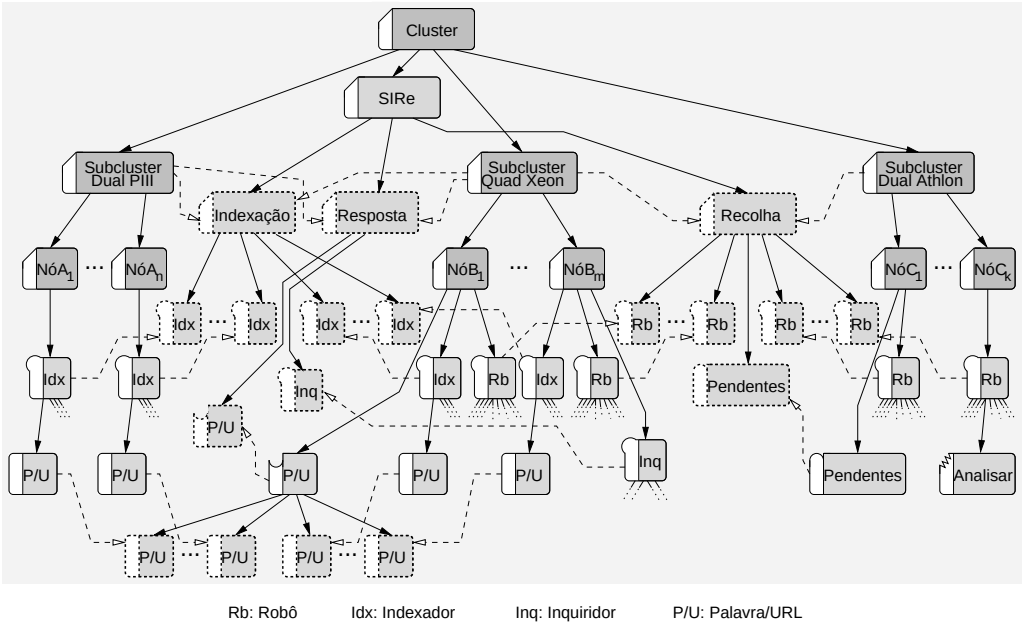


Figura 4.5: Correspondência entre recursos lógicos e físicos.

As opções do programador são, obviamente, fundamentais para obter uma correspondência de grão-fino óptima. De facto, se o programador decidir estruturar a sua aplicação com base, unicamente, no domínio raíz *Cluster*, desprezando a hierarquia de hardware existente, não conseguirá agrupar as várias tarefas da aplicação em consonância com a organização dos recursos físicos e, por conseguinte, não será capaz de explorar qualquer nível de localidade. Todavia, o $m_{\varepsilon}\mu$ é capaz de garantir que a aplicação é executada, embora com um desempenho inferior.

Apesar da aparente complexidade da hierarquia da figura 4.5, a criação das entidades lógicas, por parte do programador, ficará substancialmente facilitada após a criação dos domínios pseudónimo *Recolha*, *Indexação* e *Resposta*, os quais seleccionam os recursos físicos a ser utilizados por cada subsistema da aplicação, de forma a permitir que os nós do *cluster* sejam partilhados segundo duas estratégias distintas: partilha espacial e partilha temporal. O *subcluster Quad Xeon* é utilizado pelos três subsistemas – partilha temporal – enquanto que os *subclusters Dual PIII* e *Dual Athlon* são usados exclusivamente pelos subsistemas de recolha e de indexação, respectivamente – partilha espacial.

Cabe ao programador seleccionar os recursos físicos necessários a cada componente aplicacional, através da especificação de uma lista de requisitos e da evocação das primitivas $m_{\varepsilon}\mu$ correspondentes. A criação do domínio pseudónimo *Recolha*, por exemplo, terá por base os requisitos (`Gigabit`) $\wedge$ (`CPU=4*m+2*k`), significando que se pretende um conjunto de

nós interligados por Gigabit e com um total de $4m + 2k$ processadores, isto é, a totalidade dos processadores existentes nos *subclusters Quad Xeon* e *Dual Athlon* (ver figura 4.3).

Depois de delimitados os recursos físicos a utilizar em cada subsistema da aplicação, são criados os operões, caixas postal e agregadores de memória que integram a hierarquia de recursos lógicos, por forma a explorar efectivamente os nós do *cluster*.

4.3 Passagem de mensagens de alto-nível

O paradigma da passagem de mensagens é, indiscutivelmente, o mais vulgarizado entre os programadores que desenvolvem aplicações paralelas. O PVM e o MPI são uma referência em termos de conceitos e funcionalidade, independentemente de poderem ser usados outros sistemas para o desenvolvimento e execução de aplicações. Neste contexto, a apresentação de uma nova abordagem que integre a passagem de mensagens deverá contemplar o estabelecimento de uma correspondência com as possibilidades desses dois ambientes.

Em [Geist 96] e [Gropp 98] são examinados alguns aspectos relacionados com o desenvolvimento e execução de aplicações paralelas, com o intuito de comparar o MPI e o PVM. A modularidade, o dinamismo, os grupos e a interoperação são as principais vertentes da análise em causa, enquanto conceitos e abstracções fundamentais para qualquer sistema baseado na passagem de mensagens. O modelo de programação do $m_{\mathcal{E}}\mu$ integra mecanismos que, analisados segundo estas vertentes, permitem concluir que, para além da funcionalidade padrão dos sistemas tradicionais de passagem de mensagens, o $m_{\mathcal{E}}\mu$ proporciona abstracções de alto-nível singulares.

4.3.1 Dupla modularidade

No sentido de facilitar o desenvolvimento de aplicações paralelas, é comum a utilização de bibliotecas de funções que implementam algoritmos paralelos, tal como acontece na programação sequencial. No entanto, na programação de aplicações paralelas, a utilização de funções codificadas por outros obriga à definição de contextos de execução, de forma a impedir a recepção inapropriada de mensagens. O suporte à definição de contextos, enquanto regiões delimitadas para a troca de mensagens, é pois o mecanismo usado para garantir a modularidade básica das aplicações, isto é, para a divisão do código em módulos de funções (bibliotecas), permitindo a reutilização.

Pode ainda ser identificado um outro nível de modularidade, que se traduz na possibilidade de, em tempo de execução, substituir componentes de uma aplicação. A título de exemplo, imagine-se a substituição de um fio-de-execução de uma aplicação por um processo com vários fios-de-execução. A substituição de componentes pode ser entendida como um

mecanismo de suporte à evolução das aplicações, visto que um módulo codificado numa fase inicial pode posteriormente, já na fase de execução da aplicação, ser substituído por outro mais eficiente ou mais funcional.

No modelo de programação do $m_{\varepsilon}\mu$, as duas vertentes da modularidade são facilmente integradas. Nos sistemas tradicionais, apesar de a definição de contextos ser comum, a substituição de componentes não é contemplada.

4.3.2 Bibliotecas de funções

Numa aplicação $m_{\varepsilon}\mu$, a definição de uma região demarcada para a troca de mensagens pode ser conseguida de duas formas distintas:

- através da criação de uma caixa postal por cada tarefa;
- mediante a criação de um domínio contextualizador, abrangendo todas as tarefas.

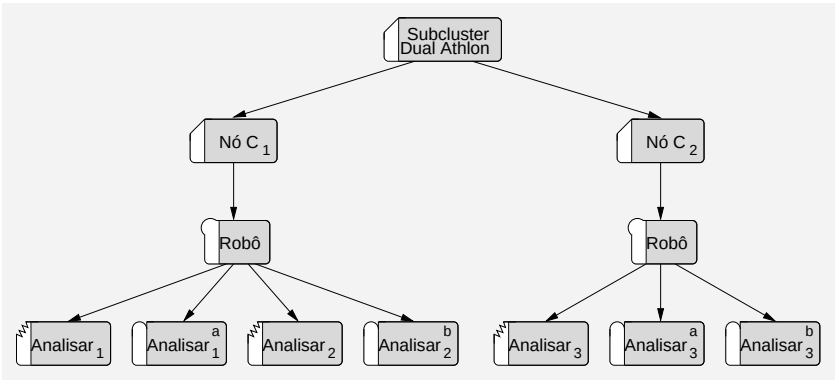
A utilização de caixas postal é, na verdade, uma forma de dotar cada tarefa de uma identificação adicional, dado que as tarefas podem enviar mensagens fazendo-se passar por outra entidade, desde que se trate de uma caixa postal, e podem naturalmente aceder às mensagens depositadas numa caixa postal.

A figura 4.6(a) apresenta um cenário onde três tarefas criam quatro caixas postal, de forma a usarem duas bibliotecas: as tarefas $Analisar_1$ e $Analisar_2$, por fazerem uso das bibliotecas a e b, respectivamente, criam as caixas postal $Analisar_1^a$ e $Analisar_2^b$, enquanto que a tarefa $Analisar_3$, pelo facto de usar ambas as bibliotecas, cria as caixas postal $Analisar_3^a$ e $Analisar_3^b$.

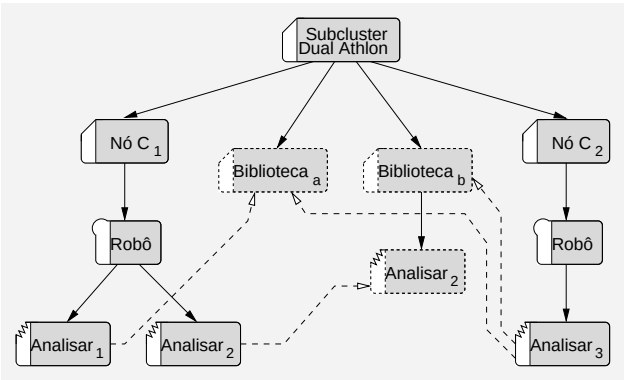
Note-se que, pela simples criação da caixa postal $Analisar_1^a$, por exemplo, não fica estabelecida qualquer associação à tarefa $Analisar_1$. Na verdade, qualquer tarefa do operão ascendente poderá aceder a esta caixa postal (ver secção 4.1.2), cabendo ao programador garantir os emparelhamentos adequados. No entanto, através do mecanismo de controlo de acesso (ver secção 3.4.3), pode ser indicado que apenas a tarefa criadora tem acesso à caixa postal criada para fazer face a uma dada biblioteca.

Os domínios contextualizadores, o equivalente aos comunicadores MPI ou contextos PVM, baseiam-se na possibilidade de duas tarefas trocarem mensagens usando, em vez das suas identificações globais, a identificação de um domínio conjuntamente com os seus identificadores de membro (no contexto desse domínio). A indicação, tanto no envio como na recepção, de um domínio distinto, por cada biblioteca em uso, possibilita diferenciação de universos de comunicação.

Na figura 4.6(b), é apresentada uma solução para o mesmo caso da figura 4.6(a), mas onde a criação de contextos se faz através de domínios pseudónimo. Assim, para cada uma das



(a)



(b)

Figura 4.6: Definição de contextos através de caixas postal e domínios.

bibliotecas, é criado um domínio pseudónimo e as tarefas que usam uma determinada biblioteca são feitas membros do domínio correspondente.

Saliente-se que o domínio $Biblioteca_a$ foi criado com base na especificação de todas as tarefas que usam a biblioteca a, as quais constituem o rol de originais desse domínio. No caso do domínio $Biblioteca_b$, apenas uma tarefa – a tarefa $Analisar_3$ – foi indicada no momento da criação do pseudónimo. Na verdade, o $m_{\mathcal{E}}\mu$ não obriga a que todas as tarefas que integram um dado contexto já existam no momento da criação do contexto, ao contrário do MPI, pois os descendentes de um domínio, que integram, por definição, a lista de membros desse domínio, podem ser criados dinamicamente. Assim, a tarefa $Analisar_2$, eventualmente criada após a definição do contexto $Biblioteca_b$, será integrada no domínio através do pseudónimo $Analisar_2$.

4.3.3 Substituição de componentes

A substituição de componentes tira partido de dois aspectos básicos do modelo de comunicação do $m_{\varepsilon}\mu$: *(1)* o envio de mensagens pressupõe a utilização de identificadores (das entidades envolvidas) obtidos através de pesquisas no directório, com base em características das entidades, e, na impossibilidade de entrega de uma mensagem a uma determinada entidade, *(2)* o sistema de comunicação notifica a aplicação, através do estado associado à mensagem, podendo esta desencadear um processo de reenvio, recorrendo novamente ao directório para localizar a entidade. Isto significa que, se uma entidade X (uma tarefa, por exemplo) for desactivada/eliminada e uma outra entidade Y, com o comportamento da primeira, for criada e registada, então todas as demais entidades da aplicação que pretendam trocar mensagens com X, após uma notificação de insucesso de envio (em relação ao destino X), poderão recorrer ao directório para descobrir Y, com quem passarão a comunicar.

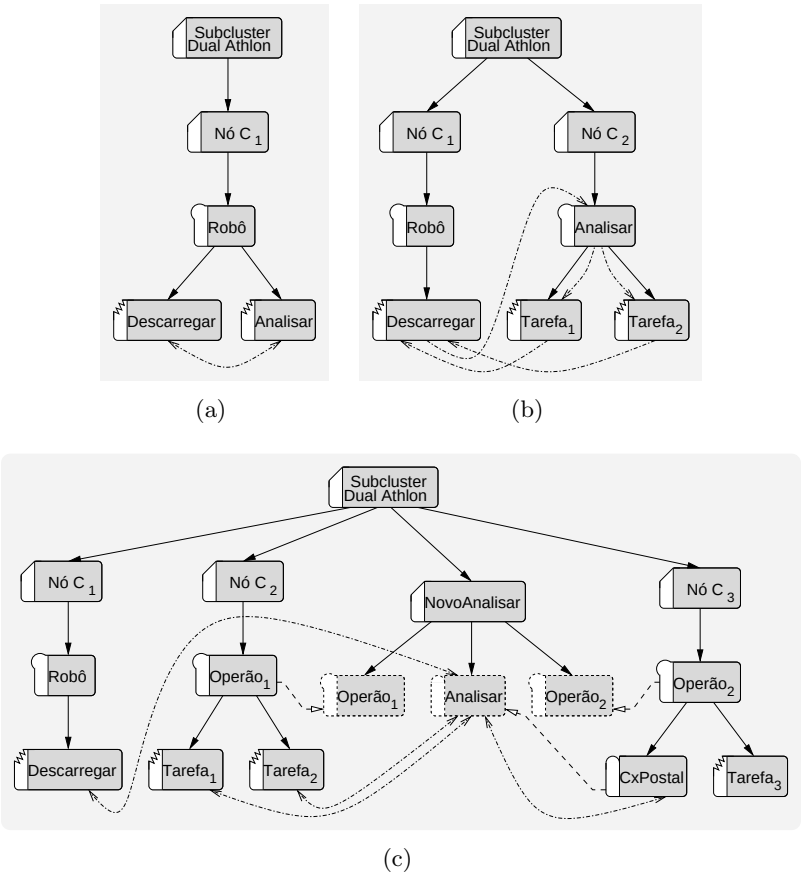


Figura 4.7: Substituição de componentes aplicacionais.

A figura 4.7 apresenta dois casos exemplificativos das implicações, do ponto de vista da

passagem de mensagens, resultantes da substituição de uma entidade simples (uma tarefa) por outras mais complexas (operões ou domínios). Partindo do cenário da figura 4.7(a), onde as tarefas *Descarregar* e *Analisar* estão envolvidas na troca de mensagens, as figuras 4.7(b) e 4.7(c) mostram de que forma a tarefa *Analisar* pode ser substituída por um operão, integrando várias tarefas em execução numa segunda máquina, ou por um domínio com tarefas espalhadas por dois operões em máquinas distintas.

No primeiro caso (figura 4.7(b)), a tarefa *Descarregar* passará a enviar mensagens para o operão *Analisar*, podendo qualquer uma das tarefas $Tarefa_1$ e $Tarefa_2$ vir a consumir cada uma dessas mensagens. No entanto, dado que o operão não é uma entidade válida como origem de mensagens, estas duas tarefas terão que enviar mensagens directamente para a tarefa *Descarregar*, o que obriga a alguma flexibilidade no código desta última. De facto, a tarefa *Descarregar* terá que estar preparada para receber mensagens sem especificar o identificador concreto da entidade *Analisar*, ou seja, as primitivas de recepção deverão especificar uma origem indeterminada para a mensagem.

No segundo caso (figura 4.7(c)), a tarefa *Descarregar* passará a enviar mensagens para a caixa postal *Analisar*, a qual poderá ser acedida por qualquer tarefa cujas cadeias de ascendência incluam o domínio *NovoAnalisar* – as tarefas $Tarefa_1$, $Tarefa_2$ e $Tarefa_3$. Estas tarefas poderão também usar a identificação dessa caixa postal, quando enviam mensagens para a tarefa *Descarregar*. Assim, a substituição da tarefa *Analisar* pelo domínio *NovoAnalisar* não dependerá de cuidados especiais na programação da tarefa *Descarregar*.

4.3.4 Aplicações dinâmicas

A possibilidade de dotar os programas de alguma capacidade de adaptação, em função de eventos externos, por exemplo, é fundamental em aplicações que executam em regime permanente ou que partilham os recursos físicos com outras. A criação dinâmica de processos e/ou fios-de-execução é comum a muitos sistemas de passagem de mensagens e constitui um mecanismo básico para o desenvolvimento de aplicações dinâmicas. No caso do PVM, a criação de processos (tarefas PVM) pode, inclusivamente, especificar as características do nó do *cluster* onde a acção deverá ter lugar.

No $m_\varepsilon\mu$, a hierarquia de um sistema de aplicações é, naturalmente, evolutível, uma vez que todas as entidades (domínios, operões, tarefas e caixas postal) são criadas dinamicamente. A própria correspondência entre recursos lógicos e físicos é igualmente dinâmica.

Evolução de uma hierarquia $m_\varepsilon\mu$

A figura 4.8 ilustra a evolução de uma hierarquia $m_\varepsilon\mu$ representativa de um sistema de robôs dinâmico.

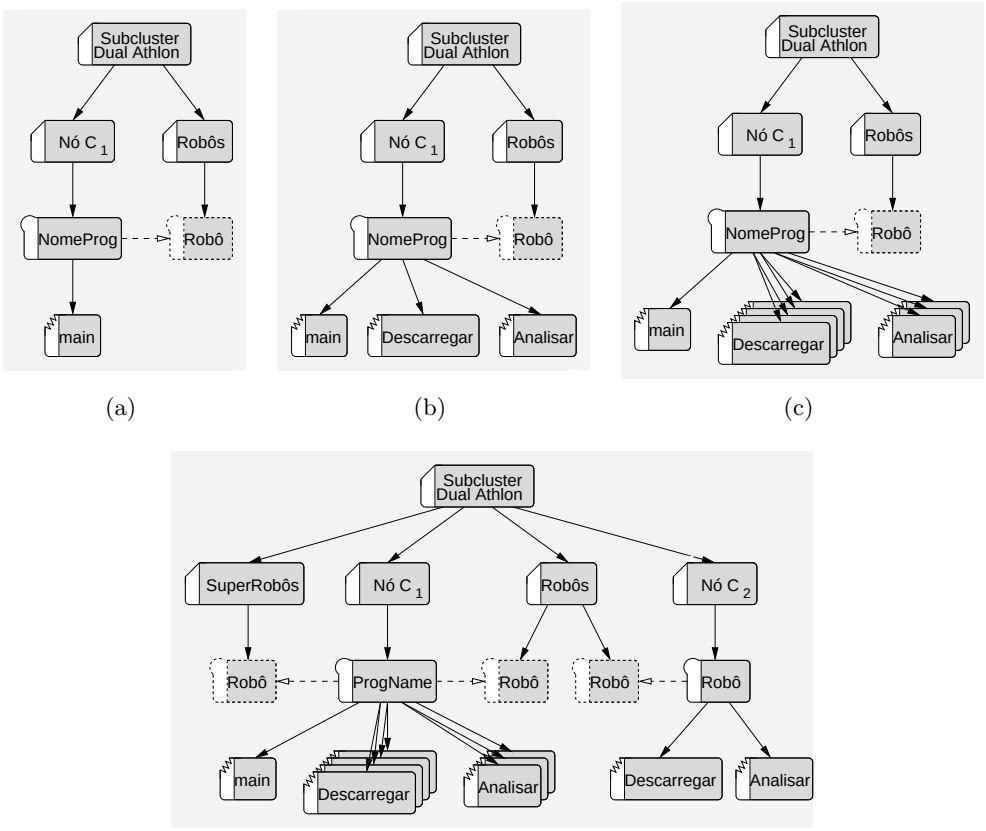


Figura 4.8: Evolução da hierarquia de uma aplicação.

O sistema inicia através do arranque de um dado programa via linha de comando. São então criados, automaticamente, um operão e uma tarefa, no domínio que representa a máquina a partir de onde se efectua o arranque do sistema (ver figura 4.8(a)). De seguida, a partir da tarefa *main*, é criado um domínio para a aplicação – domínio *Robôs* – ao qual é anexado, através de um pseudónimo, o operão referido. Já com base no operão pseudónimo *Robô*, a tarefa *main* poderá criar as tarefas *Descarregar* e *Analisar* (ver figura 4.8(b)).

Após um determinado período de operação, poderão as tarefas *Descarregar* e *Analisar* concluir que é necessário escalar o sistema, criando mais tarefas, no seio do mesmo operão inicial (ver figura 4.8(c)). Poderá ainda haver lugar à criação de um operão *Robô* numa segunda máquina, através de uma primitiva $m_{\varepsilon}\mu$, seguida da criação de duas tarefas (*Descarregar* e *Analisar*). Este operão é integrado no sistema de robôs através da criação de um pseudónimo sob o domínio *Robôs*. Por forma a distinguir o potencial dos vários robôs que vão integrando o sistema, poderá ser criado um novo domínio – o domínio *SuperRobôs*, ao qual é anexado o operão *Robô* inicial através de um pseudónimo (ver

figura 4.8(d)).

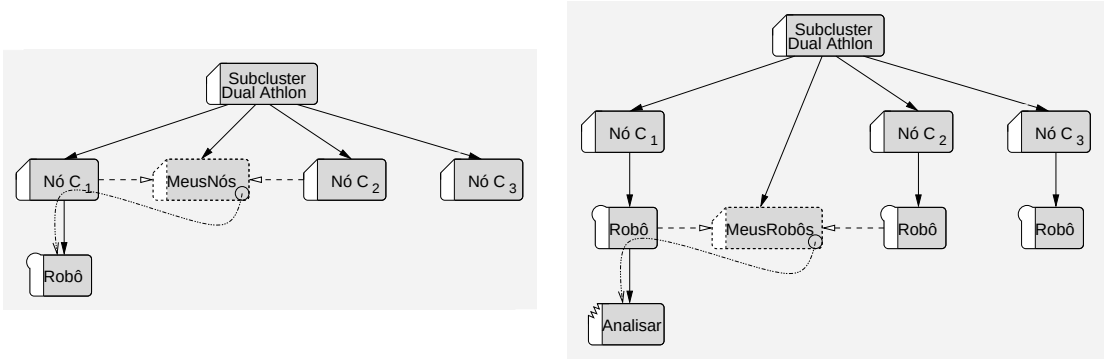
É importante referir que qualquer um dos estágios representados na figura 4.8 corresponde a uma aplicação em operação, tratando-se, portanto, de uma evolução em tempo de execução.

Delimitação de recursos

No $m_{\varepsilon}\mu$, a criação de uma entidade lógica sob um determinado recurso físico, a criação de um operão, por exemplo, faz-se através da indicação do identificador do domínio que representa esse recurso físico. Este identificador é obtido pelo programador, através do directório e com base na especificação de uma lista de propriedades, de entre aquelas definidas livremente pelo administrador. Assim, é possível especificar como alvo uma máquina que obedeça a critérios arbitrários, ao contrário do que se passa no PVM, por exemplo, onde apenas o tipo de arquitectura e o poder relativo do processador podem ser usados.

O paradigma da orientação ao recurso preconizado pelo $m_{\varepsilon}\mu$ generaliza o conceito de selecção de recursos, tratando de igual forma recursos físicos e lógicos. Deste modo, a selecção de um operão para a execução de uma tarefa, por exemplo, far-se-á nos mesmos moldes que a selecção de uma máquina para a criação de um operão.

Para além desta funcionalidade, o $m_{\varepsilon}\mu$ inclui ainda facilidades para delimitar recursos físicos, máquinas, por exemplo, com base em características comuns, e deixar a cargo do sistema a selecção de um domínio em particular (uma máquina específica), no seio do universo delimitado, para a criação de um recurso lógico. Esta abordagem é também alargada à delimitação de recursos lógicos.



(a) (b)

Figura 4.9: Delimitação de recursos.

A figura 4.9(a) mostra um domínio pseudónimo – o domínio *MeusNós* – usado para delimitar uma parte dos nós do *cluster*. Este domínio terá sido criado com base em características específicas indicadas pelo programador, podendo, portanto, corresponder aos sistemas que cumprem determinados requisitos necessários ao funcionamento da aplicação e não apenas a um mero subconjunto de nós. A criação do operão *Robô* no domínio *MeusNós* entrega ao $m_{\varepsilon}\mu$ a responsabilidade por seleccionar um dos domínios *Nó* C_1 ou *Nó* C_2, isto é, um dos nós do *cluster* delimitados pelo domínio *MeusNós*.

De igual forma, a figura 4.9(b) apresenta a criação de uma tarefa, tendo por base um domínio pseudónimo usado para delimitar parte dos operões existentes; a partir do pseudónimo, o $m_{\varepsilon}\mu$ seleccionará um dos operões originais para desencadear a execução da tarefa *Analisar*.

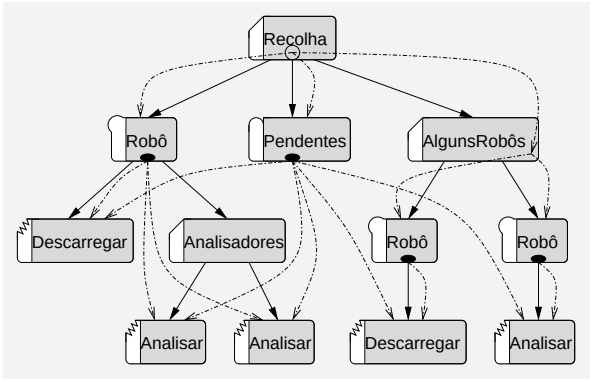
4.3.5 Grupos flexíveis

A constituição de grupos assume especial importância em aplicações que recorrem à difusão selectiva de informação. Vulgarmente, os sistemas de passagem de mensagens incluem primitivas para a criação de grupos, adesão (dinâmica) a grupos e difusão de mensagens. No $m_{\varepsilon}\mu$, por via da flexibilidade inerente à organização hierárquica de entidades e à criação de pseudónimos, são suportados a constituição dinâmica de grupos híbridos (contendo tarefas, operões, domínios e caixas postal), o encadeamento de grupos e a reestruturação de grupos. A figura 4.10 exemplifica as potencialidades do $m_{\varepsilon}\mu$ na manipulação de grupos e correspondente entrega de mensagens.

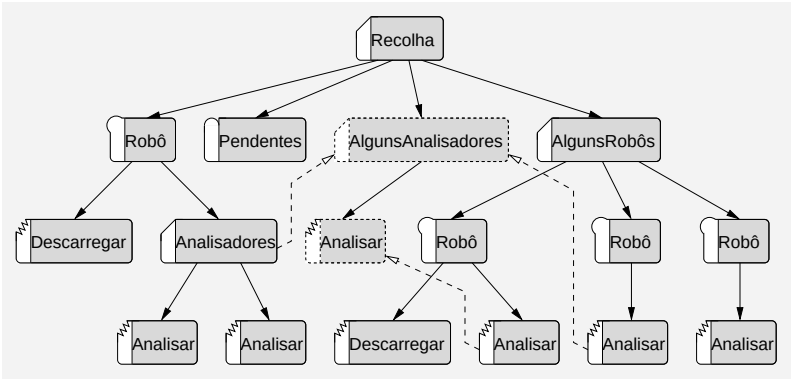
Na figura 4.10(a) podem ser identificados dois grupos, materializados nos domínios *Recolha* e *AlgunsRobôs*. O primeiro inclui como membros um operão, uma caixa postal e um domínio, tratando-se, portanto, de um grupo híbrido. O segundo grupo, que por sinal é membro do primeiro, inclui dois operões.

O envio de uma mensagem ao domínio *Recolha*, por parte de uma qualquer tarefa, incluída ou não nesse domínio, provocará a entrega de uma cópia a cada um dos seus membros. Dado que um dos membros em causa é o domínio *AlgunsRobôs*, será enviada uma nova cópia da mensagem aos membros deste último. Deste modo, todos os operões *Robô*, juntamente com a caixa postal *Pendentes*, receberão uma cópia da mensagem.

As quatro cópias da mensagem serão efectivamente consumidas por tarefas de entre as cinco existentes. Uma mensagem depositada num operão poderá ser consumida por tarefas presentes na subárvore por ele definida, enquanto que a mensagem depositada na caixa postal poderá ser consumida por qualquer tarefa. Obviamente, para além do facto de não existirem cópias da mensagem para todas as tarefas, nada impede que uma única tarefa consuma duas mensagens: uma no âmbito do seu operão e outra no âmbito da caixa postal.



(a)



(b)

Figura 4.10: Encadeamento de grupos híbridos.

Na figura 4.10(b) é constituído um grupo através da criação de um domínio pseudónimo. Este domínio traduz uma nova vista para as entidades representadas na figura 4.10(a), o que significa que, no que diz respeito à propagação de mensagens, estamos perante uma reestruturação do sistema de grupos dessa figura. Neste novo cenário, o envio de uma mensagem ao domínio *Recolha* produzirá um efeito diferente do descrito anteriormente; para além dos operões e da caixa postal, também as tarefas *Analisar*, com excepção da representada no canto inferior direito, receberão uma cópia da mensagem.

O domínio *AlgunsAnalisadores* reenviará a mensagem recebida do domínio *Recolha* a todos os seus membros, isto é, a todos os seus originais e descendentes. Entre os originais, para além de uma tarefa, encontra-se o domínio *Analisadores*, o qual, por sua vez, reenviará a mensagem às duas tarefas suas descendentes. Como descendente existe um pseudónimo de uma tarefa, o qual reenviará a mensagem ao seu original. As mensagens entregues aos operões e à caixa postal serão tratadas da forma já descrita para as mensagens do cenário apresentado na figura 4.10(a).

4.4 Epílogo

A modelação de aplicações paralelas tem sido um tema pouco debatido talvez pelo facto de os programadores usarem, maioritariamente, soluções baseadas em MPI ou PVM. De facto, estas plataformas reduzem as aplicações paralelas a meros conjuntos de processadores ou de processos, não se justificando, por isso, a criação de mecanismos muito elaborados para a estruturação das entidades comunicantes.

A hierarquia de recursos lógicos representativa de uma aplicação é um elemento fundamental para viabilizar a colaboração de múltiplos programadores no processo de desenvolvimento de um sistema de aplicações complexo. Neste contexto, as abstracções introduzidas pelo $m_{\varepsilon}\mu$ constituem uma ferramenta básica para a modelação de aplicações paralelas a alto-nível. A organização hierárquica da totalidade dos recursos lógicos permite uma visão única da aplicação, independentemente de poderem ser combinados diferentes paradigmas de programação, consoante a natureza do hardware ou por opção do programador.

Outro factor que valoriza o $m_{\varepsilon}\mu$ é o suporte à utilização de mecanismos convencionais, integrados no paradigma da orientação ao recurso através do enriquecimento de conceitos e abstracções de plataformas como o MPI e o PVM. Em particular, alguns dos mecanismos avançados de suporte à modularidade, ao dinamismo e à manipulação de grupos podem facilmente ser explorados à custa das abstracções do $m_{\varepsilon}\mu$. Isto facilita a reimplementação de aplicações já existentes e aumenta significativamente a aplicabilidade da plataforma $m_{\varepsilon}\mu$.

O suporte integrado à especificação de recursos físicos, modelação de aplicações e correspondência entre recursos físicos e lógicos torna o $m_{\varepsilon}\mu$ um sistema único; um conjunto reduzido de entidades, actualmente apenas sete entidades, e uma forma de organização única permitem unificar a modelação de aplicações e a exploração dos recursos de um *cluster* SMP multi-SAN.

Capítulo 5

Comunicação orientada ao recurso

A utilização eficiente de múltiplas tecnologias de comunicação, por parte de uma aplicação codificada numa determinada linguagem, tornou-se uma realidade com o aparecimento de bibliotecas/sistemas de comunicação de nível intermédio. Estes sistemas permitem que as aplicações explorem variados protocolos de comunicação de baixo-nível através de um interface único, garantindo assim que uma aplicação não fica confinada a um determinado *cluster*, interligado por uma tecnologia particular de comunicação.

Neste contexto, a necessidade de contemplar a utilização maciça de fios-de-execução, por forma a explorar nós SMP e introduzir novas facilidades para a modelação de um leque importante de aplicações, aliada à pretensão de ir de encontro ao paradigma da computação orientada ao recurso são as razões principais que levaram ao desenvolvimento da biblioteca R_OCl – *Resource oriented Communication library*. A especificidade desta biblioteca deriva da criação de uma imagem de sistema uno, na qual se baseia o sistema $m_\varepsilon\mu$ para oferecer conectividade total entre recursos lógicos de uma aplicação.

Neste capítulo é apresentado o modelo de comunicação orientada ao recurso e a funcionalidade básica do R_OCl. São também apresentados os mecanismos desenvolvidos para suporte à operação em ambiente *multicluster*.

5.1 Modelo de Comunicação

A biblioteca R_OCl introduz um novo modelo de comunicação, que serve de suporte a muitas das funcionalidades necessárias à computação orientada ao recurso, contribuindo, desta forma, para o desenvolvimento de abstracções de alto-nível directamente relacionadas com a programação de aplicações paralelas.

5.1.1 Conceitos gerais

O modelo de comunicação do R_OCl assenta em três entidades fundamentais, contextos de comunicação/computação, recursos e tampões de memória, e num serviço de directório de baixo-nível.

Um contexto é uma entidade que detém um ou mais portos de comunicação de baixo--nível, para envio e recepção de mensagens, comportando-se como um macrorrepositório de mensagens.

O recurso é uma abstracção usada para a modelação de entidades comunicantes e de entidades computacionais. Os recursos são tornados acessíveis mediante uma operação de registo num serviço de directório global e distribuído. Todo o recurso é associado a um contexto preexistente e possui um identificador único, no âmbito do *cluster*, atribuído pelo directório. O R_OCl não especifica quaisquer propriedades concretas de recursos, nem sequer limita a definição dessas propriedades. Os recursos são realizações de abstracções arbitrárias do nível aplicacional, cuja diferenciação resulta de uma lista de atributos especificada na sua criação.

Um atributo é um par $\langle nome, valor \rangle$, onde *nome* é uma cadeia de caracteres e *valor* é uma sequência de bytes. Um recurso R com n atributos é definido pela expressão $R = \{\langle nome_1, valor_1 \rangle, ..., \langle nome_n, valor_n \rangle\}$.

Para minimizar o número de operações de alocação e registo de memória, o modelo inclui um sistema de gestão de tampões de memória. Os tampões são zonas de memória registada usadas para manter as mensagens e, assim, permitir a comunicação sem cópias de memória.

Os identificadores globais dos recursos são usados para determinar a origem e o destino das mensagens. A identidade de um recurso poderá ser obtida a partir de uma pesquisa no directório, com base numa lista de atributos específicos desse recurso. Convém ainda salientar que, ao nível do directório, apenas os recursos são visíveis, uma vez que os contextos são virtualizados em recursos de sistema e os tampões são geridos no âmbito dos contextos.

5.1.2 Interface básico

A utilização de bibliotecas de comunicação de baixo-nível, tais como o GM e o VIA, não garante, automaticamente, comunicação livre de cópias; a camada de abstracção de alto-nível tem que definir um interface apropriado, que preserve as qualidades presentes nos sistemas de baixo-nível. O SOVIA [Kim 01] e o PVM sobre VIA [Espenica 02], por exemplo, usam VIA, o qual permite comunicação sem cópias, mas devido ao facto de os programadores continuarem limitados ao interface tradicional dos *sockets* ou do PVM,

esses sistemas são obrigados a copiar ou registar dados do utilizador (áreas de memória) antes do envio e não conseguem evitar uma cópia obrigatória na recepção. No caso do R_OCl, é importante realçar que as primitivas destinadas à manipulação de tampões e à comunicação propriamente dita foram desenhadas por forma a garantir um mecanismo de passagem de mensagens sem cópias de memória.

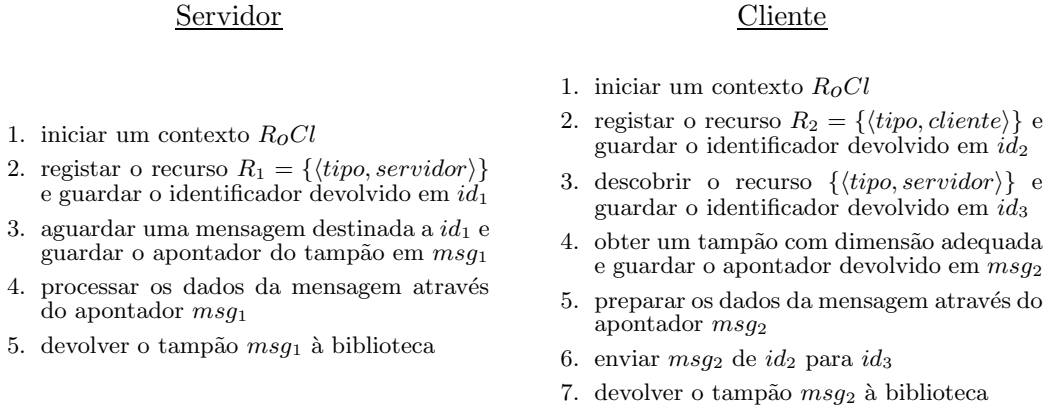


Figura 5.1: Um exemplo básico de comunicação entre recursos.

Para ilustrar o interface básico do R_OCl bem como os conceitos gerias acima apresentados, a figura 5.1 apresenta os passos necessários para o funcionamento de um interacção básica entre um cliente e um servidor, de acordo com o modelo de comunicação do R_OCl.

O conjunto básico de primitivas que os programadores podem usar para explorar o R_OCl é apresentado na tabela 5.1. Tomando como referência o exemplo da figura 5.1, a primitiva `rocl_init` corresponde aos passos *1*, `rocl_register` aos passos *2*, `rocl_query` e `rocl_bfget` aos passos *3* e *4* do cliente, respectivamente, `rocl_send` ao passo *6* do cliente, `rocl_recv` ao passo *3* do servidor e `rocl_bfret` aos passos *5* do servidor e *7* do cliente.

5.2 Serviço de directório

O modelo tem em vista a criação de um sistema totalmente dinâmico, em que as entidades comunicantes podem ser criadas e destruídas a qualquer momento durante a execução de uma aplicação, recorrendo à funcionalidade de um serviço de directório distribuído global. A importância de um directório foi já realçada em [Beck 99] e [Fitzgerald 97].

O serviço de directório foi projectado como um sistema distribuído e autónomo, que oferece acesso eficiente e escalável (ver secção 8.1) à sua base de recursos registados. Este serviço facilita o desenvolvimento de plataformas e aplicações de computação distribuída mais flexíveis.

Tabela 5.1: Primitivas R_OCl básicas.

Contextos
`int rocl_init(bool bridge)` `rocl_exit()`
Recursos
`int rocl_register(const rocl_attrl_t *attrs, const rocl_acl_t *acl, bool unique)` `rocl_delete(int gid)`
Tampões

`void * rocl_bfget(int len)`
`rocl_bfret(void *ptr)`
`rocl_bftoret(void *ptr)`
`int rocl_bfstat(const void *ptr, int timeout)`

Pesquisa
`int rocl_query(int gid, rocl_attrl_t *attrs, enum rocl_scope scope)`
Comunicação
`rocl_send(int ogid, int dgid, int tag, const void *ptr, int len)` `int rocl_recv(int dgid, int ogid, int tag, void **ptr, int *aogid, int *atag, int *alen, int timeout)`

5.2.1 Atributos

Um recurso é definido/registado através da especificação de uma lista de atributos. As primitivas usadas para manipular listas de atributos de recursos são apresentadas na tabela 5.2.

Tabela 5.2: Primitivas R_OCl para manipulação de listas de atributos.

```
rocl_attrl_t * rocl_new_attrl(int max_len)
int rocl_add_attr(rocl_attrl_t *attrs, const char *name, const void *val,
                  int len)
int rocl_get_attr(const rocl_attrl_t *attrs, const char *name,
                  void **value, int *len)
int rocl_nget_attr(const rocl_attrl_t *attrs, int lpos, char **name,
                   void **value, int *len)
rocl_destroy_attrl(rocl_attrl_t *attrs)
```

As listas de atributos são usadas tanto no registo como na pesquisa de recursos. No registo de um recurso, todos os atributos têm de ser completamente especificados, isto é, cada atributo tem de ter um nome e um valor definidos. Para efeitos de pesquisa, alguns

atributos podem ser parcialmente definidos, isto é, podem ser indicados apenas os seus nomes, por forma a indicar à biblioteca que informação deverá obter a partir do directório.

Para evitar cópias de memória supérfluas, as listas de atributos são armazenadas em zonas de memória contígua, para efeito de envio a um servidor (ver secção 5.2.2). A própria lista de atributos é usada como mensagem (pacote) de pedido ou resposta, obrigando à reserva de algum espaço à cabeça da lista, para que possa ser incluída informação de controlo.

5.2.2 Operação local

Os recursos são registados no servidor de directório local a cada nó do *cluster*, sendo produzidos identificadores globais a partir de um número de série, atribuído ao servidor na fase de arranque, e num contador local, por forma a evitar comunicação entre os diferentes nós. A base de dados de recursos locais é mantida na memória principal do nó, sendo usadas técnicas de acesso directo para acelerar as operações de pesquisa.

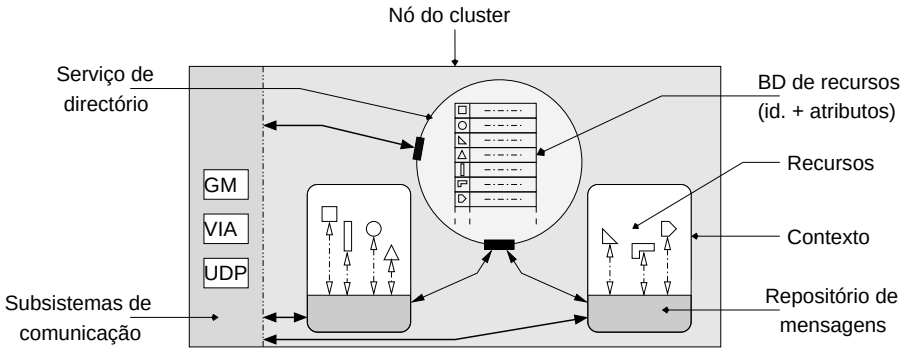


Figura 5.2: Registo de recursos locais.

A figura 5.2 apresenta o esquema de registo de recursos ao nível de um nó do *cluster*. A variedade de recursos é ilustrada através de diferentes formas geométricas. As identificações e os atributos dos recursos criados ao nível de dois contextos encontram-se registados na base de dados mantida pelo servidor de directório local. A comunicação entre as aplicações (clientes) e o respectivo servidor local é feita através de mecanismos de comunicação intra/interprocesso (IPC), enquanto que os subsistemas de comunicação são usados para a troca de mensagens entre recursos e para a comunicação entre servidores no âmbito de uma pesquisa global (ver secção 5.2.3).

Os atributos associados a cada recurso, e registados no directório, são inalteráveis, estando apenas disponível, para o programador, uma primitiva destinada à eliminação do registo de um dado recurso. Esta abordagem permite implementar, de forma mais eficiente, estruturas intermédias para armazenamento da informação mais requisitada.

Todos os recursos registados no âmbito de um dado contexto R_OCl são automaticamente eliminados pelo servidor de directório, quando este detecta que o contexto não se encontra activo. O mecanismo de detecção baseia-se em mensagens de presença que o contexto envia periodicamente ao servidor de directório local.

Uma interrogação recebida por um servidor local corresponde a um pacote de pedido que poderá incluir: o identificador do recurso, alguns atributos completamente especificados (com nomes e valores válidos) e alguns atributos parcialmente especificados (com valores *NULL*, mas tamanhos diferentes de zero associados a esses valores). Se o identificador do recurso está presente, o mecanismo de procura é trivial. No caso contrário, os atributos completamente especificados serão usados para produzir índices que facilitem o acesso directo à informação do recurso. Depois de descoberto o recurso pretendido, todos os atributos parcialmente especificados são examinados e cada um deles será completado, desde que um atributo com o mesmo nome tenha previamente sido associado ao recurso em causa. Se não existirem quaisquer atributos parcialmente especificados, então os nomes e respectivos valores de todos os atributos previamente associados ao recurso serão copiados.

5.2.3 Operação global

Quando o servidor local não é capaz de responder a uma interrogação em particular, é desencadeado um processo de procura global. Numa procura global, todos os servidores de directório em execução no *cluster*, um em cada nó, recebem o pedido, mas apenas responde aquele onde a pesquisa tiver sucesso. A figura 5.3 ilustra o processo de pesquisa global, tendo por base uma interrogação que, depois de processada ao nível do servidor local, foi enviada a dois servidores remotos.

O mecanismo de procura global baseia-se na difusão UDP, explorando o facto de a totalidade dos nós de um *cluster* estarem, habitualmente, conectados por uma rede Fast Ethernet, independentemente de poderem existir partições ao nível de tecnologias de comunicação de elevado desempenho. Esta abordagem tira partido do suporte nativo à difusão, tanto ao nível protocolar como ao nível do hardware. Além disso, num *cluster* dotado de tecnologias de comunicação de elevado desempenho, a rede Fast Ethernet apresenta-se como o meio de comunicação secundário, não sendo usado nas aplicações. Deste modo, a sua utilização para suporte ao funcionamento do serviço de directório, por um lado, rentabiliza um meio de comunicação que, normalmente, só serve para operações de configuração e gestão dos nós do *cluster* e, por outro lado, liberta as tecnologias de comunicação de elevado desempenho para a interoperação de componentes de aplicações.

Nos casos em que as operações de procura global podem ser limitadas a um único *subcluster*, os pedidos poderão ser entregues através da combinação de difusão UDP e árvores de dispersão para a tecnologia de comunicação desse *subcluster*. O funcionamento geral

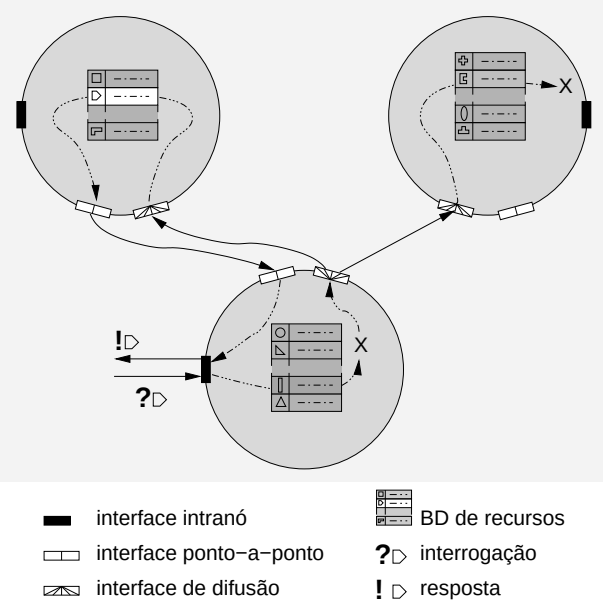


Figura 5.3: Mecanismo de pesquisa global.

será o seguinte: *1)* os servidores locais anunciam periodicamente a sua presença através de difusão UDP; *2)* cada servidor mantém uma lista dos restantes servidores activos; *3)* as árvores de dispersão são usadas para alcançar todos os servidores activos. A utilização de árvores de dispersão deve-se ao facto de as tecnologias de comunicação de elevado desempenho mais comuns não suportarem a difusão, quer seja por motivo do hardware (como é o caso da Myrinet) ou por motivo do protocolo de comunicação (como é o caso do VIA, devido à orientação à conexão).

5.2.4 Pesquisas com múltiplas respostas

As interrogações que não especificam o identificador de um recurso poderão resultar na devolução de múltiplas respostas, provenientes de um ou mais servidores de directório. De facto, recursos distintos poderão possuir atributos com nomes e valores idênticos e, portanto, uma pesquisa baseada em atributos comuns a diferentes recursos poderá devolver vários resultados.

O R_OCl fornece primitivas dedicadas para o manuseamento de múltiplas respostas, respeitantes a uma única operação de pesquisa (ver tabela 5.3). Este interface estende a primitiva básica `rocl_query` anteriormente apresentada, possibilitando a obtenção dos resultados de uma pesquisa a partir do servidor local, um de cada vez, tal como se se tratasse de uma sequência de pesquisas simples independentes.

Tabela 5.3: Primitivas R_OCl para suporte a pesquisas com múltiplas respostas.

```
rocl_handler_t * rocl_query_start(rocl_attrl_t *attrs,
                                  enum rocl_scope scope)
int rocl_query_next(rocl_handler_t *handler, rocl_attrl_t *attrs)
int rocl_query_stop(rocl_handler_t *handler)
```

O mecanismo de suporte à obtenção de múltiplas respostas é da responsabilidade de cada servidor local, o qual mantém informação de controlo/estado, por cada operação de pesquisa em curso, de modo a poder decidir entre as seguintes alternativas: procurar uma resposta na base de dados local, difundir a interrogação, armazenar respostas devolvidas por servidores remotos, procurar o próximo resultado na base de dados local, devolver uma resposta obtida a partir de um servidor remoto ou solicitar o próximo resultado a um servidor remoto.

Em resposta a um pedido recebido por via de uma determinada operação de difusão, cada servidor remoto devolve um único resultado – o primeiro que satisfizer os critérios de pesquisa – acompanhado de um índice relativo à posição na sua base de dados. O servidor local armazena as várias respostas recebidas (uma, no máximo, por cada servidor remoto) e, sempre que uma dessas respostas é devolvida à aplicação que desencadeou a procura, contacta individualmente um servidor específico (o responsável pela resposta devolvida), com o intuito de receber mais uma resposta, indicando-lhe o índice de posição anteriormente recebido. Este índice permite que o servidor remoto reinicie a operação de pesquisa a partir do local onde anteriormente tinha encontrada uma resposta.

5.3 Troca de mensagens inter-recurso

As aplicações R_OCl endereçam mensagens a recursos, cuja identificação pode ser previamente obtida através do serviço de directório. A passagem de mensagens inter-recurso levanta, essencialmente, dois problemas: o endereçamento e o despacho de mensagens. No primeiro caso, a dificuldade reside no facto de não existir uma relação directa entre recursos e endereços de portos dos subsistemas de comunicação. No segundo caso, é necessário ter em conta que, por um lado, os recursos são animados por fios-de-execução que concorrem pelo acesso a mensagens e, por outro lado, têm de ser multiplexados vários portos de comunicação associados a distintas tecnologias.

5.3.1 Endereçamento de mensagens

Os contextos são as entidades que os subsistemas de comunicação reconhecem como destino válido para as mensagens, pelo facto de possuírem portos de comunicação no âmbito de cada um desses subsistemas, o que torna necessário o estabelecimento da correspondência entre recursos e contextos.

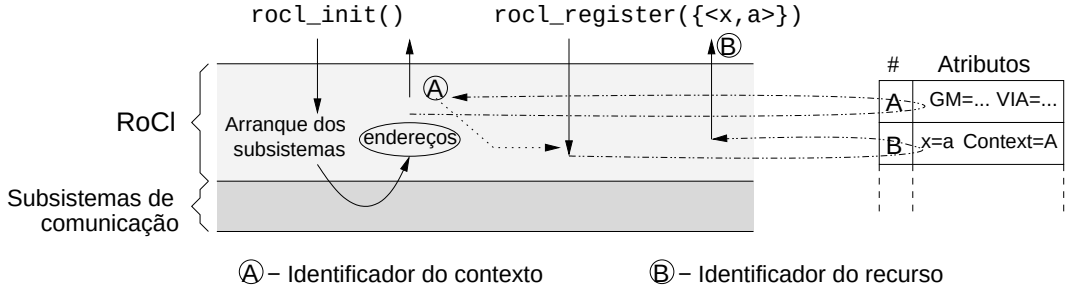


Figura 5.4: Correspondência entre recursos e contextos.

A correspondência entre recursos e contextos é efectuada conforme representado na figura 5.4. Os contextos, registados automaticamente no arranque da biblioteca, são tratados como recursos de sistema em que os endereços de portos dos subsistemas de comunicação são definidos como seus atributos. No registo de um recurso é usado implicitamente como atributo o identificador global de um contexto. Assim, todos os recursos são etiquetados com o identificador do contexto em que estão inseridos.

Esta abordagem é aparentemente ineficiente, dado que o envio de uma simples mensagem requer três passos: *1)* obter o contexto associado ao recurso através de uma consulta ao directório, *2)* obter os endereços de comunicação de baixo-nível associados ao contexto e *3)* enviar a mensagem com base num dos endereços obtidos. Os dois primeiros passos exigem a troca de alguns pacotes (pedidos/respostas ao/do serviço de directório), o que faz com que o tempo de resposta associado ao envio de uma mensagem assuma valores inaceitáveis. Para contornar esta limitação, a biblioteca usa duas estruturas de armazenamento intermédio, que guardam as correspondências entre identificadores de recursos e identificadores de contextos e entre identificadores de contextos e endereços de portos dos subsistemas de comunicação usadas mais recentemente.

5.3.2 Despacho de mensagens

Os recursos R_OCl são animados por fios-de-execução, permitindo desta forma o acesso concorrente/paralelo aos mecanismos de comunicação. Por sua vez, a recepção de mensagens numa aplicação com múltiplos fios-de-execução é totalmente assíncrona, o que significa

que, no momento da chegada de uma determinada mensagem, poderá não existir uma entidade a aguardar essa mensagem em particular.

O R_OCl é um sistema de comunicação não-orientado à conexão que usa um mecanismo de despacho baseado em fios-de-execução e filas de mensagens. Os fios-de-execução, um por cada subsistema de comunicação suportado, aguardam a chegada de mensagens, recorrendo à sondagem e a mecanismos de tratamento de interrupções, armazenando-as numa fila de recepção. Os recursos extraem mensagens da fila, com base nos seguintes critérios de selecção: destino, origem e etiqueta da mensagem. A primitiva de recepção possibilita recepções temporizadas com base no parâmetro `timeout` (ver tabela 5.1): um valor nulo indica que, no caso de não existir uma mensagem que satisfaça os critérios, a primitiva devolve imediatamente um código de erro; um valor negativo indica que a primitiva aguarda o tempo necessário até à chegada de uma mensagem; outro qualquer valor servirá para especificar o período de tempo que a primitiva poderá aguardar por uma mensagem, devolvendo um código de erro no fim desse período.

A primitiva de envio, uma vez preenchidos os parâmetros, entrega uma mensagem directamente a um subsistema de comunicação, o qual, com base no hardware do interface de comunicação, se encarrega do envio efectivo da mensagem, ficando a aplicação de imediato livre para prosseguir a sua execução.

Durante o período de tempo em que uma dada mensagem é processada pelo subsistema de comunicação, o envio de uma outra qualquer mensagem bloqueará a aplicação. No sentido de contornar esta limitação, o R_OCl também mantém uma fila de envio, que apenas é usada quando o subsistema de comunicação está indisponível, garantindo-se assim que a primitiva de envio retorna sempre de imediato à aplicação. Os fios-de-execução do sistema, para além de tratarem das mensagens recebidas, também monitorizam a fila de envio, por forma a processar envios pendentes.

É importante referir que as filas de envio e recepção apenas manipulam registos descritores de mensagens, que contêm um apontador para os dados da mensagem, evitando desta forma cópias de memória supérfluas.

Detalhes da multiplexagem

A generalidade dos sistemas de comunicação de baixo-nível oferecem mecanismos apenas para comunicação entre nós de um *cluster*, não contemplando as entidades variadas do mundo aplicacional como origem/destino das mensagens. No caso do GM, por exemplo, cada interface de comunicação suporta apenas oito portos de comunicação, o que dificulta o desenvolvimento de aplicações em que múltiplas entidades lógicas necessitam de trocar mensagens. Em muitos subsistemas de comunicação nem sequer é suportado o acesso

concorrente/paralelo de múltiplos fios-de-execução a um ponto de comunicação.

O mecanismo de despacho do R_OCl permite que um único porto de comunicação de um dado subsistema seja multiplexado por múltiplas entidades concorrentes. No que diz respeito ao envio, mesmo quando as bibliotecas de baixo-nível não suportam acesso concorrente por parte de vários fios-de-execução, a tarefa do R_OCl resume-se à coordenação das evocações de primitivas de envio dos vários fios-de-execução. No que concerne à recepção, devido ao carácter assíncrono da chegada de mensagens a uma aplicação, os fios-de-execução do sistema R_OCl que se encarregam do despacho de mensagens deverão recorrer à sondagem ou aos mecanismos de interrupção dos subsistemas, mediante a evocação de primitivas não-bloqueantes ou bloqueantes, respectivamente, disponibilizadas pelas bibliotecas de comunicação de baixo-nível.

Como já foi referido, a recepção de mensagens é feita por intermédio de uma fila de recepção. O mecanismo de despacho insere na fila as mensagens e, no caso de existir algum fio-de-execução bloqueado, à espera de uma mensagem em particular, se uma dada mensagem obedecer aos critérios indicados na primitiva de recepção, o mecanismo de despacho notificará esse fio-de-execução. Note-se que a partilha de um porto de comunicação, por parte de vários fios-de-execução que animam recursos, obriga à existência de um mecanismo específico de multiplexagem. Com efeito, por um lado, não é possível que vários fios-de-execução fiquem bloqueados, aguardando mensagens, recorrendo apenas às primitivas de uma biblioteca de comunicação de baixo-nível e, por outro lado, nenhum sistema de escalonamento poderá garantir que, no momento da chegada de uma mensagem, se encontra activo o fio-de-execução que anima o recurso ao qual a mensagem é efectivamente endereçada.

Impacto da multiplexagem

As operações de sondagem, num ambiente com múltiplos fios-de-execução, podem ser altamente penalizadoras para o tempo de execução de uma aplicação, uma vez que o mecanismo de despacho do R_OCl, materializado em alguns fios-de-execução, concorre, no acesso ao CPU, com os fios-de-execução da aplicação. No entanto, em algumas aplicações, a redução da frequência de sondagem para valores muito baixos pode influenciar drasticamente o desempenho global.

A utilização de primitivas bloqueantes das bibliotecas de comunicação de baixo-nível, que exploram mecanismos de tratamento de interrupções, também podem acarretar alguma degradação de desempenho. Com efeito, esta estratégia obriga a comutações de contexto, o que aumenta o tempo necessário para efectivamente tratar a chegada de uma mensagem, uma vez que terá que ser escalonado um fio-de-execução do sistema de despacho e, posteriormente, escalonado um outro da aplicação.

Neste contexto, houve a preocupação de minimizar o impacto do sistema de despacho de mensagens. Em primeiro lugar, incluíram-se mecanismos para comutar entre a sondagem e a recepção bloqueante; o funcionamento geral é baseado no modo bloqueante, mas, quando todos os fios-de-execução da aplicação se encontram à espera de mensagens ou quando a frequência da chegada de mensagens é muito elevada, o sistema passa a usar a sondagem. Em segundo lugar, e de forma a permitir que os fios-de-execução da aplicação também cooperem no processo de despacho de mensagens, o R_OCl disponibiliza a primitiva `rocl_dispatch` (ver tabela 5.4), que pode ser usada pelo programador, ao longo do código da aplicação, com o intuito de despoletar uma acção de sondagem. A evocação de quaisquer outras primitivas R_OCl, por parte dos fios-de-execução de uma aplicação, também produz o mesmo efeito (em acréscimo às operações específicas subjacentes a cada primitiva).

Tabela 5.4: Primitiva R_OCl para apoio ao despacho.

`rocl_dispatch()`

No sentido de minimizar o impacto do despacho de mensagens nas aplicações, alguns autores ([Langendoen 96, Hansen 00, Danjean 00]) propuseram a inclusão de mecanismos de sondagem no próprio escalonador de fios-de-execução. Apesar dos níveis elevados de desempenho que esta abordagem permite alcançar, tem como inconveniente a necessidade de alterar, ou desenhar de raíz, o escalonador, o que inviabiliza o recurso indiscriminado a bibliotecas de rotinas preexistentes, baseadas em fios-de-execução POSIX.

5.3.3 Escrita e leitura remotas

Em geral, a passagem de mensagens envolve dois intervenientes activos: o emissor e o receptor. Ao receptor cabe a tarefa de, por cada mensagem enviada, desencadear a execução de uma operação para consumo da mensagem. Em abordagens do género da adoptada no modelo de mensagens activas [von Eicken 92] não há lugar à execução explícita de uma operação de recepção no destino, uma vez que uma entidade de sistema se encarrega do processamento associado à entrega da mensagem.

Algumas tecnologias de comunicação mais recentes introduzem operações de leitura e escrita remota, em que o emissor é o único interveniente activo, vulgarmente denominadas de operações RDMA (Acesso Directo a Memória Remota).

No caso do R_OCl, a passagem de mensagens convencional implica a activação de um fio-de-execução do sistema de despacho no destino, o qual, por sua vez, irá proceder à activação do fio-de-execução que anima o recurso associado ao destino da mensagem (ver figura 5.5(b)). Com a integração de operações RDMA, o sistema R_OCl passa a dispor

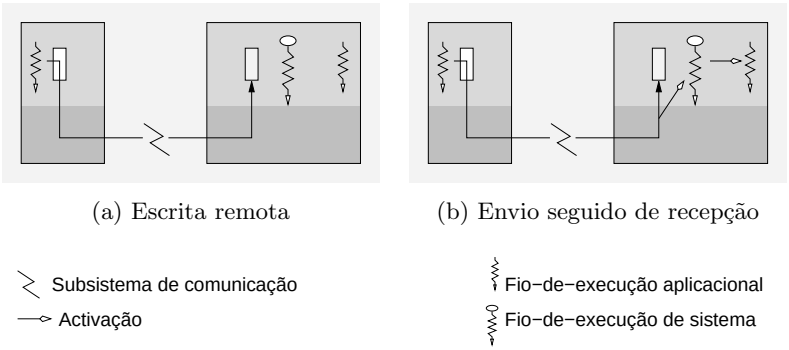


Figura 5.5: Escrita remota *vs* envio-recepção.

de primitivas de comunicação com tempos de resposta muito próximos do hardware de interligação, já que não é necessária a activação de qualquer fio-de-execução no destino (ver figura 5.5(a)).

As operações RDMA no R_OCl usam directamente os conceitos de recurso e tampão de memória, sem introduzir novas abstracções. As primitivas desenhadas para o efeito (ver tabela 5.5) permitem: associar a um recurso um determinado tampão, tornando-o acessível remotamente, isto é, permitindo a escrita e leitura remotas (`rocl_bfshare`), movimentar um fragmento de um dado tampão local para uma determinada zona do tampão remoto associado ao recurso indicado como destino (`rocl_put`) e movimentar um fragmento do tampão remoto associado ao recurso indicado como origem para uma determinada zona de um dado tampão local (`rocl_get`).

Tabela 5.5: Primitivas R_OCl para escrita/leitura remota.

```
rocl_bfshare(int ogid, const void *ptr)
rocl_put(int ogid, int dgid, const void *ptr, int loffset, int roffset,
         int length)
rocl_get(int dgid, int ogid, int roffset, void *ptr, int loffset,
         int length)
```

Saliente-se que a primitiva `rocl_bfshare` não tem por objectivo publicar um endereço de memória através do directório. Apenas terá como efeito, no contexto R_OCl onde foi evocada, a associação do endereço de um tampão ao identificador de um recurso local. Quando as primitivas `rocl_put` e `rocl_get` são evocadas, o identificador do recurso remoto indicado é usado pelo sistema R_OCl para contactar o contexto remoto correspondente e obter o endereço necessário à escrita/leitura remota. O sistema R_OCl também mantém uma estrutura de armazenamento intermédio, que associa identificadores de recursos a

endereços de tampões, para minimizar as trocas de informação entre contextos.

Esta abordagem introduz um nível de abstracção superior ao oferecido pelas bibliotecas de comunicação de baixo-nível, como é o caso do GM e do VIA. Com efeito, os modelos de comunicação destes subsistemas impõem o uso das convencionais primitivas de envio e recepção de mensagens para obter os endereços de memória necessários nas operações RDMA, isto é, antes de uma operação de leitura ou escrita remota, terá lugar o envio convencional de uma mensagem, para dar a conhecer o endereço de memória remoto. No modelo do R_OCl, o directório e a informação de controlo trocada entre contextos, de forma transparente às aplicações, permitem esconder a complexidade associada às operações de escrita e leitura remotas.

5.3.4 Controlo de acesso

No modelo de comunicação, um recurso poderá endereçar mensagens a qualquer outro recurso registado no directório, independentemente do nó do *cluster*, da aplicação ou mesmo do utilizador. A estratégia de identificação global coloca num único plano os vários recursos dispersos pelos nós de um *cluster*, independentemente das circunstâncias da criação de cada recurso. Apesar da flexibilidade que esta particularidade do R_OCl oferece, num ambiente multiprograma e multi-utilizador haverá situações onde se torna necessário impor algumas limitações à interacção entre recursos. Por esta razão, é necessário pôr à disposição dos programadores mecanismos de controlo de acesso, de forma a impedir que a um dado recurso sejam enviadas mensagens indesejáveis.

O mecanismo de controlo de acesso baseia-se no facto de que o envio de mensagens requer sempre uma consulta ao directório, quer seja para obter o identificador de uma entidade da qual se conhecem alguns atributos, quer seja para obter endereços dos pontos de acesso aos subsistemas de comunicação. Se o serviço de directório não responder a pedidos efectuados pela aplicação de um dado utilizador, relativamente a recursos a que lhe está vedado o acesso, a comunicação não poderá ter lugar, visto que, ou a aplicação não consegue obter o identificador do recurso destino, ou a biblioteca R_OCl não consegue obter os endereços associados ao identificador desse recurso.

O mecanismo de controlo de acesso obriga à associação de informação específica a cada recurso R_OCl, para além dos normais atributos usados para caracterizar o recurso. Assim, a primitiva de registo – `rocl_register` – permite a especificação de uma lista de controlo de acesso, através do parâmetro `acl` (ver tabela 5.1), que designa os utilizadores, ou grupos de utilizadores, que podem interactuar com o recurso. Esta lista é armazenada como um conjunto de atributos de sistema, inacessíveis através do interface de programação.

As interrogações enviadas pelas primitivas `rocl_query` e `rocl_query_start` ao servidor

local incluem as identificações do utilizador e do seu grupo, acrescentadas, automaticamente, pela biblioteca R_OCl. Deste modo, o serviço de directório poderá decidir se deve ou não responder a uma dada interrogação, bastando verificar se o utilizador responsável pelo pedido está contemplado na lista de controlo de acesso associada ao recurso. Saliente-se que a primitiva `rocl_query` é usada tanto pelo programador, para obter o identificador de um recurso, como pelo próprio sistema R_OCl, para obter endereços de comunicação para um dado identificador.

5.4 Difusão selectiva

Tradicionalmente, as bibliotecas de comunicação de baixo-nível são usadas para a comunicação ponto-a-ponto. Em alguns casos, estão disponíveis funcionalidades de difusão, mas apenas ao nível do nó, isto é, oferece-se a possibilidade de fazer chegar uma dada mensagem a todos os nós do *cluster*.

Na programação de aplicações paralelas, é vulgar a definição de grupos de entidades, aos quais podem ser directamente endereçadas mensagens, com o intuito de suportar difusão selectiva. Esta funcionalidade é habitualmente implementada à custa da comunicação ponto-a-ponto oferecida pelas bibliotecas de comunicação de baixo-nível, utilizando-se técnicas que permitem explorar particularidades do hardware de comunicação normalmente usado em *clusters* [Huse 99].

5.4.1 Relacionamento de recursos

Até ao momento, os recursos R_OCl foram apresentados despojados de qualquer relação ou organização entre si, a não ser o facto de se encontrarem confinados a contextos, apenas com a finalidade de partilharem infra-estruturas de comunicação. Com a finalidade de suportar operações de comunicação envolvendo mais que duas entidades, o modelo de comunicação do R_OCl inclui funcionalidades para o relacionamento de recursos.

Tabela 5.6: Primitiva R_OCl para relacionamento de recursos.

```
int rocl_set_rel(int ogid, int dgid, enum rocl_rel rel)
```

A primitiva `rocl_set_rel`, apresentada na tabela 5.6, permite organizar quaisquer recursos, mediante o estabelecimento de relações de afinidade, sendo contempladas duas formas de relacionamento:

- filiação – um determinado recurso torna-se descendente de um outro e as mensa-

gens endereçadas ao ascendente também lhe são entregues (representado pela seta contínua e unidireccional do esquema da figura 5.6(a));

- emparelhamento – um determinado recurso torna-se par de um outro e as mensagens endereçadas a qualquer um deles são entregues a ambos (representado pela seta contínua e bidireccional do esquema da figura 5.6(b)).

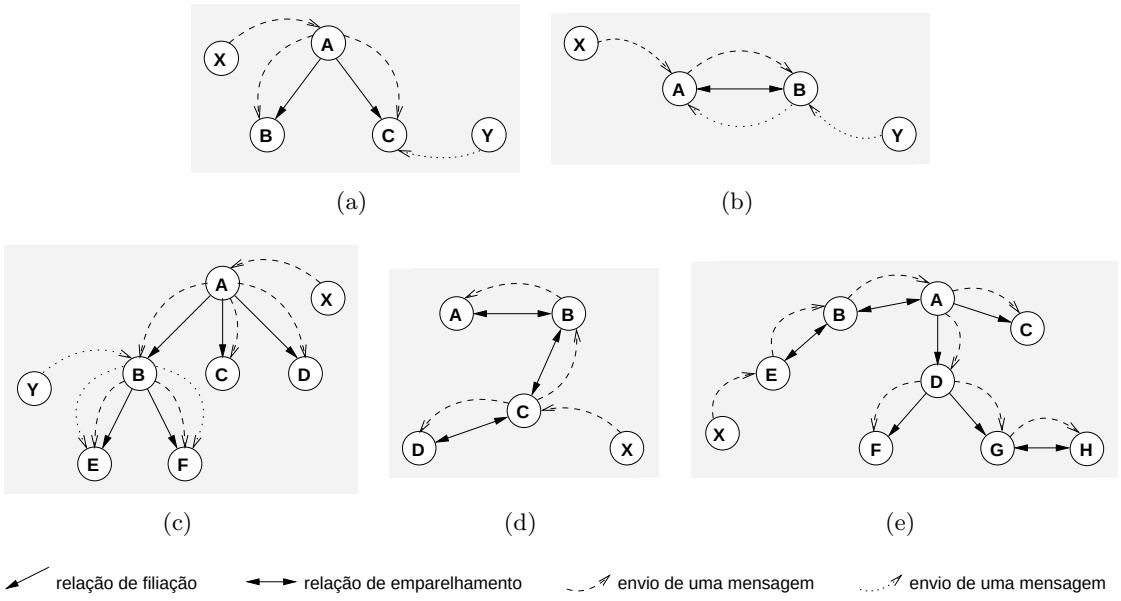


Figura 5.6: Exemplos de relacionamento de recursos e sequente entrega de mensagens.

Os esquemas 5.6(c) e 5.6(d) apresentam o encadeamento de relações de filiação e de emparelhamento, respectivamente, enquanto o esquema 5.6(e) exemplifica a combinação dos dois tipos de relações. X e Y representam recursos que endereçam mensagens a outros recursos com relações estabelecidas, enquanto que as setas curvas a tracejado ou ponteado representam a entrega das respectivas mensagens.

O relacionamento de recursos permite suportar facilidades de comunicação multiponto, sem que sejam introduzidas novas abstracções; o recurso continua a ser a única entidade endereçável, no que diz respeito à troca de mensagens, mas um único recurso pode, eventualmente, representar múltiplos destinos. Note-se que, na prática, estão em causa duas formas de criação de grupos de recursos – uma com base no conhecimento do recurso que representa o grupo (o ascendente, numa relação de filiação) e outra com base no conhecimento apenas de um dos membros do grupo (o recurso par, numa relação de emparelhamento).

Esta abordagem não explora características especiais ao nível dos subsistemas de comunicação ou do hardware de comunicação usados no *cluster*. Na verdade, esta abordagem é

do mesmo tipo das utilizadas nos sistemas ALMI [Pendarakis 01] e SCRIBE [Castro 02], os quais constituem infra-estruturas de difusão selectiva de nível aplicacional.

5.4.2 Pressupostos da abordagem

O esquema de suporte ao relacionamento de recursos levanta algumas questões, cuja resolução é fundamental.

Unicidade

O directório, por omissão, não garante a unicidade dos recursos registados, sendo possível registar diferentes recursos que têm exactamente os mesmos atributos. Note-se que preceder a operação de registo de uma simples operação de consulta, para verificar se um recurso idêntico já existe, não soluciona o problema, visto que as duas operações, em conjunto, não se comportam como uma operação atómica.

A garantia de unicidade de um dado recurso é fundamental para o suporte às relações de filiação. Neste tipo de relações, vários componentes aplicacionais tentarão filiar recursos sob um recurso específico, o qual representará o grupo. Este recurso deverá ser criado antes de qualquer operação de filiação. No entanto, numa aplicação paralela onde vários componentes cooperam, torna-se difícil decidir quem deve criar este recurso.

Para colmatar esta dificuldade, a primitiva de registo `rocl_register` inclui o parâmetro `unique` (ver tabela 5.1), que permite indicar se se admite a existência de recursos diferentes com atributos iguais. Quando a garantia de unicidade é solicitada, se já existir um recurso com atributos coincidentes com os especificados na primitiva `rocl_register`, será devolvido o identificador global do recurso já existente.

Dado que o sistema de directório do R_OCl é totalmente distribuído, a implementação desta funcionalidade requer a interacção de todos os servidores dispersos pelo *cluster*. Assim, quando se pretende um registo com garantia de unicidade, o servidor responsável tentará obter a aprovação dos restantes, isto é, envia uma mensagem com os atributos do recurso em causa a cada um dos servidores remotos e aguarda a resposta de cada um deles. Se algum servidor responder com um identificador válido, então o recurso não é registado; se todos os servidores responderem com um identificador inválido, então o recurso é registado.

Gestão de relações

A criação/nomeação de grupos e a gestão dos seus membros são essenciais no suporte à comunicação colectiva. No R_OCl, a dificuldade surge na gestão de relações, ou seja, na gestão dos membros de um grupo, visto que o problema da nomeação é facilmente

ultrapassado devido à utilização do recurso como abstracção única e devido a que no registo de recursos se pode garantir unicidade.

No momento do estabelecimento de uma relação apenas é conhecido o recurso ascendente ou o recurso par – recurso alvo de uma operação de relacionamento. Deste modo, a primitiva `rocl_set_rel` incluirá o envio de uma mensagem ao recurso alvo (parâmetro destino da primitiva), o qual acabará por conhecer o conjunto de todos os recursos descendentes, no caso de relações de filiação, ou pares, no caso de relações de emparelhamento. Isto significa que todas as relações estabelecidas relativamente a um dado recurso são mantidas num único local – o contexto do recurso alvo. No entanto, dado que os potenciais recursos alvo podem encontrar-se dispersos pelo *cluster*, o R_OCl utilizará, na realidade, um mecanismo distribuído de gestão de relações.

As relações de um recurso são tratadas como atributos especiais, dado que podem ser adicionadas ou eliminadas em tempo de execução, ao contrário dos demais atributos que são estabelecidos no momento de registo e que não podem ser eliminados ou alterados. Assim, com a evocação da primitiva `rocl_set_rel(ogid, dgid, ...)`, numa relação de filiação, ao recurso ascendente é adicionado um atributo $\langle descendente, ogid \rangle$ e ao recurso descendente é adicionado um atributo $\langle ascendente, dgid \rangle$, enquanto que, numa relação de emparelhamento, ao recurso de origem (a partir de onde é evocada a primitiva) é adicionado um atributo $\langle par, dgid \rangle$ e ao recurso alvo é adicionado um atributo $\langle par, ogid \rangle$. Estes atributos, para os recursos de cada contexto R_OCl envolvidos em relações de filiação ou emparelhamento, são armazenados numa tabela gerida pelo sistema de despacho. Obviamente, um dado recurso poderá possuir vários atributos *descendente*, *ascendente* ou *par*, de acordo com o número de relações em que está envolvido.

Entrega de mensagens

O R_OCl desencadeia processos de difusão selectiva para aquelas mensagens endereçadas a recursos que possuam descendentes ou pares. A complexidade e a eficiência do sistema de entrega das cópias de uma determinada mensagem ao conjunto de recursos em causa depende da metodologia usada para gerir as relações estabelecidas entre recursos.

Numa relação de filiação, o recurso ascendente (que pode ser entendido como gestor do grupo), pelo facto de conhecer todos os descendentes, pode proceder ao envio de uma cópia da mensagem a cada um deles, implementando assim a difusão selectiva. Se um descendente for, por sua vez, ascendente de um outro conjunto de recursos, uma cópia da cópia recebida será enviada a cada um deles e assim sucessivamente. Esta abordagem é semelhante à seguida no DECK [Cassali 00], com a vantagem de, na realidade, existirem vários pontos centralizadores distribuídos pelos vários nós do *cluster*.

No caso do emparelhamento, o mecanismo de entrega corresponde ao simples reencaminhamento de mensagens.

É importante salientar que o mecanismo flexível de estabelecimento de relações não impede a criação de ciclos. Para evitar este problema, é definido um tempo de vida para as mensagens, que impede a permanência destas, no sistema de comunicação, por tempo indeterminado, evitando a degradação de desempenho. No entanto, esta solução, por si só, não garante que uma dada mensagem não seja entregue mais que uma vez ao mesmo recurso. Consequentemente, no momento da recepção, é verificado se a mensagem é uma repetição com base em informação de controlo definida na origem.

5.4.3 Difusão selectiva optimizada

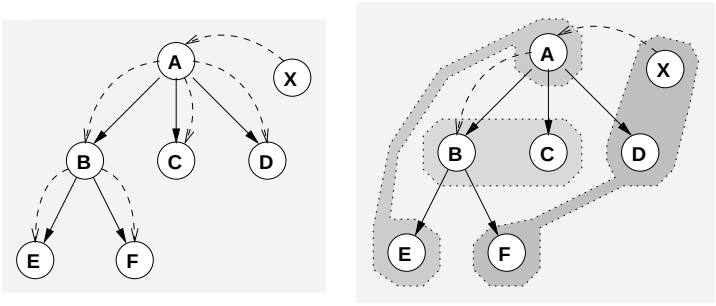
Num sistema de difusão selectiva procura-se sempre que o intervalo de tempo necessário para a entrega de uma mensagem a um dado conjunto de destinatários seja inferior ao tempo necessário para o envio individualizado e em sequência de uma cópia da mensagem a cada um dos recursos destino. No R_OCl, a existência de contextos, onde vários recursos partilham mecanismos de comunicação, abre caminho a optimizações específicas.

Difusão por contextos

A figura 5.7(a) apresenta um cenário de entrega encadeada de uma mensagem a vários recursos ligados por relações de filiação, a que correspondem seis operações de envio. Na figura 5.7(b), o mesmo cenário é repetido com a indicação a sombreado da distribuição dos recursos por contextos, verificando-se que a entrega da mensagem envolve apenas dois envios. Este exemplo permite realçar duas optimizações importantes:

- quando mais que um descendente directo pertencem ao mesmo contexto, como é o caso de B e C, deve ser enviada uma única mensagem;
- quando um descendente pertence a um contexto onde já existe uma cópia da mensagem, como é o caso de D, E e F, deve ser suprimido o envio.

Estas optimizações baseiam-se no facto de os recursos de um dado contexto partilharem uma única estrutura de armazenamento de mensagens. Assim, quando ocorrer a recepção da mensagem no contexto dos recursos B e C, por exemplo, o sistema produz uma outra cópia, por forma a que possam ser inseridas na fila de recepção duas mensagens, uma por cada recurso.

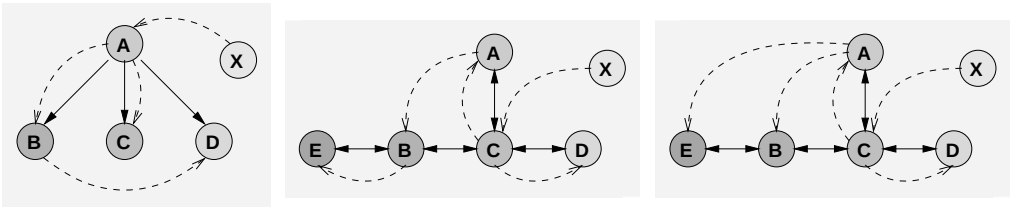


(a) (b)

Figura 5.7: Difusão por contextos.

Envios em paralelo

A difusão por contextos é compatível com a possibilidade de recorrer a árvores de dispersão, para garantir a redução dos tempos de entrega da totalidade das mensagens.



(a) (b) (c)

Figura 5.8: Delegação de responsabilidade de envio.

A figura 5.8(a) mostra uma forma alternativa de fazer chegar uma mensagem a cada um dos descendentes directos do recurso A; em vez de enviar uma mensagem a cada um dos descendentes, o recurso ascendente envia mensagens apenas a alguns destes, os quais, de forma coordenada, se encarregam de fazer chegar mensagens aos restantes. A figura 5.8(b) mostra uma situação análoga para os recursos pares do recurso C.

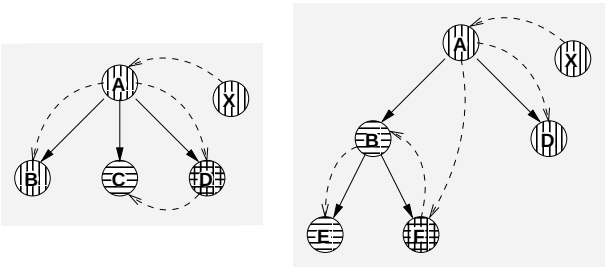
No R_OCl, a construção de árvores de dispersão, tomando um dado recurso como ponto de partida, está facilitada para os casos de envio de mensagens a recursos que distem apenas uma relação. De facto, para cada recurso, o R_OCl conhece todos aqueles directamente relacionados, por filiação ou emparelhamento, podendo a estratégia de entrega – definição de quem entrega a quem – ser estabelecida logo à partida.

A construção de árvores de dispersão que envolvam recursos a uma distância superior a uma relação (ver figura 5.8(c)) também é possível, trazendo, obviamente, vantagens do

ponto de vista do desempenho. No entanto, a informação necessária para o estabelecimento da estratégia de entrega é mais abrangente, obrigando o R_OCl a trocar alguma informação entre diferentes sistemas de despacho. Na figura 5.8(c), o sistema de despacho do contexto do recurso C, inicialmente, não tem conhecimento de E, tendo, portanto, que perguntar ao sistema de despacho do contexto de B qual a sua lista de relações.

Partições tecnológicas

O R_OCl contempla a possibilidade de troca de mensagens entre recursos sediados em contextos que não disponham de uma tecnologia de comunicação em comum, por via da implementação de um serviço de reencaminhamento, ao nível dos nós multi-interface (ver secção 6.6.1). Na difusão de mensagens, os serviços de reencaminhamento podem, em alguns casos, ser evitados, desde que o processo de construção das árvores de dispersão tome em consideração a multiplicidade de tecnologias de comunicação dos contextos dos vários recursos envolvidos.



(a) (b)

Figura 5.9: Difusão por tecnologias.

Na figura 5.9 são usados padrões com linhas verticais ou horizontais, para distinguir recursos com acesso a duas tecnologias de comunicação distintas. Relativamente à figura 5.9(a), apesar de o recurso A não dispor de meios para comunicar directamente com o recurso C, o que implicaria a utilização automática de um reencaminhador, por parte do sistema de comunicação do R_OCl, a escolha de D (com acesso a ambas as tecnologias), para delegação de responsabilidade de envio, resolve o problema. Note-se que a utilização automática de reencaminhadores corresponde a um passo suplementar na cadeia de envios, pois os reencaminhadores só serão destinos úteis no caso de um recurso alvo estar sediado num nó multi-interface.

A construção de uma árvore de dispersão que minimize o uso de reencaminhadores, isto é, que evite a utilização de contextos não compreendidos na lista de destinos naturais de uma mensagem, é facilitada pela consulta dos atributos normais dos contextos, os

quais podem ser armazenados nas tabelas usadas para gestão de relações. Os casos que envolvem recursos que distam mais que um relacionamento, como acontece na figura 5.9(b), obrigam, mais uma vez, à troca de informação entre sistemas de despacho. O recurso A apenas poderá concluir que não necessita de enviar uma mensagem directamente para B, operação essa que envolveria reencaminhamento, depois de saber que F é descendente de B e que F tem acesso à tecnologia de comunicação necessária para chegar a B sem reencaminhamento.

5.5 Operação em ambiente *multicluster*

O principal objectivo do R_OCl é a operação em ambientes baseados num único *cluster*, composto por vários *subclusters*, razão pela qual o UDP é usado para suportar o funcionamento global do serviço de directório, no pressuposto de que todos os nós do *cluster* estão interligados por Fast Ethernet. A necessidade de interligação de dois *clusters* geograficamente distantes levou à inclusão de funcionalidade para suporte à operação em ambiente *multicluster*.

5.5.1 Directório multinível

O primeiro obstáculo à operação em ambiente *multicluster* prende-se com a identificação global dos recursos e com as pesquisas globais.

O primeiro óbice pode ser ultrapassado reservando alguns bits no identificador do recurso, para armazenar a identificação do *cluster*. A escolha de um identificador único para cada *cluster* bem como a definição do número de bits necessários para o efeito, dado o seu caracter esporádico, são tarefas meramente administrativas, que têm lugar na fase de instalação da plataforma R_OCl.

No que concerne às pesquisas, uma vez que ao nível do *cluster* se recorre à difusão UDP, a qual não pode ser usada em ambientes *multicluster*, torna-se necessário organizar o serviço de directório por níveis. Muitas vezes, os nós de um *cluster* nem sequer podem ser alcançados a partir do exterior, por terem endereços IP inválidos.

Pesquisas *multicluster*

O directório é organizado em vários níveis, contemplando a utilização de um representante por *cluster*, instalado numa máquina com um endereço IP válido, para efeitos de interface com o exterior. Desta forma, os servidores de directório poderão redireccionar interrogações para *clusters* remotos, através do envio dos pedidos correspondentes aos

representantes desses *clusters*.

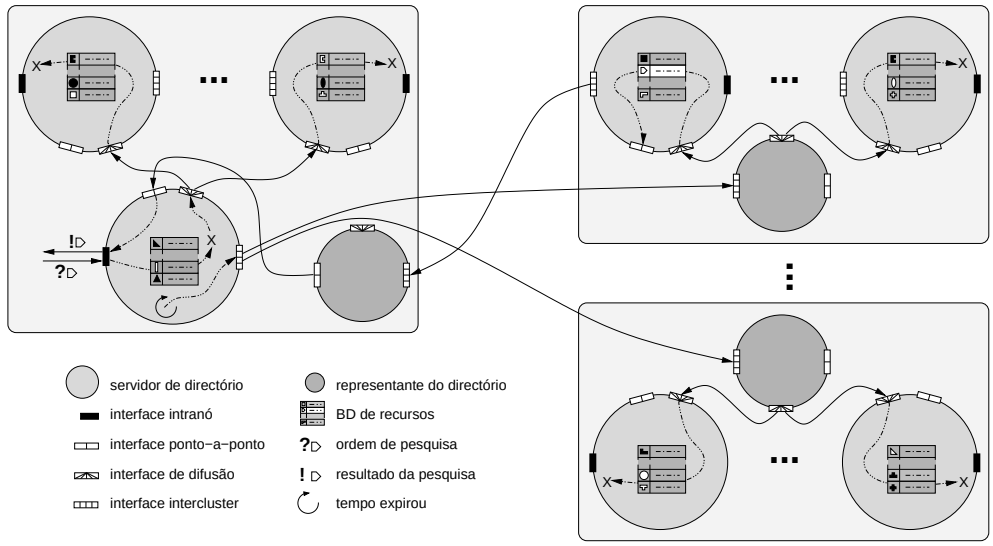


Figura 5.10: Mecanismo de pesquisa *multicluster*.

Num ambiente *multicluster* (ver figura 5.10), quando um servidor não consegue satisfazer localmente uma interrogação nem obtém respostas a um pedido difundido ao nível do *cluster* desencadeia um processo de pesquisa *multicluster*. São então enviados pedidos em sequência a cada um dos representantes de *clusters* remotos conhecidos, que se encarregarão de difundir o pedido nos respectivos *clusters*. O servidor que possuir uma resposta procederá à sua devolução, usando como intermediário o representante do *cluster* de onde partiu o pedido. Por sua vez, esse representante reenviará a resposta ao servidor que despoletou a pesquisa *multicluster*, que a entregará à aplicação que evocou a primitiva `rocl_query`.

No caso de uma aplicação cliente usar a primitiva `rocl_query_start`, para dar início a um processo de obtenção de múltiplas respostas, o servidor contactado começa por difundir, no âmbito do seu *cluster*, o pedido que lhe foi endereçado, de seguida procede ao envio desse mesmo pedido aos representantes de *clusters* remotos conhecidos e, por fim, dá início a uma pesquisa na base de dados local. As eventuais respostas, recebidas de forma assíncrona, são geridas da forma exposta na secção 5.2.4.

A pesquisa *multicluster* podia, em certa medida, recorrer a técnicas usadas pelas plataformas de construção de servidores *Web* baseados em *clusters*, como é o caso do LVS (Linux Virtual Server) [Zhang 00]. No entanto, dado que todos os intervenientes são aplicações desenvolvidas no âmbito do R_OCl, foi possível adoptar soluções menos complexas. Refira-se ainda que estratégias do género das usadas pelo OneIP [Damani 97] e pelo

NLB (Network Load Balancing) [Microsoft 00] permitiriam melhorar o desempenho, mas obrigariam à codificação de parte do serviço de directório do R_OCl ao nível do *kernel* do sistema operativo, limitando a portabilidade.

Âmbito das pesquisas

O envolvimento de servidores de directório de nós de *clusters* remotos em quaisquer operações de pesquisa poderá ser penalizador para muitas aplicações. De facto, ao programador deverá ser dada a possibilidade de, num dado momento, restringir o âmbito de uma pesquisa, dado o seu conhecimento sobre a forma como os recursos se encontram distribuídos. Com esse intuito, as primitivas `rocl_query` e `rocl_query_start` incluem o parâmetro `scope` (ver tabelas 5.1 e 5.3), o qual permite definir o âmbito de uma pesquisa, de acordo com os seguintes valores:

- `LOCAL` – apenas a base de dados do servidor local é usada, para procurar respostas;
- `SUBCLUSTER` – no caso de a base de dados local não possuir uma resposta ou tratando-se de uma pesquisa para obtenção de múltiplas respostas, o pedido é difundido, através de uma tecnologia de comunicação de elevado desempenho, com base em árvores de dispersão, a todos os nós do *subcluster*;
- `CLUSTER` – nos casos atrás indicados, o pedido é difundido, através da tecnologia Fast Ethernet, com base na difusão UDP, a todos os nós do *cluster*;
- `MULTICLUSTER` – para além dos nós do *cluster*, são também contactados nós de *clusters* remotos.

5.5.2 Comunicação *intercluster*

A comunicação entre recursos de *clusters* distintos recorre ao esquema de representantes do directório, os quais acumulam a funcionalidade de reencaminhamento de mensagens. Assim, quando o sistema de despacho de um determinado contexto conclui que o destino de uma mensagem é um recurso de um *cluster* remoto, a mensagem é enviada ao representante desse *cluster*, através de uma conexão TCP. O representante reencaminhará a mensagem para o contexto do recurso destino, com base nos mecanismos de comunicação normais do R_OCl. Obviamente, se o representante for instalado num nó multi-interface, a entrega da mensagem ao contexto do recurso destino será mais célere; caso contrário, poderá ser necessário um outro reencaminhamento, para chegar ao *subcluster* do recurso destino.

O sistema de despacho do R_OCl detecta que um dado recurso se encontra num *cluster* remoto quando, a partir do identificador do recurso, tenta obter o identificador do con-

texto associado. Nesta operação, o R_OCl evoca a primitiva `rocl_query` com o parâmetro `MULTICLUSTER`, enquanto que, para a obtenção dos endereços associados a um contexto, é usado o parâmetro `CLUSTER`. Deste modo, se se tratar de um recurso de um *cluster* remoto, a resposta com o identificador do contexto alvo será devolvida por um servidor de directório de um *cluster* remoto. Esse servidor tem a noção de que a resposta é destinada a uma entidade de outro *cluster*, até porque terá que endereçar essa resposta a um representante, e, portanto, facilmente conclui que, no lugar do identificador do contexto, deverá devolver o endereço do representante do seu *cluster*. Ao receber um endereço de um representante, no lugar de um identificador de contexto, o sistema de despacho já não tenta, obviamente, obter os endereços de comunicação associados ao contexto.

O envio de mensagens a representantes obriga à gestão de várias conexões TCP. De facto, por forma a evitar constantes operações de estabelecimento de conexão, as quais penalizam o tempo de resposta, cada contexto R_OCl vai estabelecendo conexões com representantes de *clusters* remotos, à medida das necessidades, e apenas procede ao encerramento de uma conexão quando estiver esgotada a capacidade da tabela de conexões que mantém. Os servidores de directório também implementam o mesmo mecanismo de gestão de conexões.

5.6 Epílogo

A exploração de tecnologias de comunicação de elevado desempenho requer a utilização de bibliotecas de baixo-nível, cujos interfaces de programação não garantem níveis de abstracção ajustados ao desenvolvimento de aplicações paralelas/distribuídas complexas. O R_OCl, por via do paradigma da orientação ao recurso, oferece níveis de abstracção superiores, sem comprometer os níveis de desempenho característicos destas tecnologias.

A possibilidade de combinar múltiplas tecnologias de comunicação num *cluster* e desta forma constituir um sistema heterogéneo, com múltiplas partições tecnológicas, requer suporte adequado das bibliotecas de comunicação. Algumas bibliotecas de comunicação de nível intermédio permitem a troca de informação entre quaisquer máquinas de um *cluster* deste tipo. No caso do R_OCl, os recursos de uma aplicação podem ser arbitrariamente dispersos pelos nós de um *cluster* heterogéneo ou mesmo pelos nós de vários *clusters* geograficamente distantes, dando lugar à operação em ambiente *multicluster*.

O R_OCl constitui, por si só, uma ferramenta válida para a programação de aplicações capazes de explorar *clusters* heterogéneos, do ponto de vista das tecnologias de interligação de nós. No entanto, não são fornecidos quaisquer mecanismos para a distribuição de recursos pelos nós do *cluster*. Tais mecanismos são deixados a cargo das camadas de software de níveis superiores.

Capítulo 6

Comunicação em *clusters* multi-SAN

No desenho do R_OCl deu-se especial importância ao suporte multitecnologia e à capacidade de expansão, isto é, à possibilidade de incluir suporte para novas tecnologias de comunicação. Neste sentido, houve a preocupação de não vincular o funcionamento do R_OCl a uma tecnologia em particular.

Na implementação, foram definidas como tecnologias alvo a Myrinet e a Gigabit Ethernet. Dado que o R_OCl se assume como uma biblioteca de nível intermédio, a exploração destas tecnologias faz-se por via de bibliotecas de comunicação de baixo-nível, nomeadamente o GM, para exploração da Myrinet, e o M-VIA, para exploração da Gigabit Ethernet.

Em alguns sistemas de comunicação, como é o caso do Madeleine e do DECK, é dada a possibilidade de as aplicações explorarem *clusters* heterogéneos, onde dois *subclusters* são interligados por via de um nó multiconectado. O R_OCl generaliza tal funcionalidade, integrando-a no modelo de comunicação orientada ao recurso.

Neste capítulo são apresentados os aspectos mais relevantes da implementação do R_OCl sobre GM e M-VIA. Tendo em conta que estes subsistemas são significativamente distintos, as opções de desenho que foram tomadas dotaram a plataforma da flexibilidade necessária à inclusão expedita de suporte para outras tecnologias de comunicação.

6.1 Considerações gerais

O GM é a biblioteca de comunicação proposta pela empresa Myricom, para tirar partido da sua tecnologia de comunicação – a Myrinet. O VIA é uma especificação que define a arquitectura para o interface entre controladores de rede de elevado desempenho e sistemas

de computação. O M-VIA é praticamente a única implementação VIA, de uso livre, com suporte para controladores Gigabit e Fast Ethernet de vários fabricantes.

Estas duas bibliotecas, apesar das suas diferenças ao nível do modelo de comunicação, partilham uma característica fundamental: são bibliotecas de nível utilizador que permitem contornar o sistema operativo.

6.1.1 Comunicação sem cópias

A utilização do GM e do VIA obriga ao registo de zonas de memória para que o processo de troca de mensagens não envolva cópias de memória. O registo de regiões de memória envolve a marcação das páginas de memória associadas e a entrega dos endereços reais aos subsistemas de comunicação.

O R_OCl inclui facilidades para as aplicações obterem previamente um tampão, que é usado em envios posteriores. A zona de memória associada a esse tampão é registada pelo R_OCl, tendo em conta os dois subsistemas de comunicação.

A recepção de mensagens, ao nível de uma aplicação, nestes dois subsistemas, apenas tem lugar se, antes da chegada de qualquer mensagem, o programador providenciar blocos de memória registada suficientemente grandes para conter as mensagens recebidas pelo hardware de comunicação. No caso contrário, essas mensagens serão descartadas ao nível do controlador ou da biblioteca de baixo-nível.

Dado que o modelo de comunicação do R_OCl não obriga o programador a fazer qualquer previsão sobre as mensagens que a aplicação espera vir a receber, a alocação de blocos de memória pré-registada e a indicação dos seus endereços aos subsistemas de comunicação será da total responsabilidade do sistema R_OCl.

Tanto no GM como no VIA, um bloco de memória pré-registada serve para armazenar uma única mensagem recebida. Isto significa que o sistema R_OCl deverá reabastecer constantemente os subsistemas de comunicação para que nenhuma mensagem deixe de ser entregue.

6.1.2 Endereçamento

O modelo de comunicação do GM especifica a utilização de portos para o envio e recepção de mensagens. Uma aplicação deverá deter um porto para poder enviar mensagens, independentemente do número de destinos envolvidos. Cada mensagem é endereçada a um porto de uma aplicação remota, cabendo a essa aplicação a tarefa de evocar um primitiva de recepção sobre esse porto.

Em cada nó do *cluster* podem ser abertos até oito portos, por cada controlador Myrinet

instalado, sendo usado um par $\langle controlador, porto \rangle$ para identificar cada porto. Cada controlador hardware possui um identificador numérico único (no contexto do *cluster*) atribuído por um serviço GM, sendo os portos numerados de 1 a 8 em cada controlador.

A especificação VIA determina um modelo de comunicação baseado em conexões – comunicação orientada à conexão. Uma conexão VIA não é mais que um emparelhamento de VIs (*Virtual Interfaces*), em que duas aplicações (ou componentes aplicacionais) que pretendam trocar mensagens deverão, cada uma, criar um VI e proceder ao emparelhamento destes.

Em geral, para que o envio de uma mensagem para um dado destino seja possível, uma aplicação deverá criar um VI e requerer uma conexão para esse destino, com base num endereço de rede VIA. As mensagens são sempre endereçadas a um VI local, que está conectado a um VI remoto.

No M-VIA, o número de VIs está limitado a 1024 por cada controlador Gigabit ou Fast Ethernet instalado. Uma aplicação paralela/distribuída, a executar num *cluster* com n nós e a utilizar f fios-de-execução por nó, com trocas de informação entre todos os fios-de-execução, necessitaria $f^2 \times (n-1)$ VIs, por cada nó. Mesmo no caso de um *cluster* de dimensões reduzidas, 16 nós, por exemplo, o número de fios-de-execução em cada nó ficará limitado a 8. No caso do GM, oito será o limite para o número de fios-de-execução, independentemente do número de nós do *cluster*. Estas limitações são ultrapassadas pelo mecanismo de despacho do R_OCl.

6.1.3 Envio e recepção

A especificação VIA impõe a utilização de registos descritores para o envio e recepção de mensagens. Um registo descritor é um bloco de memória registada, contendo informação sobre uma operação de envio ou de recepção, incluindo endereços de tampões, tamanho dos dados a enviar/receber e demais informação de controlo. Na recepção, os registos descritores servem, numa primeira fase, para especificar zonas de memória registada onde podem ser armazenadas mensagens recebidas pelo controlador. Estes registos descritores de recepção são processados de acordo com a ordem em que são entregues ao VIA, o que dificulta a recepção de mensagens de tamanhos variados, principalmente quando na recepção de desconhece a ordem pela qual estas serão enviadas. A título de exemplo, se forem especificados dois tampões de tamanhos t e $2t$, com base em dois registos descritores fornecidos por esta ordem, e se ocorrer a chegada, em primeiro lugar, de uma mensagem com um tamanho $x : t < x \leq 2t$, essa mensagem será descartada, pois o primeiro registo descritor não especifica um tampão com dimensão suficiente para armazenar a mensagem.

A biblioteca GM apenas requer a indicação do endereço de memória respeitante aos dados

a enviar e do tamanho desses dados, através de dois parâmetros da primitiva de envio. Na recepção, quando múltiplos blocos de memória registada estão disponíveis, a mensagem recebida pelo controlador é armazenada no bloco que minimiza o desperdício de memória, simplificando o processo de recepção de mensagens de tamanhos variados.

As primitivas disponibilizadas pelo GM para o envio e recepção de mensagens, pelo facto de não ser segura a sua utilização em ambientes com múltiplos fios-de-execução, levantam um problema adicional. No caso do VIA, tal não constitui um problema, mas a implementação M-VIA não cumpre por completo a especificação e, por conseguinte, algumas primitivas não podem ser evocadas concorrentemente. A solução passa pela utilização de mecanismos de coordenação POSIX, que não levantam dificuldades nas primitivas não bloqueantes, mas exigem cuidados especiais e conhecimento de pormenores da implementação VIA e, principalmente, da biblioteca GM, por forma a evitar o bloqueio de todos os fios-de-execução de uma aplicação quando são evocadas primitivas bloqueantes.

6.2 Gestão de tampões

O R_OCl utiliza uma colecção de tampões pré-alocados e pré-registados, para assegurar a comunicação sem cópias. Dado que, tanto a alocação como o registo de zonas de memória são operações onerosas, estes tampões são preparados na fase de arranque da biblioteca R_OCl e, durante a execução da aplicação, é feita a sua gestão de forma a minimizar novas alocações e registos. O programador deverá solicitar tampões para armazenar os dados envolvidos numa operação de envio e a biblioteca R_OCl deverá, por sua iniciativa, entregar alguns tampões aos vários subsistemas de comunicação.

A figura 6.1 exemplifica o funcionamento do sistema de gestão de tampões usado pelo R_OCl, tendo em conta que tanto o sistema como o programador despoletam a movimentação de tampões.

6.2.1 Operação comandada pela aplicação

As aplicações deverão solicitar tampões de tamanho adequado, antes de enviarem mensagens (primitiva `rocl_bfget`). O R_OCl facultará um tampão da colecção previamente reservada, eliminando desta forma os tempos de alocação de memória e sequente registo.

Depois da evocação da operação de envio, a aplicação poderá monitorizar o estado do tampão associado, no sentido de averiguar se o envio (assíncrono) foi concluído (primitiva `rocl_bfstat`). Mediante a indicação de um valor negativo para o parâmetro `timeout`, a aplicação poderá, inclusivamene, bloquear até à conclusão do envio. Após a conclusão do envio, o tampão ficará na posse da aplicação, podendo ser reutilizado num outro envio.

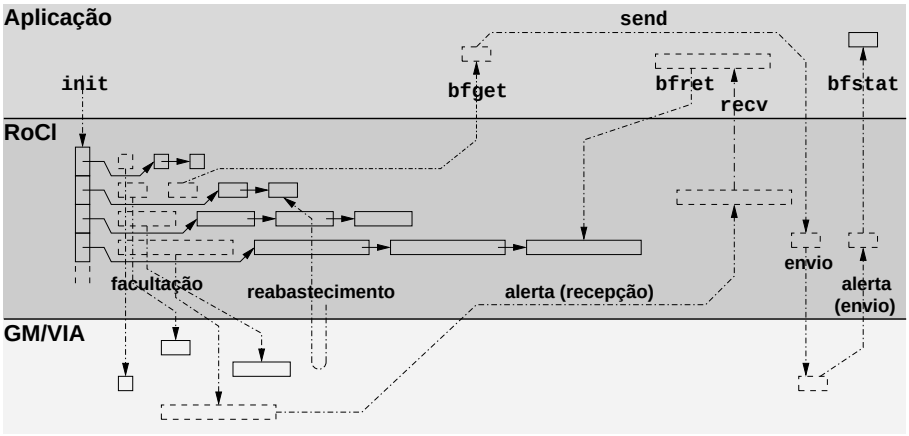


Figura 6.1: Gestão de tampões.

Quando a aplicação tem sucesso numa primitiva de recepção, isto é, a fila de recepção do R_OCl tem uma mensagem que cumpre os requisitos da aplicação, o tampão correspondente à mensagem é cedido à aplicação (primitiva `rocl_recv`). Este tampão, dará, em primeiro lugar, acesso aos dados da mensagem, mas, posteriormente, poderá também ser usado para um envio.

Quando um determinado tampão solicitado pela aplicação ou devolvido por uma operação de recepção deixar de ser necessário à aplicação, esta poderá devolvê-lo para ser reintegrado na colecção de tampões da biblioteca (primitiva `rocl_bfret`). Se a aplicação tenciona devolver à biblioteca R_OCl um dado tampão após a conclusão de um envio, então poderá notificar a biblioteca (através da primitiva `rocl_bftoret`) ainda antes da evocação da primitiva de envio, para que a biblioteca proceda à reintegração automática do tampão logo após a notificação de conclusão de envio. Desta forma a aplicação não terá que evocar as primitivas `rocl_bfstat` e `rocl_bfret`.

6.2.2 Operação comandada pela biblioteca

Como já foi referido, o R_OCl é responsável por facultar aos subsistemas GM e VIA os tampões adequados para viabilizar a recepção de mensagens.

Quando uma mensagem chega, um tampão é usado para a armazenar e o R_OCl é alertado, de forma a tomar conhecimento do endereço desse tampão. O R_OCl terá então que facultar um novo tampão ao subsistema que recebeu a mensagem, acautelando o descarte de mensagens. No entanto, até ao momento em que o R_OCl efectivamente faculta um novo tampão ao subsistema, podem chegar uma ou mais mensagens, as quais serão descartadas. Por forma a evitar tal situação, o R_OCl fornece a cada um dos subsistemas, em

antecipação, vários tampões.

Numa situação óptima, desde que a aplicação devolva tampões atempadamente, é possível gerir a colecção de tampões pré-alocados e pré-registados na fase de arranque, de forma a satisfazer as necessidades da aplicação. No entanto, no caso de o número destes tampões atingir um dado nível crítico, o R_OCl procederá ao reabastecimento da sua colecção, isto é, procederá à alocação e registo de novos tampões.

6.2.3 Tamanhos de tampões

Para o envio de mensagens, a aplicação solicita tampões com tamanhos adequados aos dados a enviar, reservando-se algum espaço no tampão, para incluir informação de controlo.

Na recepção, a biblioteca terá que lidar com mensagens de tamanho variável, sendo necessário estimar os tamanhos dos tampões necessários. Uma possibilidade de ultrapassar esta dificuldade passa pela utilização de tampões dimensionados por excesso, por forma a permitir o armazenamento de qualquer mensagem recebida. Esta opção leva a gastos de memória excessivos, a não ser que os dados da mensagem sejam copiados dos tampões para zonas de memória da aplicação, libertando imediatamente os tampões para novas recepções.

A abordagem seguida no R_OCl foi a de fixar um tamanho máximo para os tampões e trabalhar com dimensões pouco variadas, restringindo a variedade de tamanhos de tampões manipulados a uma potência de base 2, em que o expoente é limitado superiormente. Assim, para suportar a recepção de mensagens de tamanho variado, a biblioteca R_OCl terá apenas que facultar, a cada um dos subsistemas de comunicação, pelo menos um tampão para cada um dos tamanhos de tampão previstos.

Deste modo, para um pedido de um tampão para preparação de uma mensagem de x bytes, por exemplo, a biblioteca R_OCl devolverá um tampão com tamanho 2^y, em que $2^{y-1} < (x + inf.controlo) \leq 2^y$.

6.2.4 Formato dos tampões

A manipulação de um tampão no R_OCl pressupõe a delimitação de quatro zonas distintas, conforme se mostra na figura 6.2. A zona à esquerda é usada para guardar informação de controlo relativa ao tampão, com interesse para qualquer subsistema de comunicação em utilização, ou informação específica de cada subsistema (um fragmento por cada subsistema). A zona seguinte é destinada ao armazenamento dos dados da mensagem propriamente ditos, seguindo-se-lhe informação de controlo relativa à mensagem. Dependendo do tamanho dos dados da mensagem, haverá a considerar uma última zona do tampão que

eventualmente não será ocupada.

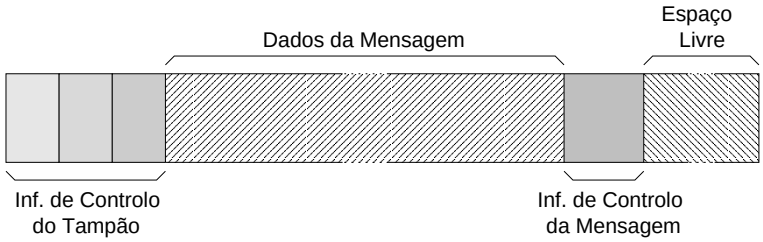


Figura 6.2: Formato de um tampão.

Quando é criado um tampão para responder a um pedido de uma dada dimensão, o R_OCl contabiliza o espaço adicional necessário para controlo (do tampão e da mensagem) e devolve o endereço imediatamente após a área delimitada para a informação de controlo do tampão. Dado que são fixados pelo R_OCl os tamanhos possíveis para os tampões a manipular, para algumas mensagens serão disponibilizados tampões com dimensões por excesso, restando algum espaço livre no final desses tampões.

Note-se que apenas o sistema R_OCl tem noção das zonas que no tampão não são preenchidas com dados da mensagem. Tanto as aplicações como os subsistemas de comunicação apenas conhecem o endereço de memória que dá acesso à zona destinada ao armazenamento dos dados. No caso dos tampões entregues à aplicação, cabe a esta respeitar o tamanho máximo dos dados a armazenar.

6.3 Despacho de mensagens

O mecanismo de despacho gere as abstracções fornecidas pelas bibliotecas de baixo-nível, de forma a multiplexar as mensagens produzidas por um vasto leque de entidades do nível aplicacional. Esta tarefa complica-se pelo facto de os subsistemas de comunicação apresentarem modelos de operação completamente distintos; o VIA é orientado à conexão e o GM é não-orientado à conexão.

6.3.1 Endereçamento baixo-nível

O interface com os subsistemas de comunicação é relativamente simples: um dado subsistema, quando detecta a chegada de uma mensagem, alerta o R_OCl e indica-lhe o endereço do tampão onde a mensagem foi armazenada, cabendo ao R_OCl a tarefa de inserir um registo descritor para essa mensagem na fila de recepção.

Numa operação de envio o R_OCl terá de endereçar a mensagem a um porto GM remoto

ou a um VI VIA local. O objectivo é fazer chegar a mensagem ao contexto R_OCl no qual o recurso alvo foi criado, usando a seguinte estratégia:

GM: É aberto um porto, o qual é anunciado através do registo dos identificadores numéricos do controlador e do porto no serviço de directório, para permitir a conversão de identificadores de recursos em portos GM.

VIA: É registado no directório o endereço de rede VIA, formado pelo endereço físico do controlador de rede (um endereço MAC) e por um discriminador. O discriminador VIA pode ser entendido como um porto UDP ou TCP, mas o programador é livre de usar qualquer sequência de bytes, à sua escolha, para esse efeito. No R_OCl usa-se como discriminador o identificador global do contexto, devolvido pelo directório aquando do seu registo. Deste modo, poderá ser solicitado o estabelecimento de uma conexão com a contexto destino adequado, por forma a fazer chegar uma mensagem a um dado recurso.

6.3.2 Detalhes das conexões VIA

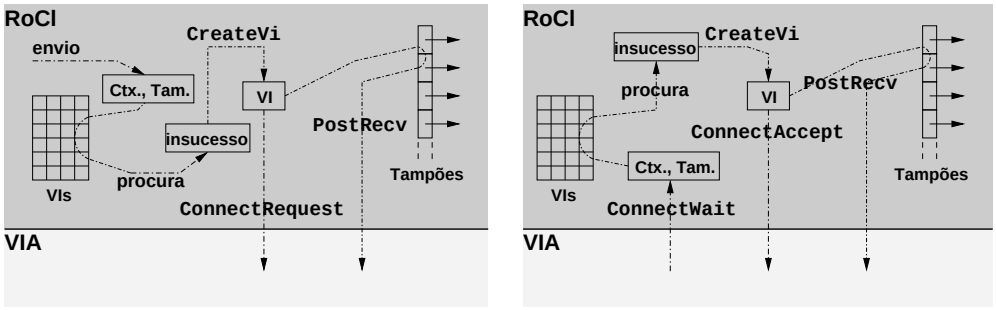
Dado que o estabelecimento de uma conexão é uma operação onerosa, do ponto de vista computacional e comunicacional, é conveniente optar por conexões persistentes. Idealmente, num dado contexto R_OCl, deveria ser criado um VI e estabelecida uma conexão com um outro contexto R_OCl remoto no momento em que, pela primeira vez, é endereçada uma mensagem a um recurso desse contexto remoto.

A utilização de uma única conexão VIA para interligar dois contextos R_OCl levanta problemas na recepção de mensagens com tamanhos variados. De facto, os registos descritores de recepção, especificados para um dado VI (conexão), são sempre processados pela ordem de entrega ao VIA, independentemente do tamanho do tampão especificado. Para contornar tal dificuldade, o R_OCl mantém, em cada contexto, um conjunto de conexões por cada contexto remoto – uma conexão para cada tamanho de tampão possível.

A figura 6.3 apresenta os passos associados ao estabelecimento de uma conexão entre dois contextos.

No contexto origem (figura 6.3(a)), no momento de envio de uma mensagem, o R_OCl utiliza o identificador do contexto destino juntamente com o tamanho da mensagem para encontrar o VI correspondente à conexão alvo. Na ausência de uma conexão estabelecida para ligação ao contexto destino é criado um novo VI e é enviado um pedido de estabelecimento de conexão ao contexto remoto.

O pedido de conexão é endereçado ao endereço de rede VIA registado no directório R_OCl, aquando da criação do contexto remoto. Previamente à solicitação da conexão, o R_OCl fa-



(a) Origem (b) Destino

Figura 6.3: Gestão de conexões VIA no R_OCl.

culta ao subsistema VIA alguns tampões, para que possam ser armazenadas as mensagens eventualmente recebidas logo a seguir ao estabelecimento da conexão.

O mecanismo de despacho do R_OCl mantém um fio-de-execução à espera de pedidos de conexão. Assim, no contexto destino (figura 6.3(b)), após a detecção de um pedido de conexão, é criado um VI. Note-se que as conexões VIA são bidireccionais, pelo que este VI pode ser usado tanto para receber como para enviar mensagens de/para o contexto que solicitou o estabelecimento de conexão.

Depois de facultar alguns tampões de recepção ao subsistema VIA, o contexto R_OCl destino informa o contexto origem acerca da aceitação da conexão.

6.3.3 Actualização de tampões no envio e na recepção

Aquando do envio de uma mensagem, o R_OCl terá, antes de mais, que proceder ao preenchimento da informação de controlo da mensagem, com base nos parâmetros indicados pela aplicação na primitiva correspondente. Haverá ainda lugar à actualização da informação de controlo do tampão, dado que este será entregue a um dos subsistemas de comunicação, através dos mecanismos de envio disponibilizados – primitiva `gm_send`, no caso do GM, e primitiva `VipPostSend`, no caso do VIA.

Após a entrega da mensagem a um subsistema de comunicação, o R_OCl não mantém qualquer referência para o tampão associado. Cada um dos subsistemas inclui funcionalidades para geração de eventos respeitantes ao sucesso ou insucesso do envio, nomeadamente, o mecanismo de *callback*, no caso do GM, e as filas de notificações de término, no caso do VIA. Assim, após a conclusão de um envio, ou logo que uma situação de erro seja detectada, o R_OCl recebe uma notificação, que inclui informação sobre o tampão envolvido – o endereço que dá acesso à zona de dados, no caso do GM, e o registo descritor, no

caso do VIA, o qual inclui esse mesmo endereço. Com base neste endereço, e dado que a informação de controlo do tampão tem tamanho fixo, é possível recuperar a totalidade da informação do tampão.

No que diz respeito à recepção de mensagens, os procedimentos são semelhantes: a primitiva `gm_provide_receive_buffer`, no caso do GM, e a primitiva `VipPostRecv`, no caso do VIA, permitem entregar tampões aos subsistemas de comunicação, enquanto as notificações de chegada de mensagens dão a conhecer, ao R_OCl, o endereço dos dados da mensagem ou o registo descritor da mensagem.

6.4 Controlo de fluxo

As bibliotecas de comunicação de baixo-nível implementam alguns mecanismos para detecção de erros de comunicação, apesar de as tecnologias de comunicação de elevado desempenho normalmente usadas na interligação de nós de *clusters* serem consideradas altamente fiáveis. Tais mecanismos são importantes para detectar situações de não entrega de mensagens devido à inexistência de tampões apropriados para o armazenamento dessas mensagens no destino. De facto, tanto o modelo de comunicação do GM como o do VIA pressupõem a facilitação, por parte da aplicação no destino, de um tampão de dimensão adequada antes da chegada da mensagem.

No caso do GM, se a mensagem não puder ser entregue, devido à falta de um tampão no destino, a origem é notificada. No caso do M-VIA, dado que apenas o segundo nível de fiabilidade na comunicação é implementado (a especificação VIA refere três níveis), a origem nem sequer é notificada quando a mensagem não é efectivamente entregue por falta de um tampão. O segundo nível de fiabilidade especifica que uma mensagem é considerada correctamente enviada se houver garantia que foi entregue ao controlador destino. No entanto, isso não impede que a mensagem seja descartada, pelo controlador, pelo facto de não existir um tampão.

Apesar de o mecanismo de despacho tentar manter os subsistemas de comunicação munidos de tampões, facultando um novo tampão sempre que é recebida uma mensagem, nada garante que, num dado destino, o R_OCl seja capaz de acompanhar o funcionamento do hardware de comunicação. De facto, várias mensagens poderão chegar de tal forma próximas no tempo que o sistema de despacho do R_OCl poderá não conseguir repor tampões ao ritmo desejado e, consequentemente, perder-se-ão algumas mensagens.

Se o subsistema em utilização for o GM, o insucesso no envio, nestas circunstâncias, é notificado. O R_OCl poderá, em consequência, proceder ao reenvio da mensagem, no pressuposto de que, entretanto, já houve tempo para a reposição de tampões no destino. No entanto, esta estratégia consome recursos de comunicação (largura de banda e tempo

de processamento dos controladores envolvidos), para além de fazer aumentar o tempo de resposta. A estratégia seguida no R_OCl baseia-se na utilização do sistema de créditos oferecido pelo GM, com o intuito de implementar um mecanismo de controlo de fluxo.

O sistema de créditos do GM, quando usado, obriga o programador a verificar a existência de créditos antes do envio de uma mensagem. Cada envio consome um crédito e, no momento da recepção da notificação de entrega da mensagem, esse crédito é devolvido. Assim, se no destino não existirem tampões para o armazenamento das mensagens, rapidamente os créditos se esgotarão (inicialmente estão disponíveis algumas dezenas de créditos, dependendo da memória disponível no controlador) e o R_OCl poderá suspender os envios posteriores, até que sejam devolvidos créditos. Note-se que o GM se encarrega da entrega efectiva de qualquer mensagem, desde que exista disponível um crédito para o efeito, mesmo que, no destino, a disponibilização de um tampão ocorra num momento posterior à evocação da primitiva `gm_send`. Na prática, os créditos correspondem à capacidade do subsistema de comunicação manter mensagens em situação pendente.

No caso do M-VIA, não existem mecanismos que facilitem a implementação de um sistema de controlo de fluxo, nem sequer há forma de detectar o insucesso de um envio (motivado pela falta de tampões, no destino). Deste modo, a abordagem seguida no R_OCl passa pela implementação de um mecanismo de janela deslizante, algo semelhante ao existente no TCP. Este mecanismo só é possível pelo facto de o VIA ser orientado à conexão.

No momento do estabelecimento de uma conexão VIA, para um dado tamanho de tampão, os dois contextos R_OCl envolvidos fixam como tamanho das suas janelas de envio o equivalente ao número de tampões facultados para o armazenamento de mensagens. Por cada mensagem enviada, no destino há-de ser gasto um tampão e, portanto, a origem reduzirá em uma unidade a sua janela de envio. Quando a janela atingir o valor zero, isso significará que, no destino, foram esgotados os tampões inicialmente facultados pelo R_OCl e, na origem, os envios deverão ser suspensos.

Sempre que no destino o R_OCl é notificado da recepção de uma mensagem, um novo tampão é facultado ao subsistema de comunicação, o que se deverá reflectir na actualização da janela da origem. Para o efeito, o R_OCl recorre à inclusão de informação de controlo nas mensagens enviadas no sentido inverso ou, caso necessário, ao envio de mensagens de controlo. Qualquer mensagem enviada pelo R_OCl inclui o número de tampões que o sistema de despacho repôs, desde a última vez em que houve oportunidade de informar o outro lado da conexão. No caso de a aplicação não gerar tráfego num dado sentido da conexão, o R_OCl envia uma mensagem de controlo quando conclui que já foram repostos tampões em número significativo, ou seja, quando a janela do outro lado da conexão estiver a atingir um nível crítico. Com esta abordagem, o número de mensagens de controlo é mantido numa proporção relativamente baixa em face às mensagens das aplicações.

A evocação da primitiva R_OCl destinada ao envio de uma mensagem, numa situação de inexistência de créditos (GM) ou esgotamento da janela de envio (M-VIA), terá como efeito a inserção da mensagem na fila de envio do mecanismo de despacho.

6.5 Escrita e leitura remotas

O R_OCl disponibiliza primitivas para suporte a operações de leitura e escrita remotas, tirando partido dos mecanismos RDMA dos subsistemas de comunicação. Em relação ao GM, são exploradas as primitivas `gm_get` e `gm_put`, as quais ficaram disponíveis com a versão 2.0 da biblioteca, ficando assim facilitada a implementação do R_OCl.

No caso do M-VIA, apesar de a especificação VIA contemplar tanto as operações de leitura como as de escrita, apenas é suportada a escrita remota. No entanto, dado que um dos objectivos do R_OCl é criar um interface único para a exploração de vários subsistemas de comunicação, todas as funcionalidades deverão ser asseguradas independentemente do subsistema de comunicação. Assim, o R_OCl implementa um sistema alternativo de leitura/escrita remota baseado na troca de mensagens entre contextos, ao nível do sistema de despacho.

O suporte a operações de leitura/escrita remota baseado em mensagens tradicionais não permite alcançar níveis elevados de desempenho. No entanto, para além de permitir contornar a falta de operacionalidade de alguns subsistemas, permite alargar a operação a entidades em execução em nós que não possuem uma tecnologia de comunicação em comum. De facto, os reencaminhadores não têm qualquer possibilidade de intervenção se forem usadas primitivas RDMA das bibliotecas de comunicação de baixo-nível. Deste modo, o R_OCl, quando detecta que o recurso destino só pode ser alcançado recorrendo a um reencaminhador, usa o sistema alternativo de leitura/escrita remota.

6.6 Integração de tecnologias

O suporte a múltiplas tecnologias de comunicação é, na generalidade dos casos, entendido como a simples possibilidade de adaptar o sistema ao funcionamento em cenários variados, mediante a selecção de uma tecnologia na fase de compilação/instalação. No entanto, no R_OCl, o objectivo principal é a exploração simultânea de múltiplas tecnologias de comunicação, no âmbito de um *cluster* multi-SAN.

6.6.1 Reencaminhamento de mensagens

No *cluster* representado na figura 3.1(a), os nós Myrinet são fisicamente interligados a nós Gigabit, por via da existência de nós multi-interface, isto é, nós que dispõem de um controlador por cada tecnologia de comunicação (Myrinet e Gigabit). Do ponto de vista do R_OCl, estes nós multi-interface funcionam como reencaminhadores de mensagens, fazendo a ponte entre os subsistemas de comunicação GM e M-VIA.

A função de reencaminhamento é assegurada pelo mecanismo de despacho do R_OCl, pelo que qualquer programa R_OCl, em execução num nó multi-interface, poderá reencaminhar mensagens. No caso de a aplicação de um dado utilizador não contemplar módulos em execução num nó multi-interface, isto é, supondo que um utilizador, para uma dada aplicação, resolveu usar alguns nós Myrinet e alguns nós Gigabit, mas não recorreu a qualquer nó multi-interface, o R_OCl disponibiliza um serviço de reencaminhamento. Este serviço não é mais que um simples programa que arranca a biblioteca R_OCl, através da primitiva `rocl_init`, e decide bloquear-se por tempo indeterminado.

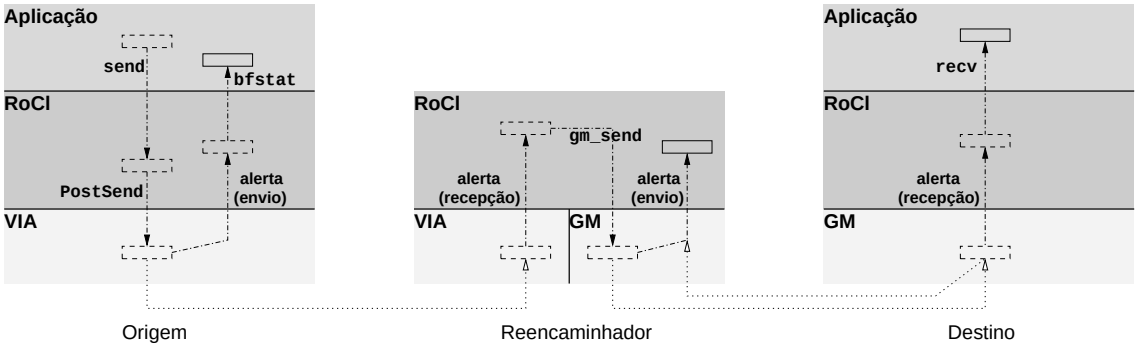


Figura 6.4: Exemplo de reencaminhamento de mensagens.

O reencaminhamento propriamente dito, do qual é apresentado um exemplo na figura 6.4, não levanta dificuldades e pode ser implementado de forma bastante eficiente. Quando o sistema de despacho de um contexto R_OCl recebe uma mensagem destinada a um outro contexto, assume-se que se trata de uma mensagem que requer reencaminhamento, bastando, para tal, entregar o tampão com a mensagem recebida ao subsistema de comunicação que permite alcançar o destino em causa. Note-se que o reencaminhamento não implica qualquer cópia de memória (entre tampões), dado que um determinado bloco de memória registada pode ser entregue tanto ao GM como ao M-VIA, independentemente de o tampão poder conter informação de controlo que, para um dado subsistema, não tem qualquer utilidade. Também não será necessário efectuar qualquer alteração da informação de controlo do tampão ou da própria mensagem, desde que a origem prepare a mensagem

como se ela fosse entregue directamente ao destino final.

A dificuldade está em fazer chegar ao reencaminhador a mensagem que, numa situação de conectividade simples, seria entregue directamente ao destino. A abordagem do R_OCl assenta na exploração da informação registada no directório para os contextos. No momento da criação de um contexto (o arranque da biblioteca R_OCl), o programador indica se autoriza a activação do mecanismo de reencaminhamento, através do parâmetro `bridge` da primitiva `rocl_init` (ver tabela 5.1). No caso de o contexto ter acesso a várias tecnologias de comunicação, e desde que a indicação do programador seja nesse sentido, o registo do contexto incluirá um atributo especial, indicando essa funcionalidade. No momento do envio de uma mensagem, quando o endereço de comunicação devolvido pelo directório, respeitante ao contexto destino, não corresponde à tecnologia de comunicação da origem, o R_OCl tenta obter, também a partir do directório, o endereço de um reencaminhador. A mensagem é então endereçada ao reencaminhador, sem que o tampão associado necessite de qualquer alteração, apesar de não ser enviada directamente para o seu destino.

O mecanismo de reencaminhamento é actualmente pouco elaborado, tolerando apenas um único nó intermédio, entre a origem e o destino, o que é, no entanto, suficiente para o tipo de *clusters* que se pretende suportar. Além disso, não fará sentido considerar um *cluster*, baseado em tecnologias que usam comutadores, como um caso de uma topologia de rede complexa.

6.6.2 Selecção do subsistema de comunicação

Os nós de um *cluster* que possuam múltiplos interfaces de comunicação, para além de poderem desempenhar o papel de reencaminhadores, podem, obviamente, ser usados como simples nós computacionais, onde alguns módulos de uma aplicação paralela executam. Neste caso, o sistema de despacho do R_OCl terá que decidir qual o subsistema de comunicação a usar, sempre que a aplicação evoca a primitiva de envio.

Esta decisão será óbvia no caso de o contexto destino possuir uma única tecnologia de comunicação, o que facilmente é detectado pela consulta do directório R_OCl. Mas, se o contexto destino residir numa máquina dotada de controladores Myrinet e Gigabit, haverá duas opções: enviar a mensagem através do GM ou através do M-VIA. O critério fundamental de selecção é o desempenho, sendo usado, por norma, o GM, o qual garante melhor desempenho (ver secção 8.2).

No entanto, se, num dado momento, o subsistema com melhor desempenho não puder garantir o envio imediato, devido ao mecanismo de controlo de fluxo, então opta-se pelo envio através do outro subsistema, desde que não se encontre, também, indisponível para o envio imediato. No caso de nenhum subsistema se encontrar disponível, a mensagem é

inserida na fila de envio e, posteriormente, será enviada através do subsistema que ficar disponível em primeiro lugar.

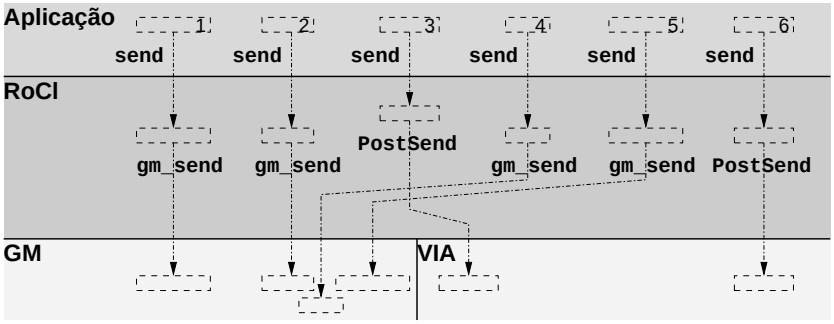


Figura 6.5: Selecção do subsistema em envios consecutivos.

Numa situação em que, a partir de um dado contexto R_oCl, é desencadeada uma série de envios consecutivos, tendo como destino o mesmo contexto, a experiência mostrou ser possível alcançar desempenhos mais elevados se, em vez de levar à exaustão o subsistema GM, obrigando o mecanismo de controlo de fluxo a actuar, e só recorrer ao M-VIA em último recurso, se procedesse ao envio alternado de mensagens através dos dois subsistemas. Na figura 6.5 mostra-se de que forma seis mensagens geradas em sequência podem ser enviadas, com base na combinação das duas tecnologias; por cada duas mensagens enviadas através do GM, é enviada uma através do VIA.

A estratégia da alternância de tecnologia de comunicação é válida apenas para sequências de mensagens com tamanho similar. Deste modo, logo que o sistema de despacho do R_oCl é confrontado com o envio de uma mensagem de tamanho significativamente diferente do da mensagem anterior, a estratégia de alternância é interrompida.

6.6.3 Fragmentação de mensagens

O modelo de comunicação do R_oCl impõe a utilização de tampões, para o armazenamento de mensagens, para os quais é definido um tamanho máximo. Devido à utilização do M-VIA, esse tamanho é fixado em 32kbytes, tamanho esse que corresponde à dimensão máxima de um segmento de dados no M-VIA. Segundo estudos apresentados em [Gusella 90] e mais tarde corroborados em [Chen 02], 80% das mensagens trocadas nas aplicações paralelas têm menos de 4kbytes e apenas 8% ultrapassam 8kbytes, pelo que o limite estabelecido para os tampões não será impeditivo para a codificação da generalidade das aplicações. No entanto, em algumas circunstâncias, é útil ter disponível uma primitiva de comunicação que permita enviar mensagens de qualquer tamanho.

O R_OCl dá a possibilidade de se especificar, como parâmetro da primitiva `rocl_send`, o endereço de um bloco de memória qualquer, em vez de o endereço de um tampão. Neste caso, a quantidade de dados a enviar (indicada também como parâmetro da primitiva `rocl_send`) poderá ultrapassar o tamanho máximo de um tampão. No entanto, pelo facto de se tratar de memória do espaço de endereçamento da aplicação, a mensagem não poderá ser enviada sem recorrer a cópias de memória. Note-se que o registo da zona de memória envolvida, imediatamente antes do envio, não é uma solução, pelo facto de o M-VIA impor um limite para o tamanho das mensagens. Além disso, a gestão de memória dos sistemas operativos actuais não garante a possibilidade de marcar todas páginas correspondentes a uma zona de memória de tamanho arbitrário.

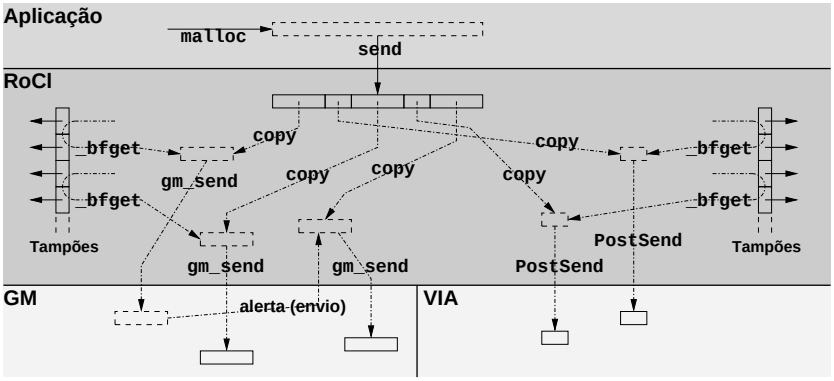


Figura 6.6: Despacho dos fragmentos de uma mensagem.

A abordagem seguida no R_OCl compreende a fragmentação da mensagem correspondente à zona de memória indicada pelo programador, conforme se mostra na figura 6.6. Cada fragmento é enviado como se se tratasse de uma mensagem simples, isto é, uma mensagem baseada num tampão. A obtenção de tampões e a cópia de fragmentos para tampões é da exclusiva responsabilidade do sistema de despacho. No caso de estarem disponíveis os dois subsistemas de comunicação, o R_OCl alterna os envios entre ambos, conforme exposto anteriormente.

No destino, o R_OCl encarrega-se de alocar uma zona de memória suficientemente grande para armazenar todos os fragmentos da mensagem. Os vários fragmentos são reorganizados com base em informação de controlo incluída pela origem. Após a chegada do último fragmento, a mensagem é inserida na fila de recepção, tal como se se tratasse de uma mensagem simples.

6.7 Epílogo

A implementação actual do sistema R_OCl tomou por base as bibliotecas de comunicação de baixo-nível GM e M-VIA. O sistema tira partido das funcionalidades mais relevantes de ambas as bibliotecas, mantendo o objectivo de oferecer um interface de programação único, mais conveniente aos programadores.

O mecanismo de gestão de tampões é fundamental para minimizar o número de operações de alocação e cópia de memória. Com base neste mecanismo, o R_OCl permite tirar partido da comunicação livre de cópias oferecida pelas bibliotecas de baixo-nível, sem comprometer a conveniência do interface de programação.

A adequação ao paradigma da orientação ao recurso é conseguida por via do sistema de despacho de mensagens. Este sistema é responsável por gerir um conjunto reduzido de pontos de comunicação, oferecidos pelas bibliotecas de baixo-nível, e dotar quaisquer recursos de capacidades comunicacionais.

A operação simultânea dos subsistemas GM e M-VIA, por parte do sistema de despacho de mensagens, garante a interoperabilidade de nós pertencentes a *subclusters* distintos, num *cluster* multi-SAN. A exploração combinada dos dois subsistemas permitiu ainda ultrapassar o desempenho alcançável com recurso unicamente a um deles em nós multiconectados.

Apesar de terem sido seleccionados apenas dois subsistemas de comunicação, para a implementação actual, o R_OCl encontra-se estruturado de forma a que facilmente sejam integrados novos subsistemas. Em especial, devido ao facto de o VIA ser orientado à conexão e o GM ser não-orientado à conexão, foi possível contemplar situações que abrangem variadas plataformas de comunicação de baixo-nível, aumentando a flexibilidade da implementação R_OCl. A título de exemplo, a inclusão do protocolo UDP, como subsistema de comunicação, foi facilmente realizada, tomando como base os módulos desenvolvidos para suportar o GM.

Capítulo 7

Interface de programação

Os conceitos da modelação de aplicações orientada ao recurso foram apresentados nos capítulos 3 e 4, mas não foram expostas as ferramentas ao dispor dos administradores e programadores para o desenvolvimento e execução de aplicações. A plataforma $m_{\varepsilon}\mu$ corresponde à exploração plena do R_OCl, o qual constitui uma primeira abordagem ao paradigma da orientação ao recurso, isto é, o interface de programação bem como todos os serviços e programas de sistema subjacentes ao $m_{\varepsilon}\mu$ foram implementados como uma camada de nível superior, sobre a camada de mais baixo nível que é o R_OCl.

Tendo o R_OCl sido apresentado nos capítulos 5 e 6, é agora possível expor os detalhes relativos ao interface $m_{\varepsilon}\mu$, nomeadamente a forma como algumas primitivas são implementadas, explorando a funcionalidade do R_OCl, e o modo de utilização associado a cada uma das primitivas e comandos do sistema. Essencialmente, são expostos neste capítulo os aspectos mais relevantes respeitantes à manipulação de abstracções do R_OCl, com o intuito de oferecer mecanismos mais adequados à programação de aplicações.

Este capítulo tem ainda como propósito expor de forma mais detalhada o sistema de suporte à execução $m_{\varepsilon}\mu$, principalmente no que diz respeito aos mecanismos para manipulação de recursos físicos e lógicos.

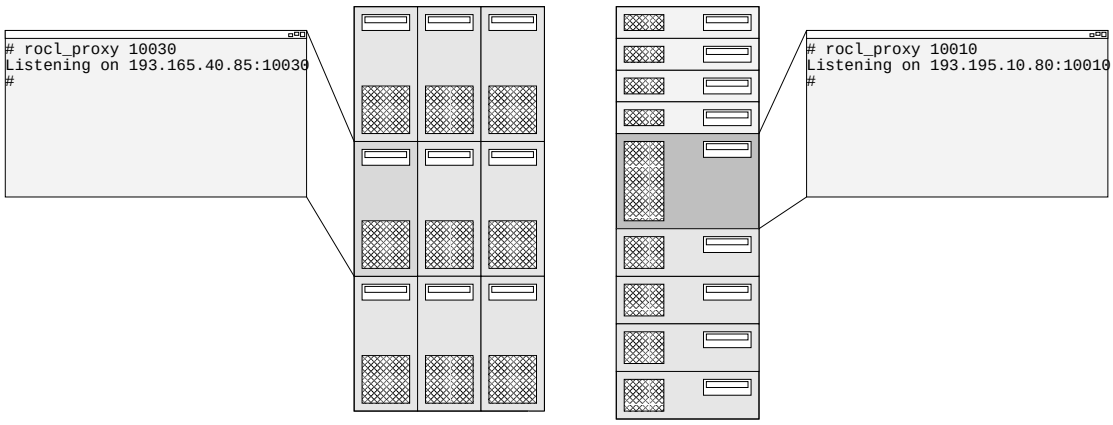
7.1 Configuração do sistema

A plataforma $m_{\varepsilon}\mu$ apresenta-se ao programador como uma biblioteca de primitivas, que permitem codificar aplicações em concordância com o paradigma da orientação ao recurso. O funcionamento correcto desta biblioteca bem como dos demais programas de sistema que compõem a plataforma, carece de parametrização adequada, por parte de administradores e utilizadores do *cluster*. É ainda necessário proceder à caracterização inicial do *cluster*, por forma a que as aplicações possam explorar o hardware disponível.

7.1.1 Serviços e programas de sistema

A operação da plataforma $m_{\mathcal{E}}\mu$ baseia-se, em grande parte, na funcionalidade oferecida pelo R_OCl, como já foi referido. Deste modo, a exploração de um (ou mais) *clusters*, através do $m_{\mathcal{E}}\mu$, requer a correcta instalação e configuração dos serviços R_OCl.

No caso de se pretender a exploração de vários *clusters*, será necessário, antes de mais, instalar um representante R_OCl em cada *cluster*, através do comando `rocl_proxy`. Para tal, deverá ser seleccionada, em cada *cluster*, uma máquina com um endereço IP válido e, no caso de se tratar de um *cluster* heterogéneo, preferencialmente com múltiplos interfaces de comunicação. A figura 7.1 mostra o arranque de representantes em dois *clusters*, localizados em Braga e Bragança.

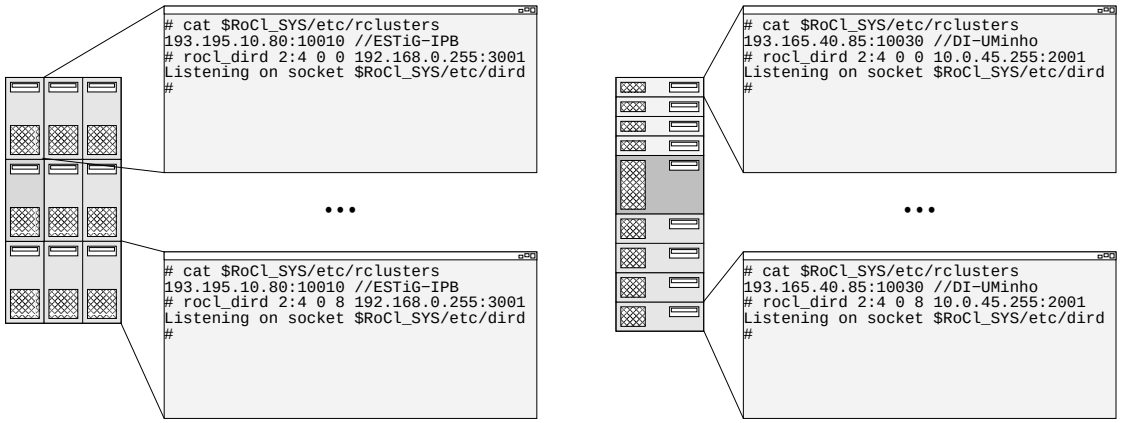


(a) *Cluster* DI-UMinho (b) *Cluster* ESTiG-IPB

Figura 7.1: Arranque de representantes.

De seguida deverão ser colocados em execução os servidores de directório, um por cada máquina de cada *cluster*, através do comando `rocl_dird`. Previamente ao arranque de um servidor, é necessário especificar os endereços IP e portos TCP correspondentes aos representantes de todos os *clusters* remotos conhecidos, através de um ficheiro definido para o efeito. Como parâmetros para o arranque de um servidor são indicados o esquema de divisão dos bits dos identificadores globais dos recursos, isto é, o número de bits reservados para a identificação do *cluster* e para o número de série de cada nó de um *cluster*, a identificação do *cluster*, o número de série do nó e o porto UDP usado na difusão de pedidos, conforme se ilustra na figura 7.2.

Ainda relativamente ao funcionamento do sistema R_OCl, resta a instalação, opcional, do serviço de reencaminhamento, por forma a garantir conectividade entre recursos criados em *subclusters* distintos, no seio de um dado *cluster* heterogéneo. O arranque deste serviço faz-se, simplesmente, mediante a execução do comando `rocl_bridged`, sem necessidade



(a) *Cluster* DI-UMinho (b) *Cluster* ESTiG-IPB

Figura 7.2: Arranque de servidores de directório.

de qualquer parametrização.

Refira-se ainda que, tanto a biblioteca R_OCl como o comando `rocl_dird` pressupõem a definição da variável de ambiente `RoCl_SYS` em todos os nós, a qual representa o directório onde residem os vários programas e ficheiros que compõem o sistema R_OCl. Obviamente, a administração de um *cluster* será mais simples se este directório for partilhado por NFS, por exemplo.

Para além da configuração do sistema R_OCl, o funcionamento da biblioteca $m_\varepsilon\mu$ requer ainda a disponibilização do serviço `rsh`, por forma a permitir a execução remota de programas de sistema – `meu_launch`, `meu_mbox` e `meu_gather` – no seio de um *cluster*. Será também necessário definir a variável de ambiente `meu_SYS`, que corresponderá ao directório onde estarão instalados esses programas, cuja finalidade será posteriormente analisada.

Os programas ou módulos que compõem as aplicações paralelas/distribuídas deverão, obviamente, estar instaladas em cada máquina do *cluster*, num directório dado a conhecer através da variável de ambiente `meu_MYDIR`, no contexto de cada utilizador.

7.1.2 Representação de recursos físicos

Os recursos físicos de um *cluster* são especificados num ficheiro de texto, através de uma linguagem simples, cuja sintaxe se apresenta na figura 7.3.

Com base nessa linguagem, a hierarquia de recursos apresentada na figura 4.3 pode ser facilmente especificada, conforme se constata pela figura 7.4. Essa especificação é processada pelo programa `meu_phyparse`, o qual se encarrega de registar, no directório R_OCl,

```
Especificação  ::  (comentário | Domínio)+
Domínio        ::  [etiqueta · ':'] · nome · [Ascendente] · [Originais] · [Propriedades] · ';'
Ascendente     ::  '⟨' · etiqueta · '⟩'
Originais      ::  '(' · etiqueta · (',' · etiqueta)* · ')'
Propriedades   ::  '{' · Propriedade · (',' · Propriedade)* · '}'
Propriedade    ::  nome · ['=' · valor]
```

Figura 7.3: Sintaxe para especificação de um domínio físico.

cada um dos domínios especificados.

Um domínio físico é registado como um recurso R_OCl, cujos atributos correspondem às propriedades definidas para esse domínio, juntamente com o seu nome e o identificador do ascendente sob o qual ele é criado. Assim, o domínio *Nó* A_1, especificado na figura 7.4, por exemplo, será constituído, ao nível do R_OCl, como um recurso $R = \{\langle$"*tipo*","*domínio*"$\rangle, \langle$"*nome*","*Nó A*1"$\rangle, \langle$"*ascend*", 1003$\rangle, \langle$"CPU", 2$\rangle, \langle$"Mem", 512$\rangle\}$, assumindo que 1003 foi o identificador global atribuído pelo directório ao domínio *Subcluster Dual PIII*.

```
/* organizadores de nós */
Cl: Cluster {FastEthernet};
Sub1: "Subcluster Dual PIII" <Cl> {Myrinet};
Sub2: "Subcluster Quad Xeon" <Cl> {Myrinet, Gigabit};
Sub3: "Subcluster Dual Athlon" <Cl> {Gigabit};
"Subcluster Myrinet" <Cl> (Sub1, Sub2);

/* nós computacionais */
"Nó A1" <Sub1> {CPU=2, Mem=512};
"Nó B1" <Sub2> {CPU=4, Mem=1024};
"Nó C1" <Sub3> {CPU=2, Mem=512};
....
```

Figura 7.4: Especificação dos recursos físicos de um *cluster*.

Na especificação dos recursos, o administrador usa etiquetas, a fim de encadear ascendentes e descendentes e originais e pseudónimos, as quais são convertidas em identificadores globais pelo programa `meu_phyparse`. Em relação à especificação da figura 7.4, na operação de registo do domínio raíz (o domínio *Cluster*), o R_OCl devolve um identificador, o qual é associado à etiqueta *Cl*. Deste modo, no registo do domínio *Subcluster Dual PIII*, já poderá ser construído, pelo programa `meu_phyparse`, um atributo R_OCl, que inclua o identificador global do ascendente desse domínio. Um domínio só poderá ser registado

quando forem conhecidas todas as etiquetas usadas na sua especificação.

O programa `meu_phyparse`, executado a partir de um dado nó do *cluster*, fará o registo local (em relação a esse nó) de todos os domínios especificados. No entanto, se o administrador entender que os domínios físicos deverão ser dispersos pelos vários nós do *cluster*, poderá ser acrescentada, à especificação de cada domínio, o nome da máquina onde o registo deverá ter lugar. Nesse caso, o programa `meu_phyparse` executará, no nó indicado, via `rsh`, o próprio programa `meu_phyparse`, passando-lhe como argumento a informação necessária para o registo em causa e obtendo como resposta o identificador atribuído pelo servidor de directório remoto.

O registo de recursos R_OCl obriga à criação de um contexto, automaticamente criado no arranque da biblioteca. Desta forma, os recursos registados pelo programa `meu_phyparse` terão associado um contexto e, do ponto de vista do R_OCl, serão tratados como quaisquer outros recursos. Desta forma, os servidores de directório R_OCl procederão à eliminação desses recursos se os respectivos contextos não estiverem activos. Por conseguinte, o programa `meu_phyparse` mantém um fio-de-execução permanentemente bloqueado, aguardando eventuais mensagens enviadas a esses recursos. Obviamente, e conforme já referido, quaisquer mensagens recebidas serão desprezadas.

7.2 Navegação na hierarquia de recursos

A hierarquia de recursos, iniciada com a especificação dos recursos físicos, conforme apresentado na secção anterior, e posteriormente ampliada com a criação de recursos lógicos por parte das aplicações e do sistema de suporte à execução do $m_\varepsilon\mu$, poderá ser manipulada com base nas primitivas apresentadas na tabela 7.1. Essencialmente, estão em causa primitivas que permitem a pesquisa, caracterização, reorganização e selecção de recursos $m_\varepsilon\mu$, os quais estão organizados hierarquicamente e cuja informação é armazenada no serviço de directório do R_OCl.

7.2.1 Pesquisas imediatas

Excluindo as primitivas `meu_get_properties`, `meu_lookup` e `meu_lookup_aggreg`, as restantes primitivas constantes da tabela 7.1 traduzem-se em simples interrogações, a desencadear à custa de evocações às primitivas `rocl_query` e `rocl_query_start`.

A primitiva `meu_get_mygid`, que permite às tarefas, as únicas entidades activas do sistema $m_\varepsilon\mu$, obterem os seus identificadores, nem sequer implica uma operação de pesquisa. Na verdade, trata-se da consulta de valores mantidos em memória para cada um dos fios-de-execução usados para implementar as tarefas. Estes valores são armazenados com base

Tabela 7.1: Primitivas $m_\varepsilon\mu$ para navegação na hierarquia de recursos.

```
int meu_get_mygid()
int meu_get_creator(int gid)
int meu_get_ancestor(int gid)
meu_stat_t meu_get_info(int gid, char *name, enum meu_entt *enttype)
meu_gidl_t * meu_get_originals(int gid)
meu_gidl_t * meu_get_descendants(int gid)
meu_gidl_t * meu_get_aliases(int gid)
meu_prpl_t * meu_get_properties(int gid)
int meu_lookup(int gid, const char *name, enum meu_entt enttype,
                const meu_prpl_t *prps)
int meu_lookup_aggreg(int gid, const meu_prpl_t *prps1,
                      const meu_prpl_t *prps2, const char *name,
                      const meu_prpl_t *prps)
meu_set_scope(enum meu_scope scope)
```

nas funcionalidades POSIX que suportam a definição de informação específica para cada fio-de-execução.

As primitivas `meu_get_creator`, `meu_get_ancestor` e `meu_get_info`, a partir dos identificadores dos recursos, permitem obter informação elementar sobre estes, nomeadamente o identificador da tarefa criadora, o identificador do ascendente ou o nome e o tipo de entidade, associados a um recurso. A informação em causa, para cada uma destas primitivas, pode ser obtida, facilmente, por via de uma evocação da primitiva `rocl_query`.

No caso da primitiva `meu_get_originals`, apesar de ser devolvido um conjunto de identificadores, cuja manipulação se faz através das primitivas constantes na tabela 7.2, também é evocada uma única vez a primitiva `rocl_query`. De facto, a lista de originais de um recurso é armazenada como um único atributo R_OCl.

Tabela 7.2: Primitivas $m_\varepsilon\mu$ para manipulação de listas de identificadores.

```
meu_gidl_t * meu_new_gidl(int max_gids)
int meu_add_gid(meu_gidl_t *gids, int gid)
int meu_nget_gid(const meu_gidl_t *gids, int lpos, int **gid)
meu_destroy_gidl(meu_gidl_t *gids)
```

Já no que diz respeito às primitivas `meu_get_descendants` e `meu_get_aliases`, será necessário desencadear pesquisas para obtenção de múltiplas resultados, visto que os recursos não armazenam informação sobre os seus descendentes ou pseudónimos. Na verdade,

obtêm-se os descendentes ou os pseudónimos de um recurso procurando no directório todos os recursos que têm como ascendente ou original esse recurso. Os múltiplos resultados (identificadores de descendentes ou pseudónimos) eventualmente devolvidos pelo R_OCl, através da utilização das primitivas `rocl_query_start` e `rocl_query_next`, são compilados pelo sistema $m_\varepsilon\mu$ e devolvidos numa lista.

7.2.2 Pesquisas baseadas em propriedades

As primitivas `meu_get_properties` e `meu_lookup` exigem o cálculo das propriedades de um ou mais recursos, pelo que o seu funcionamento é mais complexo e mais exigente do ponto de vista computacional.

Manipulação de propriedades

A manipulação das listas de propriedades associadas a cada um dos recursos $m_\varepsilon\mu$ faz-se através das primitivas apresentadas na tabela 7.3.

Tabela 7.3: Primitivas $m_\varepsilon\mu$ para manipulação de listas de propriedades.

```
meu_prpl_t * meu_new_prpl(int max_len)
int meu_add_prp(meu_prpl_t *prps, const char *name, const void *value,
                int len, enum meu_prp_ops op)
int * meu_get_prp(const meu_prpl_t *prps, const char *name, void **value,
                  int *len)
int * meu_nget_prp(const meu_prpl_t *prps, int lpos, char **name,
                   void **value, int *len)
meu_destroy_prpl(meu_prpl_t *prps)
```

As propriedades que, para além de um nome, incluem um valor diferente de *NULL*, isto é, propriedades usadas para estipular valores de características específicas dos recursos, podem ainda ter associada uma operação de acumulação, para que possam ser efectuados cálculos com essas propriedades – propriedades acumuláveis. Por exemplo, o mecanismo de sintetização deverá ser capaz de somar os valores das propriedades respeitantes ao número de processadores de cada nó computacional, por forma a se poder determinar o número total de processadores disponíveis num *cluster*.

As propriedades $m_\varepsilon\mu$ são traduzidas em atributos R_OCl da seguinte forma: o nome de uma propriedade é traduzido directamente no nome do atributo correspondente, enquanto que o valor de uma propriedade juntamente com o índice da operação de acumulação associada são traduzidos no valor do atributo. O índice da operação de acumulação, dado

que ocupa um número de bits bem conhecido, é tratado como um campo suplementar do valor do atributo.

A lista de operações de acumulação suportadas pelo $m_{\varepsilon}\mu$, para o cálculo de propriedades, pode ser alargada, exigindo, no entanto, a recompilação da biblioteca. Cada operação de acumulação é implementada como uma função que aceita dois valores e produz um resultado, à qual é associado um índice correspondente a uma entrada numa tabela de operações de acumulação. Isto deve-se à necessidade de determinar, de forma inequívoca, a operação a desencadear, em quaisquer módulos da aplicação que, dispersos pelos nós do *cluster*, obtenham propriedades de vários recursos, a partir do directório. Uma outra solução passaria pelo armazenamento da descrição da operação de acumulação (o código fonte, por exemplo), juntamente com a demais informação da propriedade, no directório R_OCl.

Localização de uma entidade

A implementação da primitiva `meu_lookup` tem em consideração as várias possibilidades de caracterização da entidade a localizar. Se apenas for especificada informação básica sobre a entidade a localizar (o nome e o tipo de entidade), o algoritmo é relativamente simples; a partir da entidade indicada, percorre-se a subárvore correspondente. No caso de serem especificadas propriedades, o algoritmo terá também que ir recalculando as propriedades das entidades que vai atravessando, sem entrar em consideração com as propriedades sintetizadas. Refira-se que, por definição, a primitiva `meu_lookup` deverá garantir que todos os descendentes da entidade devolvida possuem as propriedades indicadas como parâmetro na evocação da primitiva.

A figura 7.5 apresenta o algoritmo correspondente à primitiva `meu_lookup`. Quando se conclui que algumas das propriedades especificadas na evocação da primitiva se verificam num dado nó da árvore de entidades em análise, impõe-se que, em relação aos descendentes, se faça unicamente a verificação das propriedades em falta. Desta forma, evitam-se algumas operações de cálculo de propriedades, nomeadamente propriedades herdadas, acelerando o processo de localização. Refira-se que o algoritmo recursivo apresentado assume que na evocação inicial o parâmetro *fromwithin* tem o valor *FALSE*.

O processo de localização de entidades obriga a que, para além da operação de acumulação, seja definida uma operação de verificação de satisfação, a qual terá por finalidade a implementação do operador $\subseteq$, utilizado no algoritmo da figura 7.5; um conjunto de propriedades está contido noutro se quaisquer propriedades acumuláveis do primeiro forem satisfeitas por propriedades do segundo e se para qualquer propriedade não acumulável do primeiro existir uma exactamente igual, isto é, com nome e valor equivalentes bit a bit, no segundo. A execução iterativa deste algoritmo requer ainda que seja definida uma operação de acu-

$\underline{Lk(x, info, prps, fromwithin)}$
$\quad if(prps \neq NULL)$
$\quad\quad foreach\ y \in originals(x)$
$\quad\quad\quad p \leftarrow p \cup OP(y) \cup IP(y);$
$\quad\quad if(fromwithin = FALSE)$
$\quad\quad\quad p \leftarrow \bigtriangledown(p \cup OP(x) \cup IP(x));$
$\quad\quad else$
$\quad\quad\quad p \leftarrow \bigtriangledown(p \cup OP(x));$
$\quad if(info = inf(x)) \wedge (prps \subseteq p)$
$\quad\quad$ //e.g. $p = \{\langle$"a"$, 3\rangle_+$, "b", "c"$\} \wedge q = \{\langle$"a"$, 1\rangle_+$, "c"$\} \Rightarrow q \subseteq p$
$\quad\quad return(x);$
$\quad else$
$\quad\quad foreach\ y \in (descendants(x) \cup originals(x))$
$\quad\quad\quad if((z \leftarrow Lk(y, info, prps \setminus p, TRUE)) \neq -1)$
$\quad\quad\quad\quad$ //e.g. $p = \{\langle$"a"$, 3\rangle_+$, "b", "c"$\} \wedge q = \{\langle$"a"$, 1\rangle_+$, "c", "d"$\} \Rightarrow p \setminus q = \{\langle$"$a$"$, 2\rangle_+$, "$b$"$\}$
$\quad\quad\quad\quad return(z);$
$\quad return(-1);$

Figura 7.5: Localização de uma entidade.

mulação inversa, de maneira a implementar a subtracção de conjuntos de propriedades (operador $\setminus$).

7.2.3 Domínios agregadores

A primitiva `meu_lookup_aggreg` corresponde a uma forma especial de localização de entidades. Em vez de procurar uma única entidade que, à custa de propriedades directamente associadas ou herdadas, cumpra determinados requisitos, como acontece com a primitiva `meu_lookup`, a primitiva `meu_lookup_aggreg` tenta descobrir um conjunto de entidades que, tendo também em conta a sintetização, cumpram dois níveis de requisitos: verificação de propriedades específicas em cada nó da hierarquia e verificação de propriedades acumuladas no conjunto de todas as entidades.

O funcionamento da primitiva `meu_lookup_aggreg`, descrito pelo algoritmo da figura 7.6 (operação $Lk_{\uplus}$), envolve dois passos. Em primeiro lugar, é produzida uma lista de entidades, directa ou indirectamente descendentes da entidade indicada como parâmetro, que cumprem os requisitos da primeira lista de propriedades especificada, isto é, que garantem todas as propriedades indicadas nas várias subárvores que representam (operação $Lk_{\{\}}$). Em segundo lugar, é determinada a lista completa de propriedades que as entidades descobertas no passo anterior conseguem reunir por força dos mecanismos de partilha, sintetização e herança. Se estas propriedades satisfizerem todas aquelas correspondentes

$\underline{Lk_{\uplus}(x, p_1, p_2, name, prps)}$
$l \leftarrow Lk_{\{\}}(x, p_1 \setminus IP(x))$;
$foreach\ y \in l$
$p_3 \leftarrow p_3 \cup OP(y) \cup IP(y) \cup SiP(y) \cup ShP(y)$;
$p_3 \leftarrow \bigtriangledown(p_3)$;
$if(p_2 \subseteq p_3) \wedge (l \neq \{\})$
$if(\#l = 1)$
$return(elem(l))$;
$else$
$return(create\_alias(x, l, name, prps))$;
$else$
$return(-1)$;

$\underline{Lk_{\{\}}(x, p)}$
$foreach\ y \in originals(x)$
$p_1 \leftarrow p_1 \cup OP(y) \cup IP(y)$;
$p_1 \leftarrow \bigtriangledown(p_1 \cup OP(x))$;
$if(p \subseteq p_1)$
$return(\{x\})$;
$else$
$foreach\ y \in descendants(x)$
$l \leftarrow l \cup Lk_{\{\}}(y, p \setminus p_1)$;
$if(l = descendants(x))$
$return(\{x\})$;
$else$
$return(l)$;

Figura 7.6: Localização de entidades com eventual agregação.

ao segundo parâmetro da primitiva, então a operação terá sucesso.

Dado que a primitiva `meu_lookup_aggreg` devolve sempre um único identificador de recurso, no caso de o primeiro passo do algoritmo detectar múltiplas entidades, é criado um domínio pseudónimo, sendo-lhe atribuídos o nome e lista de propriedades indicados.

7.2.4 Âmbito das pesquisas

As pesquisas no directório R_OCl desencadeadas pela biblioteca $m_{\varepsilon}\mu$ deverão tirar partido da organização por níveis desse mesmo directório (ver secção 5.5.1). Na verdade, se qualquer pesquisa $m_{\varepsilon}\mu$ obrigar a uma pesquisa R_OCl ao nível `MULTICLUSTER`, por exemplo, o sistema terá um desempenho bastante pobre.

Dado que o $m_{\varepsilon}\mu$ não tem forma de, por si só, delimitar o âmbito de uma dada pesquisa,

pelo facto de os pseudónimos estabelecerem ligações virtuais entre quaisquer pontos de uma árvore de recursos, a solução adoptada foi a de deixar tal tarefa ao programador. Assim, é disponibilizada a primitiva `meu_set_scope`, que permite que o programador defina o nível de abrangência das pesquisas despoletadas pelo $m_{\varepsilon}\mu$, de acordo com os níveis definidos para o directório R_OCl (`LOCAL`, `SUBCLUSTER`, `CLUSTER`, `MULTICLUSTER`). Por omissão, o $m_{\varepsilon}\mu$ desencadeará pesquisas R_OCl ao nível `CLUSTER`.

7.3 Criação de entidades lógicas

As entidades lógicas são criadas através da evocação de primitivas específicas ou, em casos particulares, de forma automática pelo sistema $m_{\varepsilon}\mu$. A tabela 7.4 apresenta as primitivas que o programador tem ao seu dispor para a criação de entidades lógicas.

Tabela 7.4: Primitivas $m_{\varepsilon}\mu$ para criação de recursos lógicos.

```
int meu_init()
meu_set_acl(const meu_acl_t *acl)
int meu_new_domain(int gid, const char *name, const meu_prpl_t *prps)
int meu_new_operon(int gid, const char *name, const meu_prpl_t *prps,
                   const char *path, const meu_argl_t *args)
int meu_new_task(int gid, const char *name, const meu_prpl_t *prps,
                 const char *func, const void *args, int size)
int meu_new_block(int gid, const char *name, const meu_prpl_t *prps,
                  int size)
int meu_new_gather(int gid, const char *name, const meu_prpl_t *prps)
int meu_new_mbox(int gid, const char *name, const meu_prpl_t *prps)
int meu_new_alias(int gid, const char *name, const meu_prpl_t *prps,
                  enum meu_entt enttype, const meu_gidl_t *gids)
```

7.3.1 Integração na hierarquia

Na criação de uma entidade lógica, o programador indica o nó – um identificador – da hierarquia de recursos onde pretende inserir a entidade, o nome que lhe pretende atribuir e a lista de propriedades associadas. Depois de verificar se a entidade pode ser criada/inserida no ponto indicado, o sistema $m_{\varepsilon}\mu$ regista um recurso R_OCl ao qual são associados atributos para especificar o tipo de entidade (domínio, operão, etc.), o nome, o ascendente e cada uma das propriedades especificadas. O identificador da tarefa $m_{\varepsilon}\mu$ onde é evocada a primitiva que cria a entidade também é armazenado como um atributo R_OCl. O identificador atribuído pelo R_OCl ao recurso em causa é devolvido pela primitiva $m_{\varepsilon}\mu$.

A organização dos recursos numa árvore é implicitamente assegurada pela inclusão do identificador do ascendente na informação armazenada para cada recurso (lógico ou físico). O único recurso que não possui ascendente é o domínio que se encontra na raíz da árvore de recursos físicos. Este domínio físico pode ser especificado nas primitivas $m_{\varepsilon}\mu$ através da palavra chave `ROOT`; o sistema $m_{\varepsilon}\mu$ encarregar-se-á de obter o identificador da entidade cujo ascendente é *NULL*.

As restrições à criação de entidades derivam das regras de encadeamento referidas na secção 3.2.6; tendo em conta o tipo de entidade que se pretende criar e o identificador indicado para ascendente dessa entidade, o $m_{\varepsilon}\mu$ verifica se a hierarquia daí resultante faz ou não sentido.

Para além das limitações relativas ao encadeamento de entidades, há ainda a considerar as restrições relacionadas com questões de permissões (ver secção 3.4.3). Com o intuito de dar resposta a este requisito, foi introduzida a primitiva `meu_set_acl`, que possibilita especificar a lista de controlo de acesso a utilizar em todas as operações subsequentes de criação de recursos. Esta lista é utilizada na evocação da primitiva `rocl_register`, por parte do sistema $m_{\varepsilon}\mu$, fixando, para cada entidade (domínio, operão, etc.), o conjunto de utilizadores com permissão para criar descendentes ou pseudónimos.

Se na criação de uma entidade, o programador indicar como ascendente ou como originais entidades que não está autorizado a utilizar, tal situação será automaticamente detectada pelo R_OCl (ver secção 5.3.4), dado que a verificação das restrições de encadeamento envolve pesquisas relativas a essas entidades. Desta forma, o $m_{\varepsilon}\mu$ não requer funcionalidade para a interpretação das listas de controlo de acesso associadas aos recursos. Aliás, o R_OCl não permite conhecer a lista de controlo de acesso associada a um recurso em particular.

7.3.2 Domínios

A criação de domínios, através da primitiva `meu_new_domain`, corresponde à operação de criação mais simples do $m_{\varepsilon}\mu$, dado que não impõe grandes restrições do ponto de vista do encadeamento numa hierarquia existente. O sistema terá apenas que excluir a criação de domínios sob entidades que não sejam domínios ou operões, bastando, para tal, avaliar o tipo da entidade representada pelo identificador indicado na primitiva.

Em concreto, um domínio é simplesmente um recurso R_OCl criado no âmbito do contexto R_OCl associado ao operão onde está inserida a tarefa a partir da qual é evocada a primitiva `meu_new_domain`. Por forma a distinguir os domínios lógicos dos físicos, ao recurso R_OCl correspondente a um domínio lógico é associado um atributo $\langle tipo, domínio\_lógico \rangle$, enquanto que, no caso de um domínio físico, é usado um atributo $\langle tipo, domínio \rangle$, como referido anteriormente.

7.3.3 Operões, tarefas e blocos de memória

A criação de operões, tarefas e blocos de memória, através das primitivas `meu_new_operon`, `meu_new_task` e `meu_new_block`, respectivamente, exige cuidados especiais devido às restrições de encadeamento, que conduzem à necessidade de proceder à descoberta automática de entidades alvo.

Os operões obrigam à presença de um domínio físico representativo de uma máquina na sua cadeia de pseudo-ascendência e as tarefas e os blocos de memória à presença de um operão. A verificação destas condições seria fácil se os operões fossem obrigatoriamente criados como descendentes directos de domínios físicos e as tarefas e os blocos de memória como descendentes directos de operões. No entanto, definiu-se que estas entidades podiam ser criadas indicando, como parâmetro da primitiva de criação, o identificador de um domínio que, apesar de não corresponder a um domínio físico (no caso dos operões) ou a um operão (no caso das tarefas e dos blocos de memória), permite chegar à entidade adequada por via do seu ascendente, dos seus descendentes ou dos seus originais.

Deste modo, a criação de um operão, tarefa ou bloco de memória obriga à descoberta da entidade alvo, sob a qual é criado o recurso em causa. Após a criação efectiva do recurso (operão, tarefa ou bloco) é ainda criado um pseudónimo, para esse recurso, sob o domínio a partir de onde se deu início à descoberta da entidade alvo. A figura 7.7 apresenta, resumidamente, o algoritmo usado na operação de criação de um operão, que é similar ao usado para as tarefas e os blocos e memória.

A operação *launch* referida no algoritmo diz respeito ao arranque do programa indicado na primitiva `meu_new_operon`. O $m_{\varepsilon}\mu$ utiliza o serviço `rsh` para executar remotamente o comando `meu_launch`, o qual será responsável por efectivamente colocar em execução o programa pretendido.

No caso das tarefas e blocos de memória, a operação *searchHost* dará lugar à operação *searchOperon*, que procurará uma entidade do tipo operão em vez de um domínio que integre a propriedade $\langle "CPU", 1 \rangle$. Também a operação *launch* é substituída pelo envio de uma mensagem ao operão destino, no seio do qual a tarefa ou bloco são criados.

No caso dos operões e das tarefas, há ainda a realçar o facto de a sua criação poder ocorrer sem a evocação de uma primitiva específica. De facto, no arranque de aplicações, durante a execução da primitiva `meu_init`, o sistema $m_{\varepsilon}\mu$ procede à criação automática de um operão e de uma tarefa. O operão é criado sob o domínio que representa a máquina onde a aplicação é colocada em execução e terá como nome o nome do executável correspondente. A tarefa será criada no âmbito desse operão e terá como nome *main*.

```
newOperon(where, name, prog)
    if((target ← searchHost(where)) ≠ NULL)
        if(target ∈ (originals(where) ∪ where))
            return(launch(target, name, prog));
        else
            orig ← launch(target, prog, prog);
            return(newAlias(where, name, OPERON, {orig}));
    else
        return(NULL);

searchHost(x, visited)
    l ← ({x} ∪ {ancestor(x)} \ {ROOT} ∪ originals(x) ∪ descendants(x)) \ {visited};
    foreach y ∈ l
        visited ← visited ∪ {y};
        if({⟨"CPU", 1⟩} ⊆ OP(y))
            return(y);
        else
            if(y ≠ x)
                if((res ← searchHost(y, visited)) ≠ NULL)
                    return(res);
    return(NULL);
```

Figura 7.7: Criação de um operão.

7.3.4 Caixas postal e agregadores de memória

As caixas postal e os agregadores de memória, tal como os operões, exigem a presença de um domínio físico representativo de um sistema de computação nas suas cadeias de pseudo-ascendência. Além disso, as primitivas `meu_new_mbox` e `meu_new_gather` permitem analogamente indicar um domínio diferente daquele que corresponde ao sistema de computação alvo. No entanto, a criação destas entidades compreende dois cenários distintos, dependendo da existência, ou não, de um operão para suportar os elementos computacionais envolvidos.

Em relação ao algoritmo apresentado na figura 7.7, na criação de caixas postal e agregadores começa-se por procurar um operão (operação *searchOperon*) e, no caso de insucesso, tenta-se encontrar um sistema computacional (operação *searchHost*).

No caso de ser encontrado um operão, a criação destas entidades envolve, tal como as tarefas e os blocos de memória, o envio de uma mensagem a esse operão. No caso contrário, se existir um sistema computacional, o $m_{\varepsilon}\mu$ procede ao arranque remoto de um dos programas de sistema `meu_mbox` ou `meu_gather`. Estes programas criam um contexto R_OCl para suportar os elementos computacionais necessários às caixas postal e aos agregadores de memória, respectivamente.

7.3.5 Pseudónimos

Os pseudónimos, explícitos se forem criados através da primitiva `meu_new_alias` e implícitos se forem criados por iniciativa do sistema $m_{\varepsilon}\mu$, são, tal como os domínios, meros recursos R_OCl. A sua criação (explícita) requer, no entanto, a verificação das restrições de encadeamento, visto que, tal como outro recurso qualquer, são integrados na hierarquia de recursos $m_{\varepsilon}\mu$.

É necessário ainda ter em conta que a lista de originais indicada na primitiva apenas poderá ser um conjunto não singular se se pretender criar um pseudónimo domínio. Além disso, quando o pseudónimo a criar não for um domínio, o original terá que ser uma entidade do mesmo tipo daquela a criar.

7.4 Envio e recepção de mensagens

Os procedimentos de envio e recepção de mensagens no $m_{\varepsilon}\mu$ são semelhantes aos descritos para o R_OCl. Tal como no R_OCl, estão disponíveis primitivas para a manipulação de tampões e primitivas para o envio e recepção (ver tabela 7.5). As primeiras – `meu_bfget`, `meu_bfret`, `meu_bftoret` e `meu_bfstat` – constituem meras evocações das equivalentes R_OCl, pelo que a sua apresentação é aqui escusada. As segundas apresentam diferenças subtis relativamente às primitivas disponibilizadas no R_OCl, nomeadamente no que diz respeito à especificação de origens e destinos de mensagens.

Tabela 7.5: Primitivas $m_{\varepsilon}\mu$ para envio e recepção de mensagens.

```
void * meu_bfget(int len)
meu_bfret(void *ptr)
meu_bftoret(void *ptr)
int meu_bfstat(const void *ptr, int timeout)
```

```
int meu_send(meu_addr_t from, meu_addr_t to, int tag, const void *ptr,
             int len)
int meu_recv(meu_addr_t as, meu_addr_t from, int tag, void **ptr,
             meu_addr_t *afrom, int *atag, int *alen, int timeout)
```

7.4.1 Origem e destino de uma mensagem

No $m_{\varepsilon}\mu$, a origem e o destino de uma mensagem não são simples identificadores de recursos. Devido às abstracções de alto-nível definidas, os parâmetros `from`, `to` e `as` das primitivas `meu_send` e `meu_recv` recorrem a um tipo de dados próprio, de forma a permitir

alternativas.

No caso do envio, o parâmetro `from` poderá assumir um valor nulo, significando que quem envia a mensagem é a própria tarefa onde a primitiva é evocada, um identificador de uma caixa postal, para quando se pretende enviar uma mensagem sem dar a conhecer o identificador da tarefa envolvida, ou um tuplo composto por um identificador de um domínio e um identificador de membro, para o envio de mensagens contextualizadas. Quanto ao parâmetro `to`, o destino da mensagem, este poderá ser um identificador, correspondente a uma tarefa, a um operão, a um domínio ou a uma caixa postal, ou um tuplo, correspondente a um membro de um domínio (domínio, operão, uma tarefa ou caixa postal).

No caso da recepção, o parâmetro `as`, usado para indicar o repositório a partir de onde deve ser obtida a mensagem, poderá ser nulo, significando que a mensagem deverá ser obtida a partir do repositório associado à tarefa que evoca a primitiva de recepção, ou corresponder a um identificador, significando que a mensagem deverá ser obtida a partir do repositório do operão onde a tarefa foi criada ou a partir do repositório de uma caixa postal qualquer. Quanto ao parâmetro `from`, que permite especificar a origem da mensagem, poderá ser nulo, indicando, obviamente, que se pretende uma qualquer mensagem, independentemente da sua origem, ou então deverá corresponder a um identificador (de uma tarefa ou caixa postal) ou a um tuplo, em concordância com o parâmetro `from` da primitiva de envio.

No que diz respeito à correspondência entre as primitivas de envio/recepção do $m_{\varepsilon}\mu$ e as equivalentes do R_OCl, são necessárias algumas observações adicionais. Os parâmetros `from`, `to` e `as` das primitivas $m_{\varepsilon}\mu$ são convertidos nos parâmetros `ogid` e `dgid` das primitivas R_OCl, os quais deverão ser valores inteiros. Assim, valores nulos e tuplos, válidos no $m_{\varepsilon}\mu$, serão convertidos da seguinte forma:

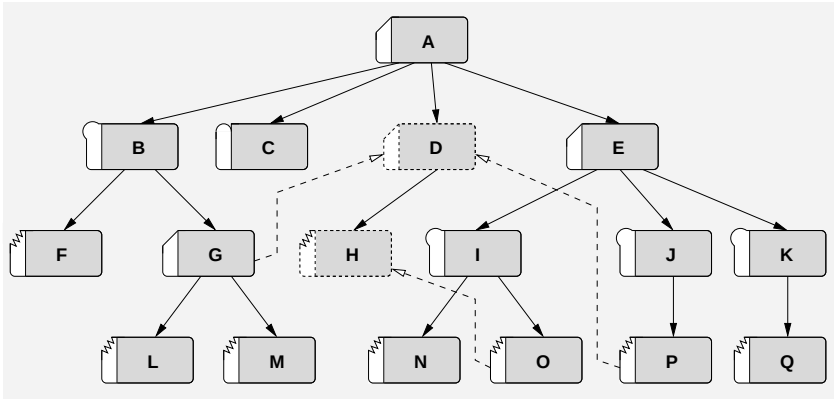
- um valor nulo do parâmetro `from` da primitiva de envio ou do parâmetro `as` da primitiva de recepção é convertido no identificador R_OCl da tarefa envolvida;
- um valor nulo do parâmetro `from` da primitiva de recepção é convertido num valor inteiro negativo, que no âmbito do R_OCl significa qualquer recurso;
- um tuplo indicado no parâmetro `from` de qualquer uma das primitivas é convertido, por via da concatenação, num valor inteiro de 64bits[1].
- um tuplo indicado no parâmetro `to` da primitiva de envio é substituído, automaticamente, pelo identificador do recurso a que efectivamente diz respeito.

[1]Apesar de os identificadores R_OCl serem inteiros de 32bits, o campo origem de uma mensagem comporta 64bits, para suportar as mensagens contextualizadas do $m_{\varepsilon}\mu$.

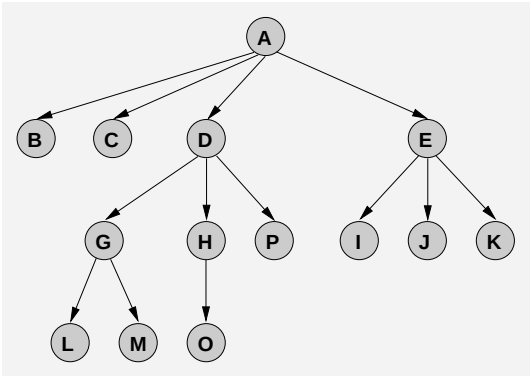
7.4.2 Múltiplas entregas

Quando o destino especificado na primitiva de envio corresponde a um domínio, o sistema $m_{\varepsilon}\mu$ faz chegar uma cópia da mensagem a cada um dos membros desse domínio, ou seja, a cada um dos seus descendentes ou originais (ver secção 4.1.2). Esta funcionalidade tira partido, como é desejável, da difusão selectiva do R_OCl.

Deste modo, sempre que é criado um recurso lógico que não um bloco de memória ou um agregador, se o seu ascendente for um domínio lógico, então é estabelecida uma relação de filiação, ao nível do R_OCl, através da primitiva `rocl_set_rel`. Adicionalmente, se estiver em causa a criação de um pseudónimo, também são estabelecidas relações de filiação com cada um dos originais indicados.



(a)



(b)

Figura 7.8: Relações R_OCl para uma hierarquia $m_{\varepsilon}\mu$.

Tomando como exemplo a hierarquia de recursos $m_{\varepsilon}\mu$ apresentada na figura 7.8(a), as relações de filiação estabelecidas de forma automática são as apresentadas na figura 7.8(b). Com base no esquema das relações R_OCl é fácil constatar que uma mensagem enviada ao

domínio A terá como destinos finais a caixa postal C, os operões B, I, J e K e as tarefas L, M, O e P.

7.5 Acesso à memória global

O acesso à memória global no $m_{\varepsilon}\mu$ tira partido das primitivas de escrita e leitura remota do R_OCl. No entanto, devido às abstracções de alto-nível que o $m_{\varepsilon}\mu$ introduz, o interface de programação exigiu novas primitivas, para garantir a funcionalidade exposta na secção 4.1.3. A tabela 7.6 apresenta as primitivas que, juntamente com as anteriormente apresentadas `meu_new_block` e `meu_new_gather`, permitem a manipulação de memória global.

Tabela 7.6: Primitivas $m_{\varepsilon}\mu$ para manipulação de memória global.

```
int meu_add_block(int bgid, int start, int length, int ggid, int offset)
void * meu_new_ref(int ggid, int start, int length)
int meu_get(void *ptr, int start, int length)
int meu_put(void *ptr, int start, int length)
int meu_lock(void *ptr, int start, int length)
int meu_unlock(void *ptr, int start, int length)
int meu_free_ref(void *ptr)
```

7.5.1 Funcionamento básico

Os blocos de memória e os agregadores são criados através das primitivas `meu_new_block` e `meu_new_gather`, respectivamente. A junção de vários blocos de memória, criando a ilusão de uma zona de memória global, faz-se através da adição de cada um dos blocos a um agregador, através da primitiva `meu_add_block`.

A adição de um bloco a um agregador corresponde à criação de um pseudónimo sob o agregador envolvido. A este pseudónimo são associados, como propriedades, os valores dos parâmetros `start`, `length` e `offset`, que especificam a fracção do bloco efectivamente integrada no agregador e a posição dessa fracção no espaço de endereçamento global. Além da criação do pseudónimo, é enviada uma mensagem ao agregador, para que este também armazene tal informação numa estrutura de dados própria. Desta forma, a criação dos blocos pseudónimo sob o agregador destina-se, unicamente, a integrar a memória global na hierarquia de recursos $m_{\varepsilon}\mu$, permitindo a utilização das primitivas genéricas de manipulação de recursos.

A evocação da primitiva `meu_new_ref` é o primeiro passo para um programa conseguir efectuar acessos a uma zona de memória global previamente constituída. Esta primitiva envia uma mensagem ao agregador, indicando qual a fracção do espaço de endereçamento global que se pretende aceder. O agregador responde com a lista dos identificadores dos blocos de memória envolvidos, juntamente com a indicação, para cada bloco, da fracção a considerar. Esta informação é mantida pelo sistema $m_{\varepsilon}\mu$, sendo devolvido ao programador apenas um apontador para uma zona de memória local.

Qualquer leitura ou escrita é efectuada por via da memória local; o apontador devolvido pela primitiva `meu_new_ref` deverá ser usado como se se tratasse de um apontador devolvido numa operação de alocação de memória dinâmica. A ligação com o espaço de endereçamento global faz-se mediante a evocação das primitivas `meu_get` e `meu_put`. Estas primitivas usam a informação devolvida pelo agregador (aquando da evocação da primitiva `meu_new_ref`) para, através das equivalentes R_OCl, manipularem, directamente, os tampões de memória associados aos recursos R_OCl correspondentes.

Para garantir o acesso em regime de exclusão mútua a uma dada zona da memória global, são ainda disponibilizadas as primitivas `meu_lock` e `meu_unlock`, que implementam um mecanismo básico de sincronização, com base num elemento centralizador, que é o agregador. A evocação de uma destas primitivas desencadeia o envio de uma mensagem ao agregador, onde são incluídos os parâmetros `start` e `length`, para que este possa gerir os acessos. A primitiva `meu_lock` apenas devolve o controlo ao programador após a recepção de uma mensagem vinda do agregador. Este responderá logo que a fracção de memória indicada seja desbloqueada.

7.5.2 Optimização dos acessos

De acordo com a fracção de memória indicada na primitiva `meu_new_ref` e tendo em conta os blocos que constituem o espaço de endereçamento total, existe um conjunto de optimizações possíveis, que permite melhorar o desempenho dos acessos à memória global.

Se a zona de memória especificada incluir um ou mais blocos criados em máquinas remotas, a primitiva `meu_new_ref` cria um tampão local e as primitivas `meu_get` e `meu_put` movimentam dados entre blocos remotos e esse tampão. Se, noutro extremo, a zona de memória especificada referenciar um único bloco criado no âmbito do operão no qual está inserida a tarefa que evoca a primitiva, então será devolvido um apontador para esse tampão (sem que haja a criação de qualquer tampão adicional) e as primitivas `meu_get` e `meu_put` não têm qualquer efeito.

Entre as duas situações limite acima referidas há a considerar três casos que ainda permitem efectuar optimizações: referência a vários blocos, todos eles criados no âmbito do

operão ao qual pertence a tarefa, referência a um único bloco criado num outro operão, mas ainda na máquina local, e referência a vários blocos, todos eles criados no âmbito da máquina local. No primeiro caso será necessário criar um tampão, de forma a criar a ilusão de um espaço de endereçamento contínuo, mas as primitivas `meu_get` e `meu_put` traduzir-se-ão em simples operações de cópia de memória (`memcpy`). No segundo caso, por via dos mecanismos IPC, é possível evitar a criação do tampão e as primitivas `meu_get` e `meu_put` não terão qualquer efeito. No terceiro caso, será necessário um tampão, pelos motivos apontados no primeiro caso, e as primitivas `meu_get` e `meu_put` corresponderão a acessos à memória partilhada entre processos.

7.6 Epílogo

As primitivas $m_{\varepsilon}\mu$ apresentadas neste capítulo constituem o principal elo de contacto entre o programador e a abordagem preconizada neste trabalho, para desenvolvimento de aplicações destinadas à execução no contexto de um *cluster* SMP multi-SAN.

Os procedimentos para a preparação inicial do ambiente de suporte à execução, os mecanismos e estratégias para navegação na hierarquia de recursos, a criação de entidades lógicas, o envio e recepção de mensagens e o acesso à memória global foram objecto de análise neste capítulo. Em particular, mostrou-se de que forma o R_OCl é explorado para implementar a funcionalidade de alto-nível do $m_{\varepsilon}\mu$.

Dado que o R_OCl foi especificamente desenvolvido para suporte às abstracções idealizados no $m_{\varepsilon}\mu$, a relativa facilidade com que algumas primitivas $m_{\varepsilon}\mu$ puderam ser concretizadas, com base na funcionalidade do R_OCl, é algo que já se esperava. De qualquer forma, a adequabilidade do R_OCl à implementação do $m_{\varepsilon}\mu$, ou seja, a compatibilidade entre os dois sistemas, ficou aqui realçada, comprovando, uma vez mais, a viabilidade da arquitectura proposta no capítulo 3 para a exploração de *clusters* SMP multi-SAN.

Capítulo 8

Avaliação de desempenho

A avaliação da biblioteca $m_{\varepsilon}\mu$ foi efectuada através de um conjunto de testes sintéticos, tendo em vista a obtenção de valores de desempenho relativos ao funcionamento de alguns componentes e do sistema como um todo.

Apenas foram avaliadas as funcionalidades que não dependem de decisões do programador, ou seja as que correspondem a evocações directas de primitivas R_OCl. A panóplia de primitivas que o $m_{\varepsilon}\mu$ oferece para organização de entidades lógicas e selecção de recursos físicos insere-se num modelo de programação próprio e, portanto, proceder à sua avaliação seria pouco proveitoso.

As experiências que conduziram aos resultados aqui apresentados foram realizadas num ambiente *cluster* constituído por 9 estações de trabalho, distribuídas por 3 *subclusters*, cujas características são as apresentadas na tabela 8.1.

Tabela 8.1: Características dos nós do *cluster* usado para avaliação de desempenho.

Subcluster	Nós	CPUs/nó	CPU	RAM	PCI	LAN	SAN
i686	4	2	PIII, 733MHz	512MB	64bits, 66MHz	Intel Ether Express Pro100	Myrinet LANai9
Athlon	4	2	Athlon, 1.8GHz	512MB	64bits, 66MHz	3Com 3c905C Tornado	SysKonnect 9821
Xeon	1	4	Xeon, 700MHz	1GB	64bits, 66MHz	Intel Ether Express Pro100	Myrinet LANai9 SysKonnect 9821

A interligação de nós é feita através de 3 comutadores: um comutador Myrinet M2M-SW16 (1.28+1.28Gbits/s), que interliga os nós dos *subclusters* i686 e Xeon, um comutador Gigabit Level One GSW-0801T, que interliga os nós dos *subclusters* Athlon e Xeon, e um comutador HP ProCurve 2224 (Fast Ethernet), ao qual estão ligados todos os nós. O

sistema operativo instalado é o Linux RedHat 9.0, com o *kernel* versão 2.40.20-9. Em relação às bibliotecas GM e M-VIA, são usadas as versões 2.0.8 e 1.2, respectivamente.

Para a realização de alguns testes, por questões de uniformização, foram instalados controladores Gigabit (SysKonnect 9821) nos nós do *subcluster* i686. Este *subcluster*, dotado de duas tecnologias de elevado desempenho, é aqui designado por i686$^+$.

Por uma questão de brevidade, os controladores Intel Ether Express Pro100, 3Com 3c905C Tornado e SysKonnect 9821 são designados, nas legendas de alguns gráficos, por *EE*, *3c* e *SK*, respectivamente.

8.1 Serviço de directório

O desempenho global do $m_\varepsilon\mu$ depende, em grande parte, do desempenho do serviço de directório, o que justificou o desenvolvimento de um pequeno programa de teste, em que sucessivas evocações da primitiva `meu_get_info` permitiram medir as taxas máximas de pesquisa suportadas.

Optou-se por efectuar pesquisas simples, que envolvem uma única resposta com um tamanho de 256bytes, por forma a eliminar a complexidade subjacente à navegação na hierarquia de recursos, que dependeria do tipo de aplicação e das opções do programador. As pesquisas consideradas equivalem às que o próprio sistema R_OCl desencadeia para descobrir endereços de comunicação associados a recursos.

8.1.1 Operação local

Nas situações mais favoráveis, as pesquisas no directório são resolvidas no próprio nó onde são desencadeadas, pelo que apenas o servidor local é contactado, não havendo lugar à troca de mensagens entre nós. O desempenho do serviço de directório, relativamente a operações locais, depende, basicamente, dos mecanismos de comunicação intra/interprocesso (IPC), do escalonamento de fios-de-execução e processos e do mecanismo de procura usado para localização de uma entrada na base de dados que o servidor mantém em memória.

A figura 8.1 apresenta as taxas de pesquisa máximas que é possível alcançar, usando desde 1 até 16 clientes (aplicações $m_\varepsilon\mu$), em execução no mesmo nó, para interrogar o servidor local. São apresentados dois cenários, correspondentes a dois tipos de máquinas: do lado esquerdo da figura 8.1 é apresentada a taxa alcançada em nós do *subcluster* i686, enquanto que, do lado direito, é apresentada a taxa alcançada em nós do *subcluster* Athlon. É importante salientar que são necessários, pelo menos, dois clientes para atingir níveis de operação máximos, apesar de se poder pensar que um cliente e um servidor seriam sufici-

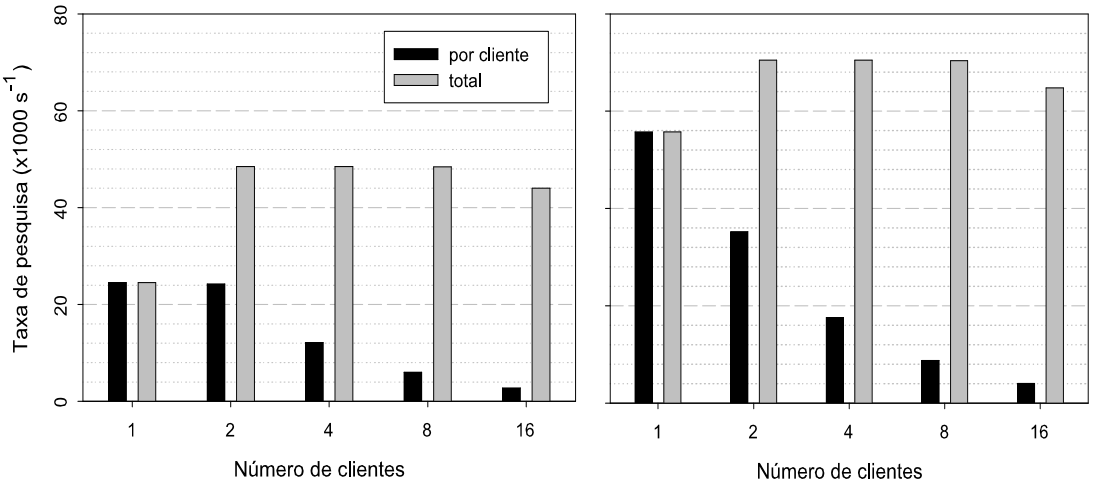


Figura 8.1: Taxas de pesquisa máximas na operação local.

entes para ocupar os dois processadores disponíveis em cada nó. Note-se também que, a partir de 16 clientes, a taxa global de pesquisa cai, devido ao facto de o servidor ser baseado num único fio-de-execução, o que não garante uma distribuição equitativa do tempo de processador, na presença de várias aplicações com múltiplos fios-de-execução activos. Como era de esperar, os resultados são substancialmente melhores nas máquinas Athlon, embora não se verifique um aumento da taxa proporcional ao aumento da frequência dos processadores.

8.1.2 Operação global

As interrogações que não podem ser satisfeitas no contexto local dão origem a pesquisas globais. No programa de teste em causa, na primitiva `meu_get_info`, é especificado o identificador de um recurso que, garantidamente, se sabe estar registado num servidor remoto. Isto significa que os clientes conhecem a localização real da informação que procuram. Tal é possível pelo facto de esses clientes serem também os responsáveis pela criação dos recursos lógicos sobre os quais, posteriormente, tentam obter informação.

O desempenho do serviço de directório no caso das pesquisas globais ao nível do *cluster* depende, essencialmente, do mecanismo de difusão UDP e do hardware Fast Ethernet que o suporta.

A figura 8.2 apresenta as taxas de pesquisa máximas que é possível alcançar nos *subclusters* i686^{+} (gráficos do lado esquerdo da figura) e Athlon (gráficos do lado direito da figura).

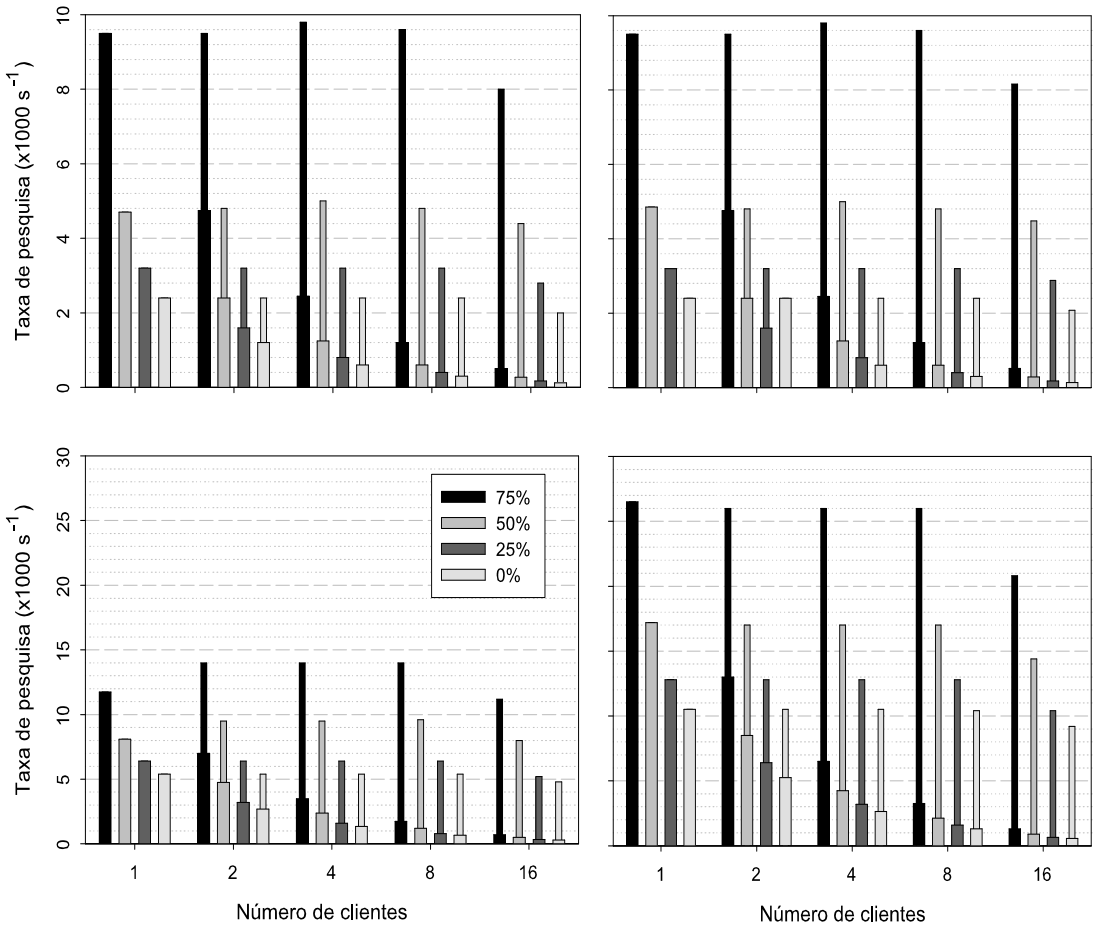


Figura 8.2: Taxas de pesquisa máximas na operação global (4 nós).

Foram também considerados dois tipos de tecnologia Ethernet, para suporte à difusão UDP: Fast Ethernet (gráficos da parte superior da figura) e Gigabit Ethernet (gráficos da parte inferior da figura). Note-se que, para estes testes, houve o cuidado de parametrizar o sistema R_OCl para evitar o recurso a árvores de dispersão no âmbito das tecnologias SAN disponíveis.

Tal como na operação local, foram executados entre 1 e 16 clientes, por nó. Adicionalmente, fez-se variar, entre 0% e 75%, a percentagem de pesquisas que não obrigam a sair do contexto do servidor local, em cada nó.

Como era de esperar, as taxas máximas de obtenção de respostas, observadas em cada cliente (barras grossas dos gráficos da figura 8.2), diminuem com a aumento da percentagem de pesquisas às quais os servidores locais não podem responder directamente. À imagem

do que se verificou para a operação local, as taxas de pesquisa globais (barras finas dos gráficos da figura 8.2) caem quando são usados 16 clientes, por nó.

Surpreendentemente, o *subcluster* Athlon apenas consegue ultrapassar marginalmente o desempenho do *subcluster* i686$^+$, quando é usada a tecnologia Fast Ethernet. Isto significa que não são necessários processadores com níveis elevados de desempenho para levar à exaustão a rede Fast Ethernet. Ainda mais surpreendente é a verificação de que a utilização de Gigabit Ethernet produz um ganho superior a 150% no *subcluster* Athlon, enquanto que, no i686$^+$, esse ganho não ultrapassa 50%. De facto, a pilha TCP/IP e em particular a difusão UDP exigem muito tempo de processador e, portanto, os nós do *subcluster* i686$^+$ não são capazes de tratar as mensagens resultantes da difusão de pedidos à velocidade máxima que a rede suporta.

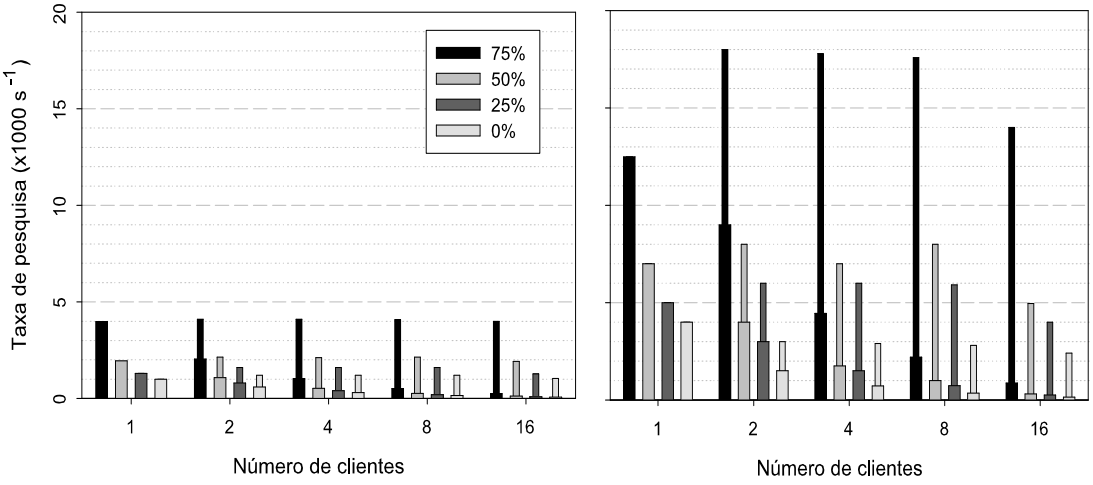


Figura 8.3: Taxas de pesquisa máximas na operação global (8 nós).

Por fim, a figura 8.3 apresenta os resultados obtidos na operação conjunta dos dois *subclusters* i686$^+$ e Athlon. Os testes foram efectuados considerando a interligação das 8 máquinas em causa através de Fast Ethernet (gráfico do lado esquerdo da figura 8.3) ou Gigabit Ethernet (gráfico do lado direito da figura 8.3).

Importa realçar que o aumento do número de nós, de 4 para 8, quando se recorre à tecnologia Fast Ethernet, provoca um decréscimo de cerca de 60% nas taxas de pesquisa globais (comparem-se as barras finas do gráfico à esquerda, na figura 8.3, com as dos gráficos na parte superior da figura 8.2). Já no caso da tecnologia Gigabit, apenas se verifica um decréscimo de cerca de 30%, por comparação com os resultados obtidos no *subcluster* com maior desempenho, o *subcluster* Athlon (comparem-se as barras finas do

gráfico à direita, na figura 8.3, com as do gráfico no canto inferior direito da figura 8.2). Se, em vez dos resultados obtidos no *subcluster* Athlon, fosse usada, como referência, a média dos resultados apresentados nos gráficos da parte inferior da figura 8.2, então o decréscimo seria apenas de cerca de 10%. Deste modo, pode-se concluir que a tecnologia Gigabit proporciona uma boa escalabilidade ao directório, ao contrário da Fast Ethernet.

8.2 Comunicação ponto-a-ponto

A comunicação ponto-a-ponto corresponde à troca de mensagens entre pares de tarefas, com ou sem envolvimento de caixas postal ou de um operão, e à leitura e escrita de blocos de memória global, sem contemplar a possibilidade de existirem pseudónimos. Em relação à troca de mensagens, a avaliação de desempenho teve em consideração a localização dos recursos lógicos envolvidos e a agregação de tecnologias nos nós dotados de múltiplos interfaces de comunicação.

8.2.1 Troca de mensagens intranó

A troca de mensagens entre entidades localizadas no mesmo nó tira partido dos mecanismos IPC. No $m_{\mathcal{E}}\mu$, as tarefas são criadas no contexto de operões, aos quais correspondem processos do sistema operativo.

O envio de uma mensagem a uma tarefa ou caixa postal, a partir de uma tarefa do mesmo operão, corresponderá a uma simples operação de cópia seguida de um evento de sincronização. A cópia de memória poderá, eventualmente, ser dispensada, se o programador indicar que o tampão de memória associado à mensagem deve ser descartado após a conclusão do envio.

Se o recurso alvo de uma mensagem (tarefa, caixa postal ou operão) não pertencer ao operão da tarefa origem, mas se estiver inserido no mesmo nó do *cluster*, então, a exploração dos mecanismos de *loopback* implementados pelo M-VIA e pelo UDP[1] garantem desempenhos equivalentes ao IPC e facilitam a implementação da biblioteca.

A figura 8.4 apresenta os tempos de ida-volta para mensagens trocadas entre duas tarefas em execução num dado nó do *cluster*. Estes testes foram efectuados numa das máquinas do *subcluster* i686.

Dos resultados apresentados, importa realçar que o tempo de ida-volta tem como limite mínimo 8μs, valor que corresponde às operações de sincronização e escalonamento de fios-de-execução do utilizador (usados para implementar as tarefas) e do sistema de despacho.

[1] O GM incluirá também um mecanismo de *loopback* numa futura versão.

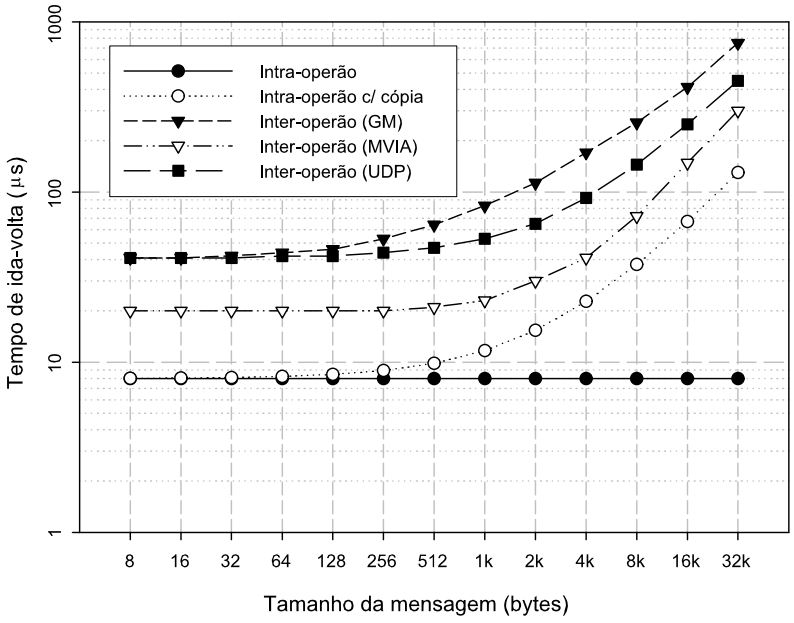


Figura 8.4: Tempo de ida-volta intranó.

Pode-se ainda verificar que o mecanismo de *loopback* do M-VIA garante melhor desempenho na troca de mensagens entre tarefas de operões distintos. Refira-se que, no caso do GM, dado que ainda não existe um mecanismo de *loopback*, as mensagens são enviadas através do comutador, tendo como origem e destino o mesmo controlador Myrinet.

8.2.2 Troca de mensagens *intrasubcluster*

Ao nível do *subcluster*, o $m_{\varepsilon}\mu$ explora uma determinada tecnologia de comunicação de elevado desempenho, Myrinet ou Gigabit, através das bibliotecas de baixo-nível GM e M-VIA. Para efeitos de comparação, os testes de desempenho levados a cabo neste âmbito também incluíram a utilização do protocolo UDP, suportado para permitir a experimentação quando não existe uma rede SAN.

Por uma questão de uniformização, todos os resultados apresentados foram obtidos usando duas máquinas do *subcluster* i686^{+}. No entanto, a utilização de máquinas do *subcluster* Athlon, apenas traz vantagens para o protocolo UDP, o qual requer uma significativa intervenção do processador. Refira-se que uma das características das tecnologias de comunicação de elevado desempenho é oferecerem tempos de comunicação pouco dependentes do processador.

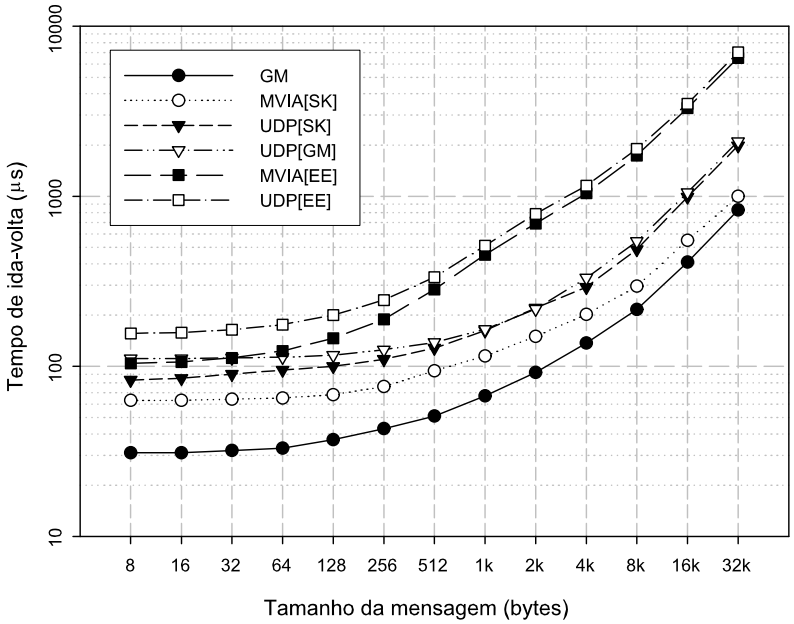


Figura 8.5: Tempo de ida-volta internó, *intrasubcluster*.

A figura 8.5 apresenta os tempos de ida-volta obtidos para a troca de mensagens entre tarefas em execução em máquinas distintas do *subcluster* i686$^+$. A experiência incidiu essencialmente na avaliação do suporte aos subsistemas de comunicação GM, M-VIA e UDP. O GM é, evidentemente, usado para explorar a tecnologia Myrinet. O M-VIA, como foi referido, é usado para explorar a tecnologia Gigabit, mas também permite explorar a tecnologia Fast Ethernet[2]. O UDP permite, naturalmente, explorar as tecnologias Gigabit e Fast Ethernet e, por via do suporte incluído no GM, também permite explorar a tecnologia Myrinet.

O gráfico da figura 8.5 permite confirmar dois aspectos importantes: a tecnologia Myrinet e o respectivo subsistema de comunicação (o GM) permitem alcançar os melhores resultados; o protocolo UDP impõe uma sobrecarga de processamento tal, que não permite alcançar os níveis de desempenho que o hardware de comunicação é capaz de oferecer.

8.2.3 Troca de mensagens *intersubcluster*

Um dos objectivos principais do $m_{\varepsilon}\mu$ e do R_OCl é a exploração eficiente de *clusters* multi--SAN. Nos casos em que o programador não é capaz de tirar partido da localidade ao nível

[2]Os controladores SysKonnect 9821 tal como os Intel Ether Express Pro100 são suportados pelo M-VIA, enquanto que os 3Com 3c905C Tornado não o são.

do *subcluster*, os recursos lógicos da aplicação, espalhados por nós de diferentes *subclusters*, terão, eventualmente, que trocar informação entre si. Neste cenário, a comunicação é efectuada com base no mecanismo de reencaminhamento do R_OCl.

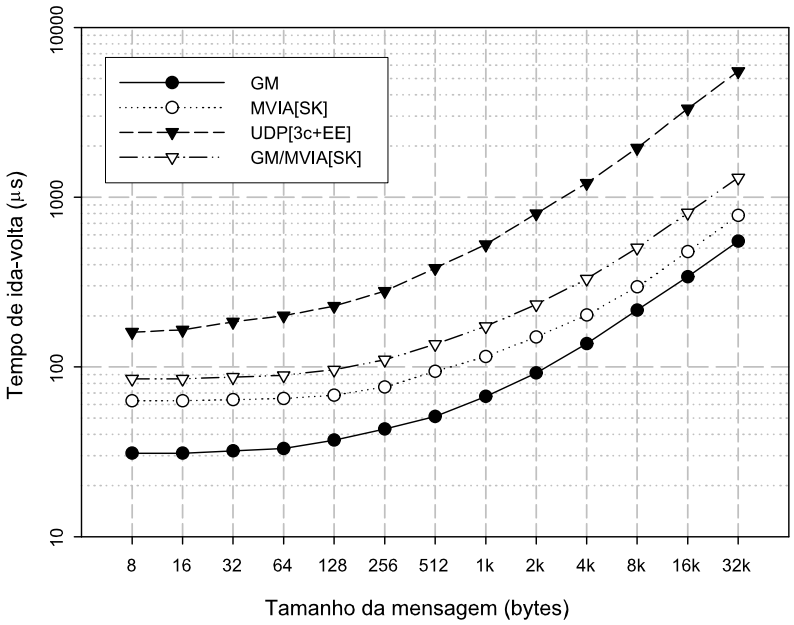


Figura 8.6: Tempo de ida-volta *intersubcluster*.

A figura 8.6 apresenta os tempos de ida-volta para mensagens trocadas entre tarefas em execução em máquinas de *subclusters* distintos – uma tarefa num nó do *subcluster* i686 e outra num nó do *subcluster* Athlon. As mensagens são transportadas da rede Myrinet para a rede Gigabit e vice-versa através do serviço de reencaminhamento da máquina do *cluster* Xeon.

Como referência, são apresentados os tempos de ida-volta para mensagens trocadas entre tarefas em execução em nós do *subcluster* i686 (rede Myrinet, subsistema GM) e em nós do *subcluster* Athlon (rede Gigabit, subsistema M-VIA). São também apresentados os resultados que se podem alcançar usando a rede Fast Ethernet e o subsistema UDP, os quais constituem um meio de comunicação alternativo entre máquinas dos dois *subclusters*[3].

Pode-se concluir que o reencaminhamento R_OCl permite alcançar desempenhos bastante superiores aos alcançáveis pela simples utilização do subsistema UDP. O tempo de ida-volta *intersubcluster*, como seria de esperar, é aproximadamente igual à soma dos tempos de ida-volta em cada um dos *subclusters* envolvidos.

[3]Note-se que os controladores 3Com 3c905C Tornado não são suportados pelo M-VIA.

8.2.4 Envio de mensagens com agregação de tecnologias

Os nós que dispõem de múltiplas tecnologias de comunicação, para além de servirem de ponte entre *subclusters* distintos, deverão também colocar à disposição das aplicações locais uma largura de banda superior. O R_OCl utiliza estratégias básicas para proceder à agregação dos subsistemas GM e M-VIA de forma transparente às aplicações, nomeadamente alternando entre os dois subsistemas no envio de mensagens consecutivas.

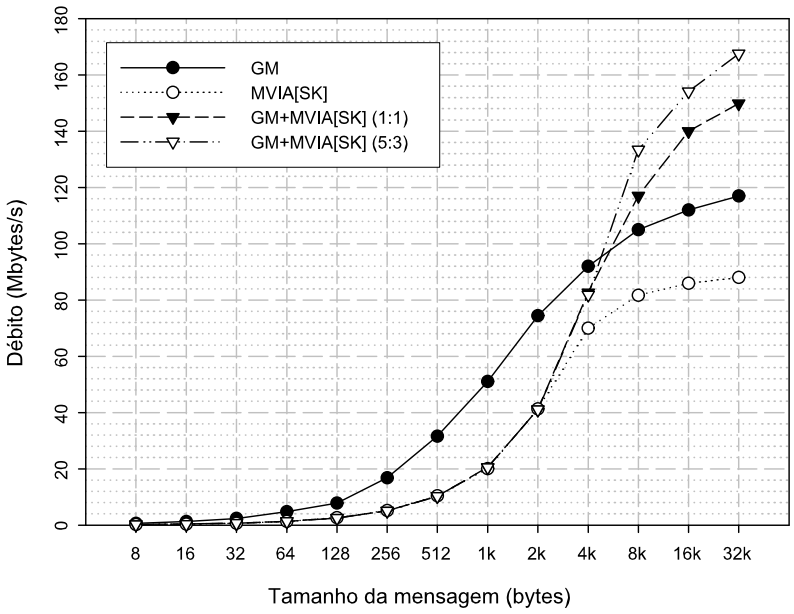


Figura 8.7: Débito na agregação de tecnologias.

A figura 8.7 apresenta os débitos alcançáveis no envio de sequências de mensagens, num único sentido, entre duas tarefas em execução em dois nós do *subcluster* i686$^+$. Para efeitos de comparação, são apresentados valores respeitantes à utilização individual de cada um dos subsistemas de comunicação (GM e M-VIA), juntamente com os valores relativos a duas estratégias de agregação de tecnologias.

Como se pode verificar, a simples alternância entre o GM e o M-VIA (*1:1* – 1 mensagem enviada através do GM, 1 mensagem enviada através do M-VIA e assim sucessivamente) já permite ultrapassar o desempenho alcançado usando unicamente o M-VIA, para mensagens superiores a 2kbytes, ou mesmo o GM, para mensagens superiores a 4kbytes. Para mensagens pequenas não há qualquer ganho, devido ao facto de a agregação das duas tecnologias obrigar à actuação de dois fios-de-execução, um por cada tecnologia, ao nível do sistema de despacho do R_OCl. Estes dois fios-de-execução concorrem entre si pelo tempo

de processador e, portanto, os ganhos que advêm da utilização de dois canais físicos de comunicação acabam por ser anulados. Tal não acontece em mensagens de tamanho superior, dado que o sistema de despacho actua menos vezes, isto é, de forma mais espaçada no tempo.

Se forem enviadas 5 mensagens através do GM seguidas de 3 através do M-VIA e assim sucessivamente, o débito alcançado é ainda superior ao conseguido com a alternância simples. De facto, dado que o GM proporciona níveis de desempenho superiores, é normal que esse subsistema seja usado para enviar um maior número de mensagens. Através de uma simples análise às curvas do GM e do M-VIA, conclui-se que o GM permite atingir um débito cerca de 45% mais elevado, para mensagens de 32kbytes.

8.2.5 Acesso à memória global

O mecanismo de acesso à memória global do $m_{\varepsilon}\mu$ pode, eventualmente, envolver operações complexas, por via das decisões do programador, quanto à organização dos vários blocos e agregadores de memória envolvidos. Com o intuito de analisar de forma mais directa o desempenho deste mecanismo, foi considerado, apenas, o acesso a simples blocos de memória.

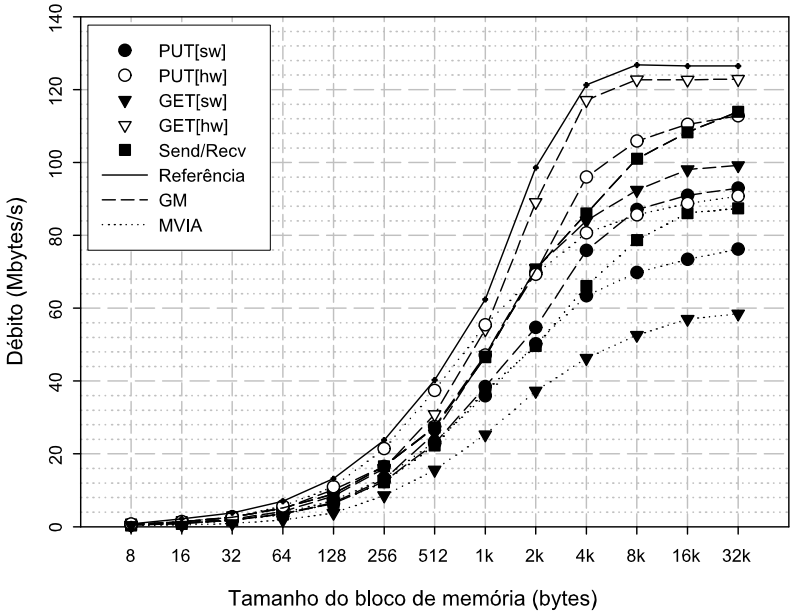


Figura 8.8: Débito nas operações de escrita e leitura remotas.

A figura 8.8 apresenta os débitos máximos alcançados no acesso a blocos de memória remotos, por parte de uma tarefa, usando os mecanismos de leitura e escrita remota, oferecidos pelo R_OCl, nas suas duas variantes, isto é, com recurso ou não ao suporte RDMA do GM e do M-VIA. Na legenda da figura, as designações PUT e GET referem-se às operações de escrita e leitura, respectivamente, enquanto os sufixos *[hw]* e *[sw]* se referem à utilização ou não, respectivamente, do suporte RDMA. Para efeitos de comparação, é também apresentado o débito alcançado através do envio de mensagens (com evocação da primitiva `send` na origem e `recv` no destino). Encontra-se ainda representada uma curva de referência, a qual corresponde a valores de débito virtuais que superam todos os demais e que será usada na análise do grau de sustentação apresentada posteriormente.

Os valores obtidos, para os vários tamanhos de blocos de memória, permitem concluir que, tanto a leitura como a escrita remota, desde que exista suporte RDMA, permitem alcançar maiores débitos que o normal emparelhamento de primitivas de envio e recepção, para além das vantagens que o paradigma da memória global poderá ter, de acordo com a aplicação. É importante referir que a leitura remota com suporte RDMA do GM permite alcançar os melhores resultados, para mensagens superiores a 1kbyte. No caso do M-VIA, como já foi referido, apenas existe suporte RDMA para a operação de escrita.

8.3 Comunicação e múltiplos fios-de-execução

Uma das características mais relevantes do $m_\varepsilon\mu$ é a sua adequação à implementação de aplicações que recorrem a um número elevado de fios-de-execução. De facto, tanto o modelo de comunicação do R_OCl com o modelo de programação $m_\varepsilon\mu$ assumem a existência de fios-de-execução POSIX, oferecidos pelo sistema operativo, como elementos essenciais para a modelação de aplicações e a exploração de nós SMP.

8.3.1 Impacto da comutação de contextos

Habitualmente, os sistemas de passagem de mensagens convencionais não integram fios-de--execução ou então incluem sistemas de fios-de-execução proprietários com funcionalidade limitada, por forma a evitar tempos de comutação e sincronização supostamente muito elevados e, portanto, incompatíveis com os tempos de resposta do hardware de comunicações. No contexto em que as bibliotecas R_OCl e $m_\varepsilon\mu$ foram desenhadas, o aparecimento da biblioteca NPTL veio mostrar que a utilização de fios-de-execução POSIX não compromete, necessariamente, o desempenho do sistema de passagem de mensagens.

A figura 8.9 compara os tempos de resposta – intervalo de tempo desde a evocação da primitiva de envio até ao retorno da primitiva de recepção, na tarefa de destino – associados

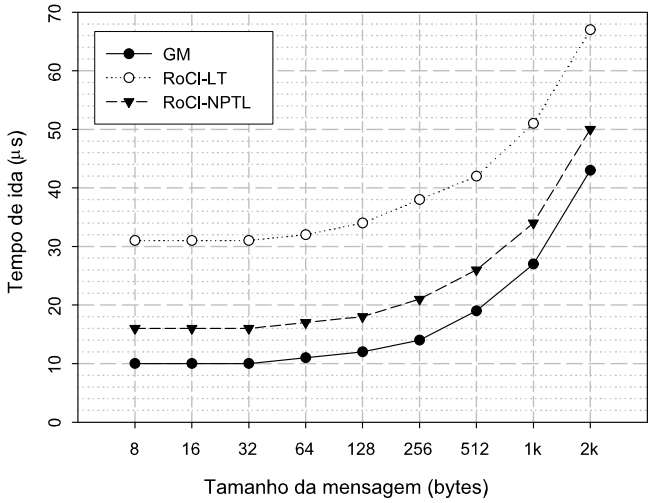


Figura 8.9: Impacto da comutação de contextos no tempo de resposta do envio.

ao envio de mensagens, através do GM, entre duas tarefas em execução em dois nós do *subcluster* i686. A curva *GM* mostra o desempenho que é possível alcançar com a biblioteca de baixo-nível GM, ocupando um processador, a tempo inteiro, para efectuar a sondagem do meio de comunicação. As curvas *RoCl-LT* e *RoCl-NPTL* mostram o desempenho do sistema R_OCl, em função das bibliotecas LinuxThreads e NPTL. Refira-se que o R_OCl recorre às primitivas de recepção bloqueante do GM, pelo que não ocupa um processador enquanto aguarda a chegada de mensagens.

É de realçar o aumento de desempenho obtido com a introdução do NPTL, que se deve, essencialmente, à diminuição dos tempos de comutação e de sincronização entre fios-de--execução. Os 5μs adicionais visíveis na figura 8.9, que o R_OCl introduz em relação ao GM, são o preço a pagar pela flexibilidade oferecida, isto é, pela orientação ao recurso e pelo modelo de comunicação totalmente assíncrono.

8.3.2 Tempo de resposta a um evento

Para analisar, de forma mais exaustiva, as possíveis vantagens da utilização da biblioteca NPTL, foi desenhada uma aplicação que permite avaliar o impacto da comutação de contextos (em fios-de-execução) no tempo necessário para reagir a um determinado evento. A aplicação em causa consiste num fio-de-execução produtor que gera eventos e em vários fios-de-execução consumidores que tratam esses eventos. Desta forma, é reproduzido o ambiente de execução $m_{\mathcal{E}}\mu$, no que diz respeito à recepção e processamento de mensagens.

O produtor gera um número elevado de eventos e entrega, cada um deles, aleatoriamente, a um consumidor. Depois de entregar um evento a um consumidor, através dos mecanismos de sinalização dos fios-de-execução POSIX, o produtor aguarda um determinado intervalo de tempo, previamente fixado. Por sua vez, os consumidores aguardam por eventos bloqueando-se em variáveis de condição. Na resposta a um evento, o consumidor apenas actualiza uma variável de estado associada a esse evento, indicando que este foi efectivamente tratado.

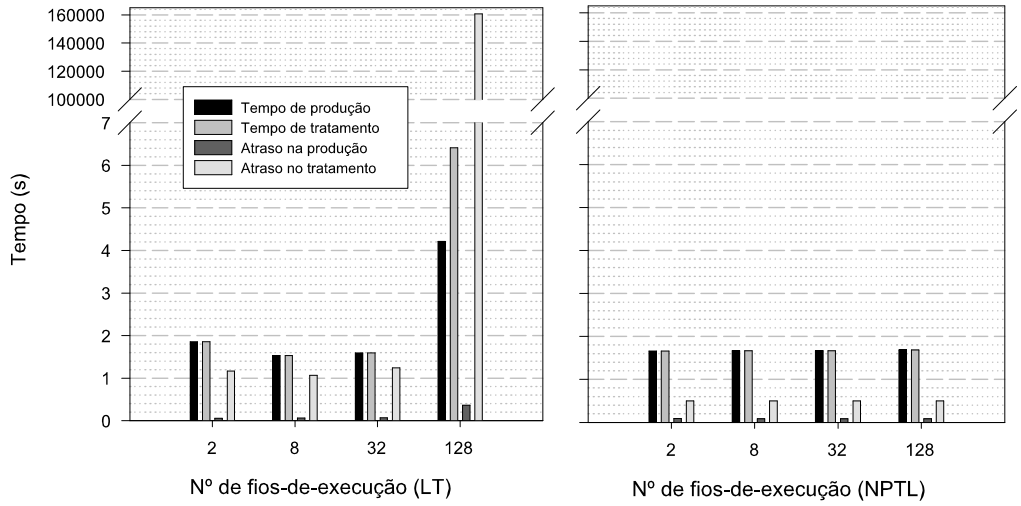


Figura 8.10: Tratamento de eventos com fios-de-execução LT e NPTL.

A figura 8.10 apresenta os tempos totais de produção e tratamento dos eventos bem como os respectivos atrasos acumulados. O tempo de produção diz respeito ao tempo necessário para o produtor gerar e entregar todos os eventos, o qual nunca será inferior a 1s, dado que são gerados 100000 eventos com um período 10μs. O tempo de tratamento diz respeito ao tempo necessário para que todos os eventos sejam efectivamente tratados, pelos vários consumidores, e será sempre superior ao tempo de produção. O atraso na produção de um evento é calculado como a diferença entre o instante em que o evento foi gerado e o instante em que o devia ter sido (um evento deverá ser gerado 10μs após o instante em que o evento anterior foi entregue/assinalado). O atraso no tratamento de um evento é calculado em função do instante em que o evento é entregue (imediatamente a seguir à evocação da primitiva `pthread_cond_signal`) e do instante em que o consumidor inicia o seu tratamento (imediatamente antes de o consumidor actualizar a variável de estado associada ao evento).

A execução desta aplicação de teste, num dos nós do *subcluster* i686, mostrou que os tempos de atraso acumulados, no tratamento de eventos, são cerca de 50% inferiores,

quando é usada a biblioteca NPTL. Isto significa que os fios-de-execução correspondentes aos consumidores são escalonados mais rapidamente, permitindo uma resposta a eventos mais expedita. No entanto, é com um número elevado de fios-de-execução que se pode perceber a principal vantagem do NPTL; esta biblioteca, ao contrário do LinuxThreads, foi desenhada para lidar de forma escalável com um elevado número de fios-de-execução, garantindo níveis de desempenho praticamente constantes.

8.3.3 Troca de mensagens concorrente

Um pressuposto importante deste trabalho é a utilização de múltiplos fios-de-execução, sendo de esperar que, em determinados momentos da fase de execução de uma aplicação, ocorram acessos concorrentes aos mecanismos de comunicação. Por exemplo, será natural coexistirem, num nó computacional, várias tarefas a trocar mensagens com outras tarefas num outro nó.

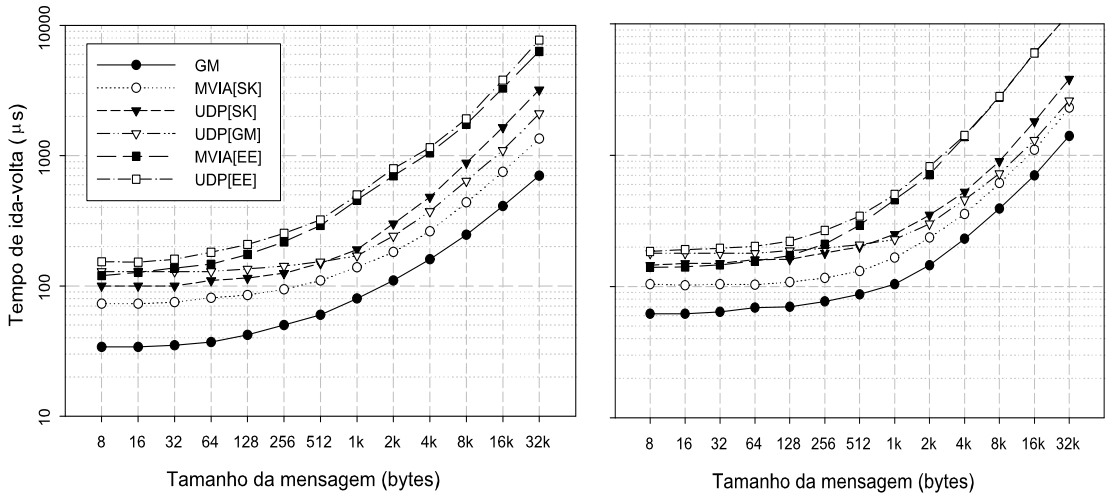


Figura 8.11: Tempos de ida-volta na troca de mensagens concorrente.

A figura 8.11 apresenta os tempos de ida-volta para a troca de mensagens concorrente, entre tarefas em execução em duas máquinas. No gráfico do lado esquerdo são apresentados os resultados obtidos com dois pares de tarefas e no do lado direito os resultados obtidos com quatro pares de tarefas. Cada par de tarefas, uma em cada máquina, troca mensagens sem entrar em consideração com os demais pares de tarefas. Por forma a facilitar a comparação com os resultados apresentados na secção 8.2.2, relativos à troca de mensagens sem concorrência, foi usada a mesma configuração de hardware. Refira-se que, uma vez que cada máquina possui apenas dois processadores, a utilização de quatro pares de tarefas

é suficiente para garantir que nenhum fio-de-execução, da aplicação ou do sistema, pode usar em exclusividade um processador.

Da análise dos resultados pode-se concluir que os tempos de ida-volta não são proporcionais ao número de fios-de-execução que simultâneamente enviam mensagens, mesmo quando o hardware de comunicação obriga o processador a uma intervenção prolongada, como acontece com a tecnologia Fast Ethernet, quando explorada via UDP. De facto, quando são usados dois pares de tarefas, o impacto é relativamente baixo, essencialmente devido ao facto de existirem dois processadores em cada nó computacional. Quando são usados quatro pares, os tempos de ida-volta passam para o dobro (e não para o quadruplo). Para mensagens de grande dimensão, o impacto do acesso concorrente é maior, pelo facto de se exigir uma maior largura de banda do meio de comunicação, visto que, para o mesmo tempo de preparação da mensagem, onde o processador intervém, o hardware de comunicação tem mais trabalho a seu cargo.

8.3.4 Grau de sustentação

Para uma correcta avaliação de um sistema de passagem de mensagens, é indispensável avaliar o impacto da comunicação na computação e vice-versa. De facto, numa aplicação paralela, de nada servirá optimizar o tempo de comunicação se isso tiver um impacto muito negativo na disponibilidade do processador para desempenhar outras tarefas, principalmente se existirem múltiplos fios-de-execução, por nó, que efectuam assincronamente cálculos e enviam/recebem mensagens.

Para avaliar esta interdependência, determinou-se a taxa de execução de uma determinada rotina de cálculo, executando dois fios-de-execução (duas rotinas de cálculo), um por cada processador de um nó do *subcluster* i686$^+$. Durante a execução dessa rotina, a máquina em causa não executou qualquer operação adicional. Em seguida foram executados os testes de avaliação do tempo de ida-volta, cujos resultados, em condições óptimas, foram apresentados na secção 8.2.2, em simultâneo com as tais rotinas de cálculo.

O impacto na computação pode ser expresso pelo rácio TE_{SC}/TE_0, onde TE_{SC} representa a taxa de execução da rotina de cálculo quando, concorrentemente, é executado o teste de avaliação do tempo de ida-volta para um dado subsistema de comunicação SC (GM sobre Myrinet, M-VIA sobre Gigabit, etc.) e TE_0 representa a taxa de execução obtida inicialmente, sem que tivesse ocorrido qualquer operação de comunicação em simultâneo. De igual forma, o rácio IV_{SC_0}/IV_{SC} expressa o impacto na comunicação, onde IV_{SC_0} e IV_{SC} representam os tempos de ida-volta obtidos para um dado subsistema de comunicação, com ou sem computação a decorrer em simultâneo, respectivamente. Para uma comparação mais fácil dos diferentes rácios relativos ao impacto na comunicação, usou-se o rácio IV_{GM_0}/IV_{SC}, onde IV_{GM_0} representa o tempo de ida-volta, sem computação de

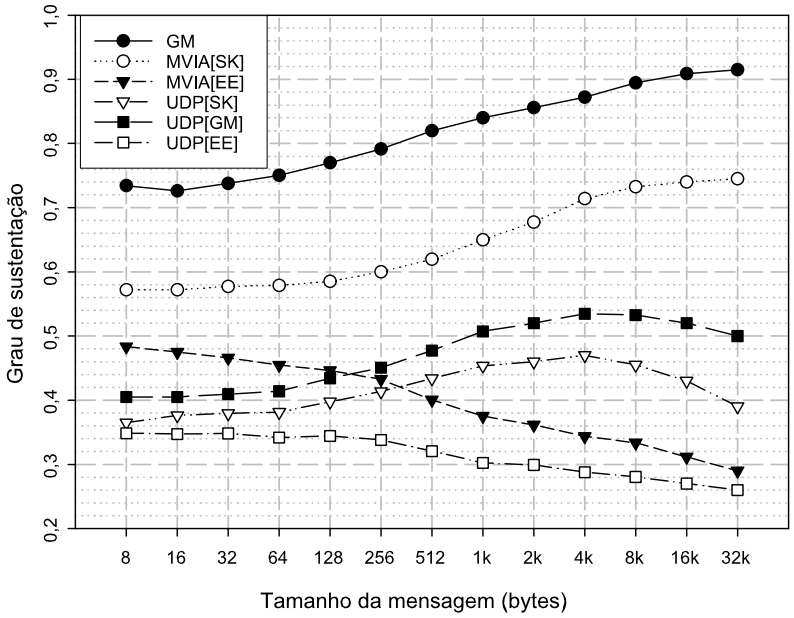


Figura 8.12: Grau de sustentação do desempenho global na troca de mensagens.

fundo, para o subsistema GM, o qual garante os tempos de ida-volta mais baixos.

Os rácios TE_{SC}/TE_0 e IV_{GM_0}/IV_{SC} expressam o grau de sustentação do desempenho na computação e na comunicação, quando o tempo de processador é partilhado pelas duas. Considerando que a computação e a comunicação têm igual peso no desempenho global de uma aplicação, pode ser usado um único rácio global, por cada subsistema de comunicação, calculado como a média geométrica dos rácios definidos ($\sqrt{TE_{SC}/TE_0 \times IV_{GM_0}/IV_{SC}}$).

A figura 8.12 apresenta o grau de sustentação do desempenho global (com base no rácio global apresentado) que se pode esperar numa aplicação $m_\varepsilon\mu$. Note-se que os graus de sustentação estão em concordância com a qualidade do hardware de comunicação e a arquitectura do subsistema de comunicação: os controladores Fast Ethernet, não só têm o pior desempenho, como também requerem uma maior intervenção do CPU, à medida que o tamanho das mensagens aumenta; o protocolo UDP, devido à sua complexidade, necessita de mais tempo de processador que o M-VIA e o GM, fazendo com que o desempenho caia.

Leitura e escrita remotas

Em relação ao acesso à memória global, também é possível avaliar o impacto da leitura/escrita remota na computação e vice-versa. No entanto, em vez do tempo de ida-

-volta, usado para a troca de mensagens, terá agora que ser usado o débito, ou seja, no lugar do rácio IV_{SC_0}/IV_{SC}, é usado o rácio D_{SC_0}/D_{SC}. Além disso, na leitura e escrita remotas, o envolvimento da origem – a máquina onde é evocada a primitiva de leitura ou escrita remota – e do destino – a máquina alvo da leitura ou escrita remota – não é o mesmo e, portanto, torna-se necessário distinguir entre o grau de sustentação do desempenho na computação na origem e o seu equivalente no destino. Assim, o rácio TE_{SC}/TE_0 é substituído pelo rácio $(TE_{SC}^{orig} + TE_{SC}^{dest})/(2 \times TE_0)$.

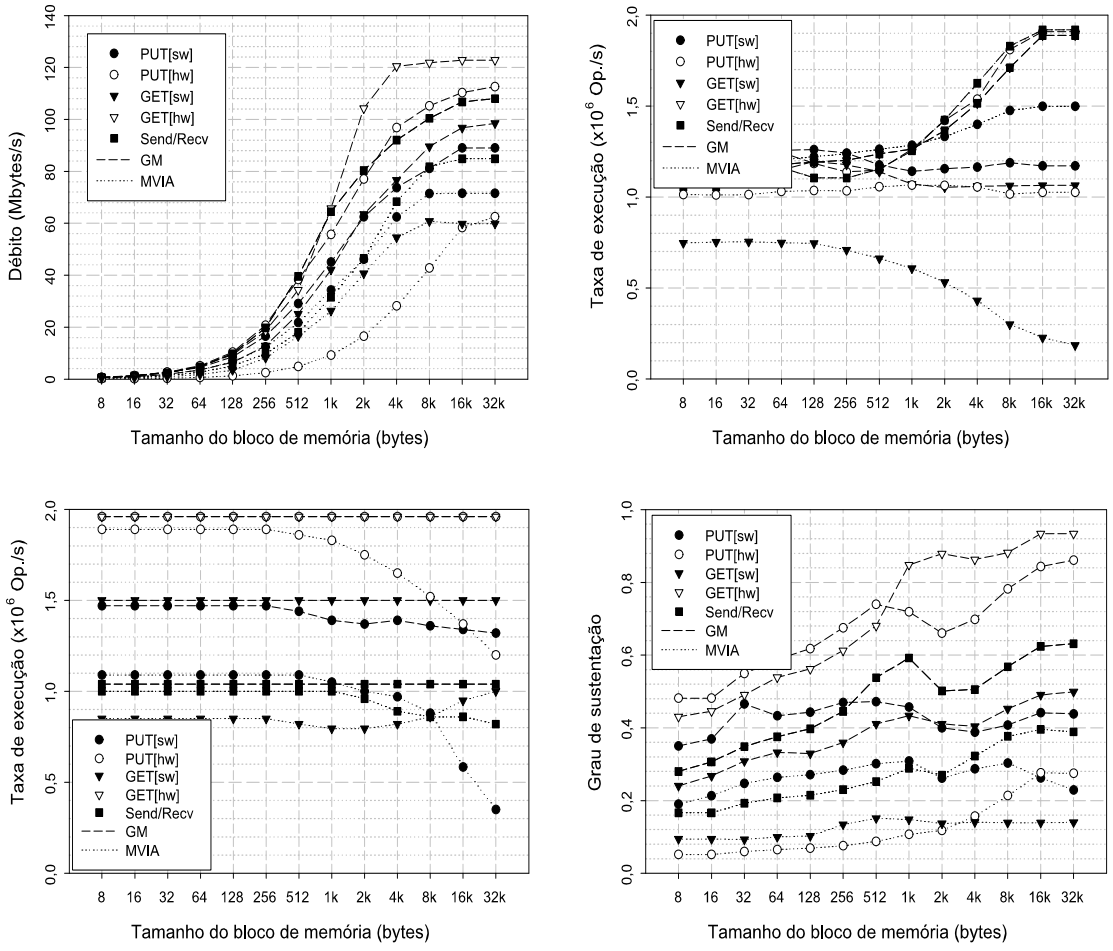


Figura 8.13: Grau de sustentação do desempenho no acesso à memória remota.

A figura 8.13 apresenta os valores dos débitos alcançados na leitura/escrita de memória remota e no envio seguido de recepção explícita no destino (gráfico do canto superior esquerdo), os valores das taxas de execução da rotina de cálculo na origem e no destino (gráficos do canto superior direito e canto inferior esquerdo, respectivamente) e os valores do grau de sustentação do desempenho global, cujo cálculo se efectuou através da expressão

$\sqrt{(TE_{SC}^{orig} + TE_{SC}^{dest})/(2 \times TE_0) \times D_{ref}/D_{SC}}$ (gráfico do canto inferior direito). Refira-se que, para o cálculo do grau de sustentação do desempenho na comunicação, usou-se a curva de referência apresentada no gráfico da figura 8.8, isto é, no lugar do rácio IV_{GM_0}/IV_{SC} usou-se o rácio D_{ref}/D_{SC}.

Pode-se concluir que o suporte RDMA do GM/Myrinet permite atingir desempenhos significativamente superiores aos alcançáveis com envios emparelhados com recepções explícitas. No que diz respeito ao M-VIA/SysKonnect, a escrita remota (a única operação com suporte RDMA) produz resultados decepcionantes devido à ocupação do processador. É particularmente interessante verificar que a operação de escrita remota implementada ao nível do R_OCl, no caso do M-VIA, permite alcançar um nível de desempenho superior ao da operação de escrita que tira partido do suporte RDMA, para blocos de memória inferiores a 16kbytes[4].

8.4 Avaliação integrada de desempenho

Na generalidade dos casos, a avaliação, depuração e comparação de sistemas de passagem de mensagens baseia-se em medições dos tempos de ida-volta e da largura de banda, apesar de os programadores estarem mais interessados em conhecer o verdadeiro impacto da comunicação em aplicações reais.

Dado que não existe uma ferramenta de avaliação genérica para plataformas paralelas/distribuídas, foi desenvolvida de raíz uma aplicação de teste que poderá ser usada noutros ambientes.

8.4.1 Descrição do funcionamento

No cenário de teste idealizado, vários testemunhos circulam através dos nós de um *cluster*, consumindo tempo de processador cada vez que chegam a um nó.

Na fase de arranque do teste, um nó do *cluster* produz testemunhos, enviando-os aleatoriamente aos restantes nós. Em cada um destes nós, são criados múltiplos operões e tarefas, com base em parâmetros indicados. O envio de um testemunho pressupõe a indicação de uma tarefa de destino.

Cada testemunho transporta consigo uma semente, calculada aleatoriamente, um tempo de vida e um nível de esforço computacional. A semente e o tempo de vida são usados, à chegada a uma dada tarefa, para determinar o próximo destino do testemunho. O tempo de processamento exigido por cada testemunho é indicado pelo nível de esforço

[4]Note-se que a implementação R_OCl para as primitivas de leitura e escrita remotas baseada em operações de envio e recepção implica cópias de memória.

computacional. O tempo de vida é descontado em cada tarefa e, quando atingir o valor zero, indicará que o testemunho deverá ser devolvido ao nó produtor.

O produtor contabiliza o intervalo de tempo desde a introdução do primeiro testemunho até à chegada do último. Através da variação do número de operões em cada nó, consegue-se avaliar o impacto da utilização de múltiplos contextos R_OCl, ou seja, múltiplos portos de comunicação das bibliotecas de baixo-nível. Através da variação do número de tarefas por operão, consegue-se avaliar o impacto dos mecanismos de multiplexagem. Para averiguar a capacidade de sobrepor comunicação e computação, basta variar o nível de esforço computacional exigido por cada testemunho.

8.4.2 Valores de desempenho

A avaliação do $m_\varepsilon\mu$, com base na plataforma de avaliação descrita, foi efectuada usando os quatro nós do *cluster* i686 e o nó único do *cluster* Xeon, tendo-se considerado, unicamente, a tecnologia de comunicação Myrinet. Cada nó do *cluster* i686 foi usado para executar múltiplas combinações de operões (O) e tarefas (T), sendo que $O \in \{1,2,4\} \wedge T \in \{1,2,4,8\}$. O nó Xeon foi usado para produzir 32 testemunhos (Tm), cada um com 256bytes, tempo de vida (TV) 10000 e um nível de esforço computacional $C \mid C \in \{0,1,10\}$. Um nível de esforço computacional C equivale a $145C\mu$s de computação, simulados através de uma séria de operações aritméticas.

Para cada cenário, o tempo de execução pode ser estimado através da expressão $[(Tm \times TV \times C \times 145) \div max(O \times T \times 4, 2 \times 4) + TC]$ μs, assumindo que os testemunhos visitarão uniformemente todos os nós do *cluster* i686 e considerando que, em cada nó, existem dois processadores. O tempo de comunicação (TC) poderia ser estimado através da expressão $[Tm \times TV \times (TempoIdaVolta \div 2)]$ μs, usando os valores do tempo de ida-volta da figura 8.5, mas isso seria assumir que a computação não tem impacto na comunicação. Em alternativa, definiu-se o tempo de comunicação como o tempo gasto na troca de mensagens juntamente com o acréscimo infligido no tempo de execução da aplicação.

A figura 8.14 apresenta os tempos de comunicação – tempos de circulação da totalidade dos testemunhos – para todos os cenários possíveis. Os valores foram calculados subtraindo os tempos de processamento dos testemunhos ao tempo de execução do teste (TT) medido para cada cenário, ou seja, $TC = TT - [(Tm \times TV \times C \times 145) \div max(O \times T \times 4, 2 \times 4)]$.

Tal como se esperava, o tempo de comunicação aumenta em função da computação. Por um lado, o sistema de despacho de mensagens consome ciclos de processamento, reduzindo a capacidade computacional do nó. Por outro lado, o processamento exigido para cada testemunho atrasa a operação do sistema de despacho, degradando a comunicação.

O número de tarefas por operão é também determinante para o desempenho global do

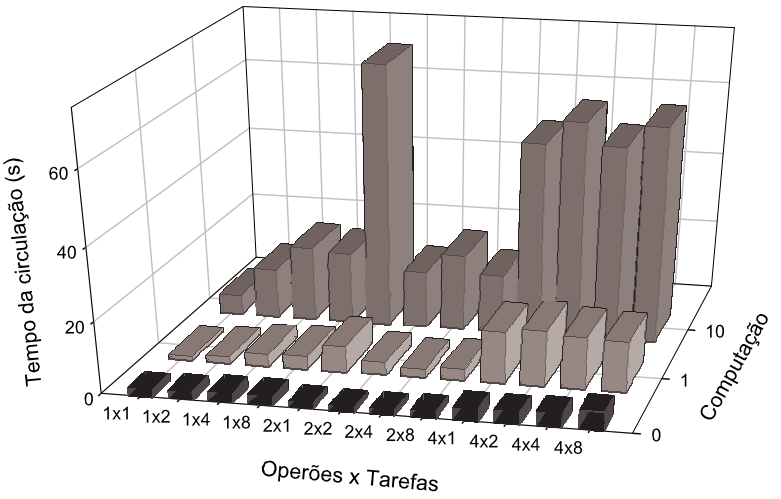


Figura 8.14: Tempo da circulação de testemunhos em diferentes cenários.

sistema, visto que um número elevado de tarefas reduz a percentagem de tempo de processador disponível para o sistema de despacho. De facto, os fios-de-execução correspondentes às tarefas concorrem pelo acesso ao processador em igualdade de circunstâncias com os fios-de-execução do sistema de despacho, os quais são em número fixo (um por cada tecnologia de comunicação).

Por fim, é importante verificar que o número de operões por nó também influência os resultados do teste; para um dado número de tarefas por nó ($O \times T$), são obtidos piores resultados quando o número de operões é superior. Isto permite concluir que é preferível o esquema de multiplexagem do R_OCl – um porto de comunicação de baixo-nível para vários recursos criados num único contexto – que a utilização de múltiplos portos de comunicação de baixo-nível, um por processo, automaticamente multiplexados pelo GM.

8.5 Desempenho de um sistema de armazenamento

Para além da avaliação de desempenho baseada em testes sintéticos, foi também analisado o comportamento do $m_\varepsilon\mu$ em algumas situações reais. Aqui é apresentado um exemplo de aplicação que traduz o funcionamento básico de um sistema de tabelas de *hash* distribuído adaptado à operação em ambiente *cluster*.

8.5.1 Funcionamento básico

A distribuição de partições pelos nós do *cluster* é um procedimento dinâmico que pode ser brevemente descrito da seguinte forma:

- a primeira entidade a entrar no sistema assume a totalidade dos índices de *hash*, passando a deter uma superpartição com todas as entradas da tabela de *hash*;
- uma entidade subsequente escolhe aleatoriamente um índice e solicita ao detentor da partição que contém esse índice que lhe ceda metade dessa partição;
- uma entidade que recebe uma subpartição guarda a identificação da entidade que detinha a partição original e vice-versa, para efeitos de reencaminhamento;
- para alcançar a entidade que detém a partição correspondente a um dado índice, é enviado um pedido a uma entidade escolhida aleatoriamente, a qual procederá ao seu reencaminhamento, se necessário, de acordo com a informação acima mencionada.

Para o armazenamento e a recuperação de informação, um cliente deverá calcular um índice de *hash* e contactar a entidade detentora desse índice da forma acima descrita.

8.5.2 Implementação

Para a implementação deste sistema foram desenvolvidos dois pequenos programas: um servidor, que deverá ser instalado nos nós do *cluster* usados para suportar o sistema, e um cliente, que permite, a partir de qualquer nó do *cluster*, manipular a tabela de *hash*.

Funcionamento do servidor

O servidor começa a sua operação criando um operão e uma tarefa, à qual é associada a propriedade `tipo=servidor`. De seguida, tentará encontrar no sistema uma qualquer tarefa com a propriedade `tipo=partição`. Se for encontrada uma tarefa, o seu identificador será usado como ponto de entrada no sistema; caso contrário, o servidor cria uma outra tarefa, associando-lhe a propriedade `tipo=partição`, que se encarregará de assumir o controlo da superpartição. O identificador desta tarefa será o ponto de entrada no sistema.

O ponto de entrada é usado para solicitar uma subpartição; uma mensagem com um índice calculado aleatoriamente é enviada à tarefa que representa o ponto de entrada. Cada servidor poderá solicitar múltiplas subpartições, usando um ou mais pontos de entrada.

Depois de solicitar uma subpartição, o servidor aguarda a chegada de uma mensagem. A origem dessa mensagem (identificador da tarefa que cede a subpartição) é usada para integrar a subpartição recebida na totalidade do sistema (caminho para a partição ascendente

ou original). Por cada partição, o servidor cria uma tarefa (associando-lhe a propriedade `tipo=partição`), cujo identificador é enviado à tarefa que lhe cedeu essa partição.

As tarefas que representam partições tratam quatro tipos de mensagens: mensagens de divisão, de identificação, de armazenamento e de pesquisa. Uma mensagem de divisão originará o fraccionamento da partição manipulada pela tarefa ou, no caso de o índice de *hash* não se encontrar dentro dos limites dessa partição, o reenvio da mensagem para outra tarefa. Uma mensagem de identificação é usada para integrar uma subpartição anteriormente cedida (caminho para a partição descendente). As mensagens de armazenamento e pesquisa (usadas para armazenar e recuperar informação) são encaminhados como se fossem mensagens de divisão, até alcançarem a partição/tarefa alvo.

A qualquer momento e em qualquer nó do *cluster* poderão ser activados novos servidores, sem preocupações de sincronização. É ainda possível manter múltiplos servidores por nó.

Funcionamento do cliente

Os clientes simplesmente enviam mensagens de armazenamento e pesquisa a tarefas escolhidas aleatoriamente e aguardam por respostas.

8.5.3 Valores de desempenho

A avaliação do desempenho deste sistema envolveu, numa fase inicial, os *subclusters* i686$^+$ e Athlon. Em cada nó foi activado um servidor para gerir 32 partições e foram executados entre 1 e 8 clientes, de maneira a avaliar a taxa máxima de recuperação de informação. Cada mensagem de pesquisa enviada por um cliente compreende 16bytes, podendo atravessar 14 tarefas (partições) antes de chegar à tarefa alvo. A resposta a uma dada pesquisa, contendo 8kbytes, é enviada directamente ao cliente pela tarefa alvo.

A figura 8.15 apresenta as taxas de operação (número de consultas por segundo), conseguidas em cada cliente, nos *clusters* i686$^+$ (gráfico do lado esquerdo) e Athlon (gráfico do lado direito). Naturalmente, para a mesma tecnologia de comunicação (M-VIA ou UDP com controladores SysKonnect), os clientes que executam em nós Athlon alcançam uma taxa de operação superior. No entanto, usando um único cliente por nó, o desempenho máximo é conseguido nos nós do *cluster* i686$^+$, através do subsistema GM, apesar de os processadores possuírem um desempenho significativamente inferior aos do *cluster* Athlon. Note-se que, com um número reduzido de clientes, o subsistema UDP permitiu alcançar um desempenho equivalente ou até superior ao conseguido com o M-VIA.

Numa segunda fase, foram usados conjuntamente os *subclusters* i686$^+$ e Athlon. A figura 8.16 mostra que as taxas de operação por cliente baixam, relativamente aos valores

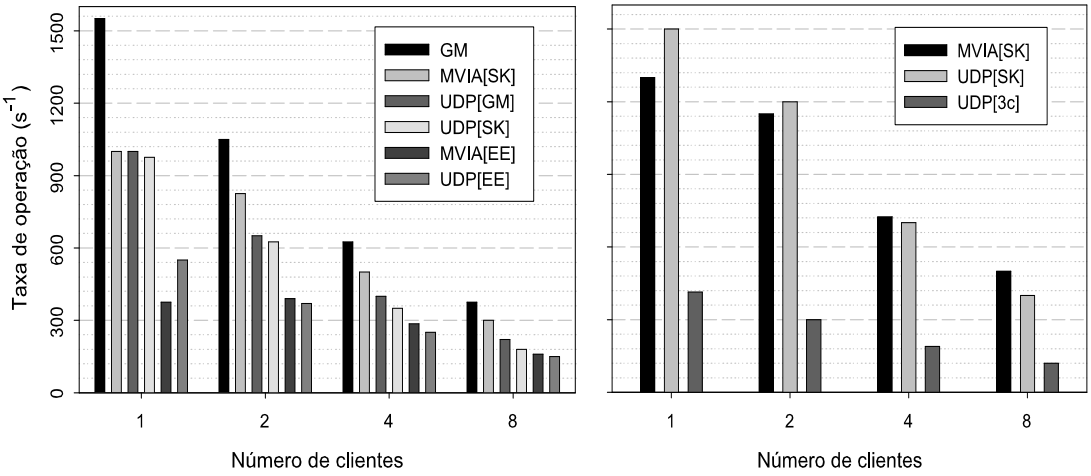


Figura 8.15: Taxas de operação de uma DHT (4 nós).

obtidos com apenas 4 nós. No entanto, a taxa de operação global alcançada pela totalidade dos nós aumenta; para o subsistema M-VIA, por exemplo, com 4 nós o sistema atinge 9600 pesquisas por segundo, enquanto que com 8 nós é possível atingir 16000 pesquisas por segundo. Isto significa que o sistema escala satisfatoriamente.

A conclusão mais importante a retirar desta análise diz respeito à interligação dos *subclusters* i686 e Athlon através da máquina do *subcluster* Xeon. A utilização conjunta das tecnologias Myrinet e Gigabit, a primeira no *cluster* i686 e a segunda no *cluster* Athlon, e o recurso à funcionalidade de reencaminhamento do *subcluster* Xeon, permitem alcançar os melhores resultados.

8.6 Escalamento de uma solução SMP

Algumas aplicações são facilmente modeladas utilizando múltiplos fios-de-execução, sem grandes preocupações de sincronização, principalmente quando está em causa um grande volume de operações de entrada/saída. Aplicações deste tipo têm ainda a vantagem de serem facilmente escaladas para um ambiente *cluster*, desde que existam mecanismos para o manuseamento de fios-de-execução remotos, como acontece no $m_{\varepsilon}\mu$.

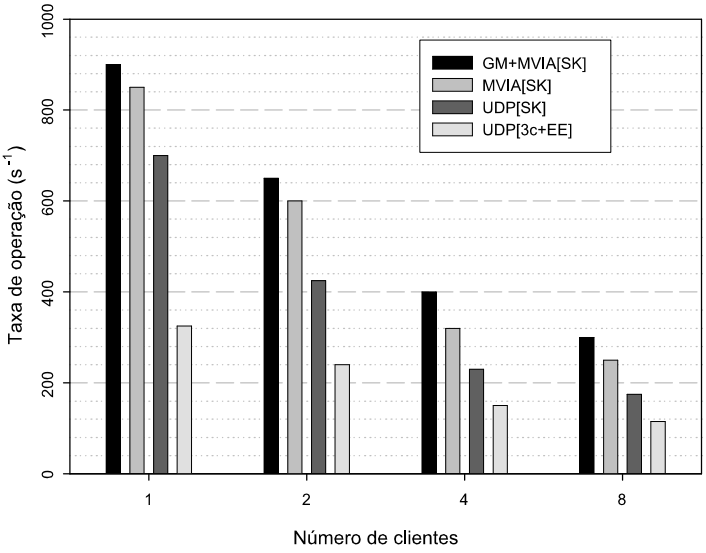


Figura 8.16: Taxa de operação de uma DHT (8 nós).

8.6.1 Visionamento de imagens de grandes dimensões

Com o intuito de avaliar a integração dos mecanismos de passagem de mensagens com as facilidades para a utilização de fios-de-execução do $m_{\varepsilon}\mu$, apresenta-se aqui uma aplicação destinada ao visionamento de imagens de grandes dimensões. As imagens que se pretende manusear correspondem a paisagens virtuais, geradas computacionalmente, que se encontram armazenadas como um conjunto de imagens mais pequenas (imagens JPEG com 640x480 pontos).

No caso de estudo analisado, pretendia-se visualizar paisagens com 9600x9600 pontos, constituídas por uma matriz de 15x20 imagens JPEG. O principal objectivo era a visualização de regiões arbitrárias das paisagens, implicando, eventualmente, o redimensionamento das imagens envolvidas.

A arquitectura proposta para manusear as paisagens em causa tenta dar resposta aos seguintes requisitos:

- elevado poder de cálculo, por forma a proceder ao redimensionamento das imagens;
- capacidade de armazenamento elevada, por forma a guardar o maior número possível de paisagens;
- boa largura de banda de entrada/saída, por forma a suportar o carregamento rápido das imagens, a partir do disco.

8.6.2 Desenho de uma solução SMP

Partindo do princípio que se dispõe de uma máquina SMP com recursos suficientes para o manuseamento das paisagens, pode ser desenvolvida uma solução baseada em múltiplos fios-de-execução, para acelerar a conversão de imagens JPEG em mapas de bits e o sequente redimensionamento desses mapas de bits.

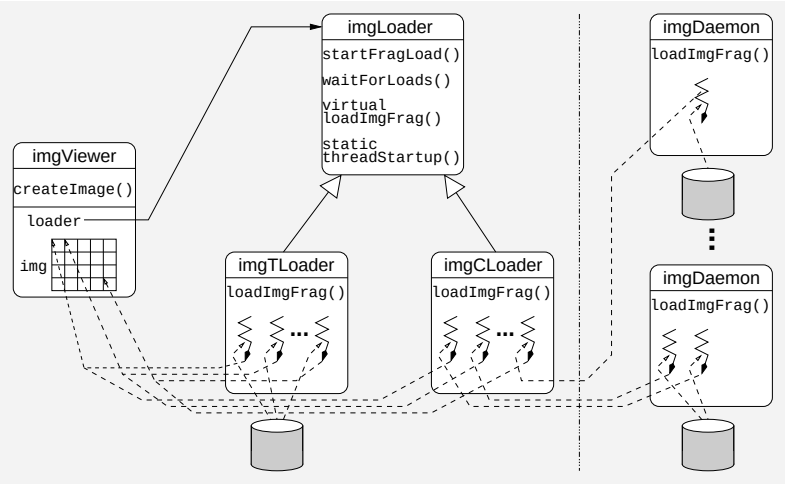


Figura 8.17: Modelo de objectos para o visionamento de paisagens.

A figura 8.17 mostra duas classes C++ usadas para modelar uma solução básica, baseada em múltiplos fios-de-execução. Um objecto `imgViewer` é usado para mostrar uma determinada região da paisagem, de acordo com um tamanho de janela especificado, após a criação do mapa de bits correspondente. O mapa de bits é criado recorrendo a um objecto `imgLoader`, o qual cria um fio-de-execução para a execução do método `startFragLoad`. O objecto `imgViewer` evoca o método `startFragLoad` da classe `imgLoader`, por cada imagem JPEG necessária para produzir o mapa de bits final.

Para mostrar a região de uma paisagem correspondente à totalidade dos seus pontos (9600x9600 pontos), por exemplo, são necessários 300 fios-de-execução para carregar as respectivas imagens JPEG, proceder à sua conversão para mapas de bits e no final efectuar o redimensionamento necessário. Se for usada um janela com 600x600 pontos, para mostrar a região da paisagem, cada fio-de-execução terá que reduzir, por um factor 16, uma imagem JPEG original, passando de uma imagem com 640x480 para um fragmento com 40x30 pontos. O objecto `imgViewer` é responsável por juntar os vários fragmentos produzidos ao nível dos fios-de-execução.

8.6.3 Escalamento de uma solução SMP

Assumindo agora que se dispõe de um *cluster* com suficiente espaço em disco, em cada um dos nós, é possível particionar a totalidade das imagens JPEG, cabendo a cada um dos nós a responsabilidade de armazenar, apenas, uma fracção da informação total. Desta forma, poder-se-ão ultrapassar limitações de armazenamento, que, eventualmente, surgiriam com a utilização de uma única máquina, bem como limitações de processamento, pelo facto de as máquinas SMP comuns possuírem um número de processadores reduzido. Note-se que o redimensionamento das múltiplas imagens, necessárias para um dado visionamento, é um caso evidente de exploração imediata de paralelismo, podendo ser entregue a cada nó do *cluster* a responsabilidade pelo redimensionamento de um dado subconjunto de imagens.

Pelo facto de as imagens se encontrarem distribuídas pelos vários nós, a largura de banda (relativa ao carregamento de imagens, a partir de disco) efectivamente explorada será a resultante da agregação das larguras de banda individuais, dos vários nós do *cluster*. Obviamente, será necessário um mecanismo de localização de imagens, isto é, será necessário determinar em que nó uma dada imagem está armazenada, mas isso poderá ser facilmente conseguido através da utilização de uma função de dispersão.

A figura 8.17 apresenta uma bateria de objectos `imgDaemon`, correspondentes a servidores em execução em cada um dos nós do *cluster*. Estes objectos destinam-se ao carregamento e transformação de imagens, de acordo com pedidos enviados a partir de um objecto `imgLoader` remoto. O objecto `imgLoader` usado em ambiente *cluster* solicita fragmentos de imagens a nós remotos do *cluster*, em vez de os obter, directamente, a partir do disco local.

A classe `imgLoader` é, na verdade, uma classe virtual usada para derivar duas classes:

1. `imgTLoader` – para criação de mapas de bits quando é usada uma única máquina SMP;
2. `imgCLoader` – para criação de mapas de bits em ambientes *cluster*, actuando como um representante.

Note-se que o desenvolvimento de uma solução multi-fio-de-execução, para ambiente *cluster*, assumindo que, previamente, tinha sido desenvolvida uma solução destinada à utilização numa única máquina SMP, torna-se uma tarefa simples, envolvendo as seguintes fases:

- uma classe virtual `imgLoader` é introduzida, por forma a permitir a utilização do mesmo objecto `imgViewer`;

- uma nova classe `imgCLoader` é derivada, para tratar pedidos e compilar a informação devolvida pelos fios-de-execução remotos;
- o código da classe `imgTLoader` destinado ao carregamento de imagens JPEG e ao seu redimensionamento é copiado para o servidor (objecto `imgDaemon`) e executado em cada nó do *cluster*.

Esta abordagem pode ser usada para escalar muitas aplicações multi-fio-de-execução desenvolvidas a pensar unicamente em máquinas SMP. A codificação destas aplicações, utilizando o $m_{\mathcal{E}}\mu$, é bastante simples, dado que existem primitivas para criação de tarefas remotas e existe a possibilidade de endereçar mensagens, directamente, a tarefas.

8.6.4 Análise da escalabilidade

Para comprovar a viabilidade desta abordagem, foram efectuados alguns testes, envolvendo o *subcluster* i686 e a única máquina do *subcluster* Xeon.

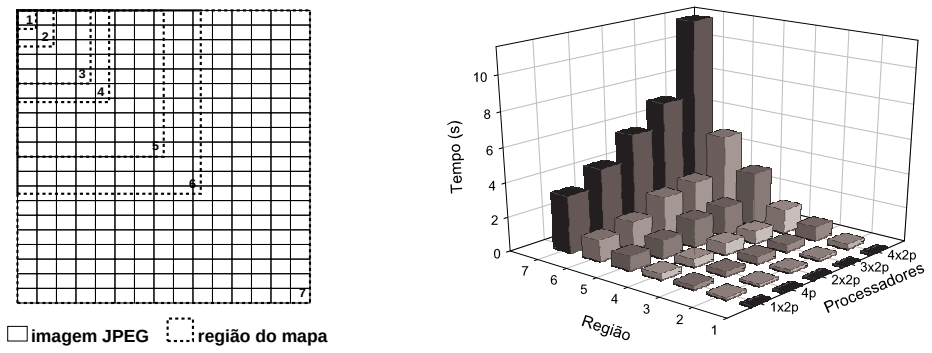


Figura 8.18: Desempenho da aplicação de visionamento em vários cenários.

A figura 8.18 apresenta os tempos necessários para o visionamento de 7 regiões distintas de uma paisagem, usando uma janela de 600x600 pontos. Do lado esquerdo da figura são apresentadas as 7 regiões em causa, as quais incluem desde 2 até 300 imagens JPEG. Estas regiões, identificadas de 1 a 7, correspondem a redimensionamentos de 1:1, 1:2, 1:4, 1:5, 1:8, 1:10 e 1:16, respectivamente. Do lado direito da figura são apresentados os resultados obtidos usando:

- unicamente a máquina Xeon, sendo usado um objecto `imgTLoader` pela aplicação (designação `4p`);
- de 1 a 4 nós do *cluster* i686, sendo usado um objecto `imgCLoader` pela aplicação (designações `1x2p`, `2x2p`, `3x2p` e `4x2p`).

É importante realçar que os resultados obtidos com a solução *cluster*, com base em 2 nós (4 processadores) já consegue ultrapassar os resultados obtidos com a máquina Xeon, a qual também inclui 4 processadores. A principal razão é a maior largura de banda disponível para carregamento de imagens JPEG, a partir de disco.

8.7 Epílogo

A avaliação do desempenho de uma plataforma de programação paralela é uma tarefa complexa. Para além da selecção ou implementação de aplicações apropriadas, para levar a cabo um conjunto relevante de testes, é necessário delimitar cuidadosamente o hardware onde o processo de avaliação é desencadeado. No caso da computação baseada em *clusters*, é possível definir variadas plataformas hardware, por variação do tipo de nós computacionais envolvidos e das tecnologias de interligação usadas.

O *cluster* utilizado nestes testes, apesar de corresponder a um sistema de pequena dimensão, permitiu chegar a conclusões relevantes respeitantes à funcionalidade do R_OCl e à forma como o $m_\varepsilon\mu$ explora essa funcionalidade. Foi possível verificar que, relativamente às bibliotecas de comunicação de baixo-nível, as sobrecargas impostas pelo R_OCl e pelo $m_\varepsilon\mu$ são mínimas, tendo em conta a funcionalidade acrescentada. Em particular, ficou comprovada a viabilidade da utilização de múltiplos fios-de-execução e da exploração de *clusters* heterogéneos ao nível da tecnologia de comunicação (*clusters* multi-SAN).

Foram ainda apresentadas três plataformas de avaliação de desempenho que vão muito para além dos tradicionais testes de largura de banda e tempo de ida-volta que normalmente se utilizam. Estas plataformas permitem captar a essência de muitas aplicações paralelas, podendo portanto ser usadas para obter informação de referência sobre o comportamento que certas aplicações poderão vir a ter, quando desenvolvidas com recurso ao $m_\varepsilon\mu$.

Para além dos valores de desempenho apresentados, que estão, obviamente, muito mais relacionados com o R_OCl do que com o $m_\varepsilon\mu$, poderia ainda analisar-se a aplicabilidade ou a conveniência de utilização do modelo $m_\varepsilon\mu$. No entanto, a modelação de uma aplicação e a correspondência entre recursos lógicos e físicos dependem do programador, o que dificulta uma análise objectiva.

Poderiam ainda ser apresentados valores relativos a algumas operações $m_\varepsilon\mu$ mais complexas, como por exemplo a criação de domínios agregadores. No entanto, por um lado, a complexidade da hierarquia de recursos é altamente dependente da aplicação e do programador e, por outro lado, a evocação de tais operações numa aplicação é efémera.

Capítulo 9

Discussão

O trabalho desenvolvido nesta dissertação usa a orientação ao recurso como base para a criação de num novo paradigma e de uma metodologia própria para o desenvolvimento de aplicações paralelas e a subjacente exploração de *clusters*.

A caminhar se faz o caminho, como diz o poeta. Da mesma forma, como corolário da investigação produzida, a validação e experimentação dos conceitos introduzidos ao longo deste largo caminho teve a sua concretização na plataforma a que convencionamos chamar $Ro_cme\mu$ e que representa o potencial combinado das bibliotecas R_oCl e $m_\varepsilon\mu$.

9.1 Comunicação orientada ao recurso

A orientação ao recurso encontra na imagem de sistema uno oferecida pelo R_oCl os níveis de abstracção necessários para a modelação e execução de aplicações complexas e altamente exigentes em termos de desempenho, permitindo uma efectiva exploração de *clusters*. Tais abstracções são particularmente importantes quando se utilizam *clusters* que integram múltiplos nós SMP e múltiplas tecnologias de comunicação SAN.

A biblioteca R_oCl, apesar de o seu desenho inicial ter sido orientado para a operação *intracluster*, também inclui funcionalidades específicas para a operação em ambientes *multicluster*, por via da hierarquização do serviço de directório e da inclusão do protocolo TCP. Convém, no entanto, distinguir a abordagem que no R_oCl conduziu a uma imagem de sistema uno minimalista suportada por um serviço de directório básico do gigantesco trabalho que tem vindo a ser conduzido no âmbito da *Grid*, não apenas em termos de objectivos como também de dimensão. No R_oCl pretende-se apenas dar resposta às questões de heterogeneidade relacionadas com as tecnologias de comunicação e, pontualmente, suportar a interligação de um número reduzido de *clusters* geograficamente dispersos.

Na sua concretização actual, o R_OCl oferece ao programador de sistema um interface único, que uniformiza o acesso às bibliotecas de comunicação de baixo-nível GM e M-VIA, mas que permite tirar partido das especificidades de cada uma delas. Na presença de nós multiconectados, a operação combinada das duas bibliotecas de comunicação pode, em muitos casos, vir a traduzir-se em níveis de desempenho superiores, como atestam as experiências realizadas. A arquitectura R_OCl e a orientação ao recurso são suficientemente flexíveis para vir a incorporar novos subsistemas de comunicação, o que constitui uma mais valia importante num contexto de constante evolução tecnológica.

O suporte à programação com múltiplos fios-de-execução, mantendo os níveis de desempenho do hardware de comunicação de um *cluster*, constituiu a orientação principal para o desenvolvimento do R_OCl. Oferecendo ao programador o controlo efectivo do potencial de computação e comunicação de um *cluster*, a construção de aplicações que maximizam a utilização dos recursos físicos de um *cluster* SMP multi-SAN poderá ser uma realidade. Através de um número significativo de testes, pôde-se comprovar a adequação do modelo a cenários variados e, em particular, mostrou-se ser promissora a utilização de múltiplos fios-de-execução, com base na biblioteca NPTL, em ambientes que exploram tecnologias de comunicação de elevado desempenho através de bibliotecas de comunicação de baixo-nível.

9.2 Modelação e exploração unificadas

Com base na funcionalidade do R_OCl, foram criadas abstracções de alto-nível, que apesar de mais convenientes, não impedem o programador de recorrer a uma utilização informada dos recursos físicos de um *cluster*. Isto significa que o programador pode fazer uma gestão de recursos mais próxima do hardware da arquitectura alvo, se assim o entender, por forma a garantir níveis elevados de desempenho na execução de certas aplicações que, através de abordagens baseadas em imagens de sistema uno, não tiram partido efectivo do potencial disponível. Por outras palavras, o $m_{\varepsilon}\mu$ expõe a hierarquia de recursos de um *cluster* SMP multi-SAN ao programador, possibilitando a exploração dos múltiplos níveis de localidade.

Neste contexto, as abstracções idealizadas e materializadas na biblioteca $m_{\varepsilon}\mu$ destinam-se à modelação de aplicações, tendo em vista uma adaptação óptima das entidades lógicas ao equipamento constituinte do *cluster*, inclusivamente em cenários multi-aplicação e multi--utilizador. O programador desenha a aplicação e posteriormente escreve o código destinado à criação dos vários componentes lógicos dessa aplicação, tendo sempre em consideração a organização dos recursos físicos do *cluster*. A abordagem garante a possibilidade de se definirem diferentes visões (perspectivas) da organização dos recursos físicos; quando o desempenho não é o principal requisito, o programador pode desprezar a forma como estão organizados os recursos físicos, assumindo, por exemplo, que o *cluster* se comporta,

simplesmente, como uma máquina com muitos processadores.

A definição de um conjunto básico de abstracções para a manipulação tanto dos recursos físicos de um *cluster* como das entidades lógicas de uma aplicação conferem ao $m_{\mathcal{E}}\mu$ características únicas. A orientação ao recurso permitiu unificar a modelação de aplicações e a exploração de recursos físicos, fundindo num único conceito entidades lógicas e físicas. Isto possibilita, por uma lado, uma implementação mais simples de toda a plataforma e, por outro lado, a redução dos tempos de aprendizagem, por parte de novos utilizadores.

Ao longo da dissertação apresentam-se exemplos que mostram de que forma os conceitos e abstracções do paradigma podem ser usados, pelo programador, para criar e executar aplicações que recorrem aos paradigmas tradicionais da memória partilhada, passagem de mensagens e memória global.

9.3 Sinopse

A exploração eficiente e conveniente de *clusters* SMP multi-SAN foi apontada, no capítulo 1, como um dos objectivos a atingir no âmbito deste trabalho. Face à constatação, no capítulo 2, da inexistência de mecanismos adequados para esse efeito, do ponto de vista da eficiência, pretendia-se *(i)* fornecer os meios necessários para as aplicações tirarem o máximo partido das diferentes tecnologias de comunicação e *(ii)* explorar os diferentes padrões de localidade existentes na hierarquia de recursos que resulta da utilização de múltiplas tecnologias SAN e múltiplos nós SMP. Quanto à conveniência, pretendia-se compatibilizar os paradigmas tradicionais de programação paralela, com a possibilidade de garantir a cooperação entre aplicações em ambientes multi-utilizador, multi-aplicação e multi-fio-de-execução.

Neste contexto, esta dissertação apresenta: uma abordagem inovadora para a exploração dos recursos físicos de um *cluster* (capítulo 3), novas abstracções para a modelação de aplicações (capítulo 4), um novo modelo de comunicação adequado a tecnologias de elevado desempenho (capítulos 5 e 6) e um interface simples para o desenvolvimento e execução de aplicações paralelas/distribuídas (capítulo 7). Usando, como indicadores, a análise de desempenho apresentada no capítulo 8 e os exemplos de modelação presentes ao longo da dissertação, conclui-se que o $Ro_cme\mu$ é uma resposta qualificada aos objectivos enunciados.

9.4 Perspectivas

A *Grid* tem vindo a conquistar um lugar importante, senão o mais importante, na área do computação paralela/distribuída. Neste contexto, o projecto Globus tem sido lugar

de encontro e de motivação para muitos investigadores que antes se debruçavam sobre os problemas da computação baseada em *clusters*. A metodologia proposta nesta dissertação, integrando recursos lógicos e físicos, poderia, de alguma forma, ser aplicada à *Grid*. Todavia, há ainda um longo caminho a percorrer, no sentido de redesenhar o sistema de registo e armazenamento de informação relativa a recursos, para que tal seja possível. O maior desafio seria a adaptação do serviço de directório do R_OCl à dimensão exigida para gerir a informação associada à *Grid*.

Alguns dos conceitos do $m_\varepsilon\mu$, nomeadamente a definição de pseudónimos e os mecanismos de partilha de propriedades, perspectivam-se como podendo vir a ser úteis quando aplicados a outras áreas de conhecimento. No entanto, apenas a formalização matemática das relações de uma hierarquia $m_\varepsilon\mu$ poderá conduzir à sua aplicação em problemas que actualmente são tratados através de técnicas amplamente divulgadas e testadas.

A adaptação das bibliotecas R_OCl e $m_\varepsilon\mu$, de maneira a permitir a sua utilização no desenvolvimento de aplicações paralelas em Java, constitui um potencial objectivo a médio prazo. Tal não se trataria de uma mera reimplementação, sendo eventualmente necessário redesenhar alguns componentes ou até incluir novas abstracções, dado que o Java já possui alguns mecanismos para suporte à programação concorrente/distribuída. No entanto, a conjugação da elevada portabilidade do Java com a flexibilidade do $m_\varepsilon\mu$ para a manipulação de recursos resultaria numa máquina virtual poderosa, para a exploração de *clusters*.

Noutra direcção, os resultados obtidos ao longo da investigação poderão ser usados em aulas, para a apresentação de conceitos relativos ao desenvolvimento e execução de aplicações paralelas e, essencialmente, para a demonstração de técnicas e metodologias que permitem uma exploração efectiva de *clusters* que combinam múltiplas tecnologias de comunicação. Neste âmbito, poderão ainda ser retiradas conclusões mais específicas sobre a conveniência de algumas abstracções propostas, tendo em conta o processo de aprendizagem de um novo utilizador, a produtividade e a qualidade das soluções alcançadas.

Finalmente, a consolidação de conceitos e do ambiente de programação desenvolvido, nomeadamente no que diz respeito à optimização de alguns componentes, melhoramento da sua robustez e validação da sua aplicabilidade num conjunto mais alargado de casos de estudo, será o futuro mais provável. Em particular, perspectiva-se a inclusão de suporte para a tecnologia Infiniband num futuro próximo.

Bibliografia

[10GEA 02] 10 Gigabit Ethernet Alliance. *10 Gigabit Ethernet*. Technology Overview White Paper, 2002.

[Abt 02] Bill Abt, Saurabh Desai, David Howell & Dave McCracken. *Next Generation POSIX Threading - Moving Linux to the Enterprise*. IBM, 2002.

[Adve 96] Sarita Adve & Kourosh Gharachorloo. *Shared Memory Consistency Models: A Tutorial*. Computer, vol. 29, no. 12, pp. 66–76, 1996.

[Alves 02a] Albano Alves, António Pina, José Exposto & José Rufino. *High Performance Multithreaded Message Passing on a Myrinet Cluster*. In 7^{th} International Conference on Applications of High-Performance Computers in Engineering (HPC'02), pp. 241–250. WITpress, 2002.

[Alves 02b] Albano Alves, António Pina, José Exposto & José Rufino. *Scalable Multithreading in a Low Latency Myrinet Cluster*. In 5^{th} International Meeting on High Performance Computing for Computational Science (VECPAR'02) - Selected Papers and Invited Talks, no. 2565 in LNCS, pp. 579–592. Springer, 2002.

[Alves 03a] Albano Alves, António Pina, José Exposto & José Rufino. *Evaluating Applications Performance in a Multi-networked Cluster*. In 15^{th} IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'03), pp. 375–380. ACTA Press, 2003.

[Alves 03b] Albano Alves, António Pina, José Exposto & José Rufino. *RoCL: a Resource oriented Communication Library*. In Euro-Par 2003, no. 2790 in LNCS, pp. 969–979. Springer, 2003.

[Alves 03c] Albano Alves, António Pina, José Exposto & José Rufino. *ToCL: a Thread Oriented Communication Library to Interface VIA and GM Protocols*. In 3^{rd} International Conference on Computational Science (ICCS'03), no. 2658 in LNCS, pp. 1022–1031. Springer, 2003.

[Alves 04a] Albano Alves, António Pina, José Exposto & José Rufino. *Deploying Applications in Multi-SAN SMP Clusters.* In 5^{th} Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS'04), pp. 63–70. Springer, 2004.

[Alves 04b] Albano Alves, António Pina, José Exposto & José Rufino. *Mapping application-level components into hierarchical systems resources.* In 10^{th} International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'04), pp. 246–252. CSREA Press, 2004.

[Alves 04c] Albano Alves, António Pina, José Exposto & José Rufino. *$m_{\varepsilon}\mu$: unifying application modeling and cluster exploitation.* In 16^{th} Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'04), pp. 132–139. IEEE Computer Society, 2004.

[Amza 96] Christiana Amza, Alan Cox, Sandhya Dwarkadas, Pete Keleher, Honghui Lu, Ramakhrishnan Rajamony, Weimin Yu & Willy Zwaenepoel. *TreadMarks: Shared Memory Computing on Networks of Workstations.* Computer, vol. 29, no. 2, pp. 18–28, 1996.

[Aumage 00] Olivier Aumage, Luc Bougé, Alexandre Denis, Jean-François Méhaut, Guillaume Mercier, Raymond Namyst & Loïc Prylli. *Madeleine II: A Portable and Efficient Communication Library for High-Performance Cluster Computing.* In 2^{nd} IEEE International Conference on Cluster Computing (CLUSTER'00), pp. 78–87, 2000.

[Baden 00] Scott Baden & Stephen Fink. *A Programming Methodology for Dual-tier Multicomputers.* IEEE Transactions on Software Engineering, vol. 26, no. 3, pp. 212–226, 2000.

[Bader 99] David Bader & Joseph JáJá. *SIMPLE: A Methodology for Programming High Performance Algorithms on Clusters of Symmetric Multiprocessors (SMPs).* Journal of Parallel and Distributed Computing, vol. 58, no. 1, pp. 92–108, 1999.

[Baker 00] Mark Baker (editor). *Cluster Computing White Paper.* IEEE Task Force on Cluster Computing, 2000.

[Bal 99] Henri Bal, Aske Plaat, Thilo Kielmann, Jason Maassen, Rob Von Nieuwpoort & Ronald Veldema. *Parallel Computing on Wide-Area Clusters: the Albatross Project.* In 2^{nd} Extreme Linux Workshop - USENIX Annual Technical Conference, pp. 20–24, 1999.

[Banikazemi 01] Mohammad Banikazemi, Jiuxing Liu, Dhabaleswar Panda & Ponnuswamy Sadayappan. *Implementing TreadMarks over VIA on Myrinet and Gigabit Ethernet: Challenges, Design Experience, and Performance Evaluation.* In 30^{th} International Conference on Parallel Processing (ICPP'01), pp. 167–174, 2001.

[Barak 98] Amnon Barak & Oren La'adan. *The MOSIX Multicomputer Operating System for High Performance Cluster Computing.* Future Generation Computer Systems, vol. 13, no. 4-5, pp. 361–372, 1998.

[Barak 99] Amnom Barak, Ilia Gilderman & Igor Metrik. *Performance of the Communication Layers of TCP/IP with the Myrinet Gigabit LAN.* Computer Communications, vol. 22, no. 11, 1999.

[Baratloo 96] Arash Baratloo, Mehmet Karaul, Zvi Kedem & Peter Wyckoff. *Charlotte: Metacomputing on the Web.* In 9^{th} IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'96), 1996.

[Barreto 00] Marcos Barreto, Rafael Ávila & Philippe Navaux. *The MultiCluster Model to the Integrated Use of Multiple Workstation Clusters.* In International Workshop on Personal Computer based Networks Of Workstations (PC-NOW'00), pp. 71–80, 2000.

[Beck 99] Micah Beck, Jack Dongarra, Graham Fagg, Al Geist, Paul Gray, James Kohl, Mauro Migliardi, Keith Moore, Terry Moore, Philip Papadopoulous, Stephen Scott & Vaidy Sunderam. *HARNESS: A Next Generation Distributed Virtual Machine.* Future Generation Computer Systems, vol. 15, no. 5-6, pp. 571–582, 1999.

[Becker 95] Donald Becker, Thomas Sterling, Daniel Savarese, John Dorband, Udaya Ranawak & Charles Packer. *BEOWULF: A Parallel Workstation for Scientific Computation.* In 24^{th} International Conference on Parallel Processing (ICPP'95), pp. 11–14, 1995.

[Begel 02] Andrew Begel, Philip Buonadonna, David Culler & David Gay. *An Analysis of VI Architecture Primitives in Support of Parallel and Distributed Communication.* Concurrency and Computation: Practice and Experience, vol. 14, no. 1, pp. 55–76, 2002.

[Bershad 93] Brian Bershad, Matthew Zekauskas & Wayne Sawdon. *The Midway Distributed Shared Memory System.* In IEEE Computer Conference (CompCon'93), pp. 528–537, 1993.

[Betz 02] Anthony Betz & Paul Gray. *Gigabit Over Copper Evaluation.* http://www.cs.uni.edu/~gray/gig-over-copper/gig-over-copper.html, 2002.

[Bhoedjang 96] Raoul Bhoedjang & Koen Langendoen. *Friendly and Efficient Message Handling.* In 29^{th} Hawaii International Conference of System Sciences (HICSS'96), pp. 121–130, 1996.

[Bhoedjang 98a] Raoul Bhoedjang, John Romein & Henri Bal. *Optimizing Distributed Data Structures Using Application-Specific Network Interface Software.* In 27^{th} International Conference on Parallel Processing (ICPP'98), pp. 485–492, 1998.

[Bhoedjang 98b] Raoul Bhoedjang, Tim Rühl & Henri Bal. *LFC: A Communication Substrate for Myrinet.* In 4^{th} Annual Conference of the Advanced School for Computing and Imaging (ASCI'98), 1998.

[Blumofe 95] Robert Blumofe, Christopher Joerg, Bradley Kuszmaul, Charles Leiserson, Keith Randall & Yuli Zhou. *Cilk: An Efficient Multithreaded Runtime System.* In 5^{th} Symposium on Principles and Practice of Parallel Programming (PPoPP'95), pp. 207–216, 1995.

[Boden 95] Nanette Boden, Danny Cohen, Robert Felderman, Alan Kulawik, Charles Seitz, Jakov Seizovic & Wen-King Su. *Myrinet: A Gigabit-per-Second Local-Area Network.* IEEE Micro, vol. 15, no. 1, pp. 29–36, 1995.

[Bougé 98] Luc Bougé, Jean-François Méhaut & Raymond Namyst. *Madeleine: an effcient and portable communication interface for multithreaded environments.* In 7^{th} International Conference on Parallel Architectures and Compilation Techniques (PACT'98), pp. 240–247, 1998.

[Bozeman 99] Patrick Bozeman & Bill Saphir. *A Modular High Performance Implementation of the Virtual Interface Architecture.* In 2^{nd} Extreme Linux Workshop - USENIX Annual Technical Conference, 1999.

[Brightwell 00] Ron Brightwell & Arthur Maccabe. *Scalability Limitations of VIA-Based Technologies in Supporting MPI.* In 4^{th} MPI Developer's and User's Conference (MPIDC'00), 2000.

[Brin 98] Sergey Brin & Lawrence Page. *The Anatomy of a Large-Scale Hypertextual Web Search Engine.* Computer Networks and ISDN Systems, vol. 30, no. 1-7, pp. 107–117, 1998.

[Brune 97] Matthias Brune, Jörn Gehring & Alexander Reinefeld. *Heterogeneous Message Passing and a Link to Resource Management.* Journal of Supercomputing, vol. 11, pp. 1–17, 1997.

[Brune 99] Matthias Brune, Alexander Reinefeld & Jorg Varnholt. *A Resource Description Environment for Distributed Computing Systems.* In 8^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC'99), pp. 279–286, 1999.

[Bucur 03] Anca Bucur & Dick Epema. *Trace-Based Simulations of Processor Co-Allocation Policies in Multiclusters.* In 12^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC'03), pp. 70–79, 2003.

[Buntinas 01] Darius Buntinas, Dhabaleswar Panda & Ponnuswamy Sadayappan. *Fast NIC-Based Barrier over Myrinet/GM.* In 15^{th} International Parallel and Distributed Processing Symposium (IPDPS-01), 2001.

[Buyya 01] Rajkumar Buyya, Toni Cortes & Hai Jin. *Single System Image (SSI).* Journal of High-Performance Computing Applications, vol. 15, no. 2, pp. 124–135, 2001.

[Cappello 00] Franck Cappello, Olivier Richard & Daniel Etiemble. *Investigating the performance of two programming models for clusters of SMP PCs.* In 6^{th} International Symposium on High-Performance Computer Architecture (HPCA'00), pp. 349–359, 2000.

[Cassali 00] Ricardo Cassali, Marcos Barreto, Rafael Ávila & Philippe Navaux. *Group Communication Service for DECK.* In XXVI Conferencia Latinoamericana de Informatica (CLEI'00), 2000.

[Castro 02] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec & Antony Rowstron. *SCRIBE: A large-scale and decentralized application-level multicast infrastruture.* IEEE Journal on Selected Areas in Communications, vol. XX, no. Y, 2002.

[Chen 00] Helen Chen & Pete Wyckoff. *Simulation studies of Gigabit ethernet versus Myrinet using real application cores.* In 4^{th} Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing (CANPC'00), pp. 130–144, 2000.

[Chen 02] Yu Chen, Xiaoge Wang, Zhenqiang Jiao, Jun Xie, Zhihui Du & Sanli Li. *MyVIA: A Design and Implementation of the High Performance Virtual*

Interface Architecture. In 4^{th} IEEE International Conference on Cluster Computing (CLUSTER'02), pp. 160–170, 2002.

[Chiu 02] Kenneth Chiu, Madhusudhan Govindaraju & Dennis Gannon. *The Proteus Multiprotocol Message Library*. In IEEE/ACM Supercomputing Conference (SC'02), 2002.

[Cho 02] Junghoo Cho & Hector Garcia-Molina. *Parallel Crawlers*. In 11^{th} International World Wide Web Conference (WWW'02), 2002.

[Chowdappa 94] Aswini Chowdappa, Anthony Skjellum & Nathan Doss. *Thread-Safe Message Passing With P4 and MPI*. Technical report TR-CS-941025, Computer Science Department and NSF Engineering Research Center, 1994.

[Ciaccio 99] Giuseppe Ciaccio. *A Communication System for Efficient Parallel Processing on Clusters of Personal Computers*. PhD thesis, Università di Genova, 1999.

[Clariion 98] CLARiiON Corporation, Dell Computer Corporation, GigaNet, IBM Corporation, Intel Corporation, qLogic Corporation & Visual Insights. *Demonstrating the Benefits of Virtual Interface Architecture*, 1998.

[Cohen 98] William Cohen, Charlie Patel & Aravind Seshagiri. *Cost of User and Kernel Level Threads Operations on Linux*, 1998.

[Compaq 97] Compaq Computer Corporation, Intel Corporation & Microsoft Corporation. *Virtual Interface Architecture Specification*. Version 1.0, 1997.

[Coulaud 95] Olivier Coulaud & Eric Dillon. *Para++: C++ Bindings for Message Passing Libraries - User Guide*. Technical report, INRIA, 1995.

[Czajkowski 98] Karl Czajkowski, Ian Foster, Nicholas Karonis, Carl Kesselman, Stuart Martin, Warren Smith & Steven Tuecke. *A Resource Management Architecture for Metacomputing Systems*. In 4^{th} Workshop on Job Scheduling Strategies for Parallel Processing, pp. 62–82, 1998.

[Czajkowski 01] Karl Czajkowski, Steven Fitzgerald, Ian Foster & Carl Kesselman. *Grid Information Services for Distributed Resource Sharing*. In 10^{th} IEEE International Symposium on High-Performance Distributed Computing (HPDC'01), pp. 181–194, 2001.

[Dagum 98] Leonardo Dagum & Ramesh Menon. *OpenMP: A Proposed Industry Standard API for Shared Memory Programming.* IEEE Computational Science & Engineering, vol. 5, no. 1, 1998.

[Damani 97] Om Damani, Pi-Yu Chung, Yennun Huang, Chandra Kintala & Yi-Min Wang. *ONE-IP: Techniques for Hosting a Service on a Cluster of Machines.* In 6^{th} International World Wide Web Conference (WWW'97), 1997.

[Damianakis 97] Stefanos Damianakis, Yuqun Chen & Edward Felten. *Reducing Waiting Costs in User-Level Communication.* In 11^{th} International Parallel Processing Symposium (IPPS'97), 1997.

[Danjean 00] Vincent Danjean, Raymond Namyst & Robert Russel. *Linux Kernel Activations to Support Multithreading.* In 18^{th} IASTED International Conference on Applied Informatics (AI'00), pp. 718–723, 2000.

[de Oliveira 00] Fábio de Oliveira, Marcos Barreto, Rafael Ávila & Philippe Navaux. *A Comparative Study on Low-level APIs for Myrinet and SCI-based Clusters.* In 4^{th} World Multiconference on Systematics, Cybernetics and Informatics (SCI'00), 2000.

[de Oliveira 01] Fábio de Oliveira, Rafael Ávila, Marcos Barreto, Philippe Navaux & César De Rose. *DECK-SCI: High-Performance Communication and Multithreading for SCI Clusters.* In 3^{rd} IEEE International Conference on Cluster Computing (CLUSTER'01), pp. 372–379, 2001.

[Dedu 00] Eugen Dedu, Claude Timsit & Stéphane Vialle. *Comparison of OpenMP and Classical Multi-Threading Parallelization for Regular and Irregular Algorithms.* In 1^{st} International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing (SNPD'00), 2000.

[Dolphin 96] Dolphin Interconnect Solutions. *The Dolphin SCI Interconnect.* White Paper, 1996.

[dos Santos 99] Rodrigo dos Santos, Ricardo Bianchini & Claudio Amorim. *A Survey of Messaging Software Issues and Systems for Myrinet-Based Clusters.* Parallel and Distributed Computing Practices, special issue on High-Performance Computing on Clusters, vol. 2, no. 2, pp. 133–146, 1999.

[Drepper 03] Ulrich Drepper & Ingo Molnar. *The Native POSIX Thread Library for Linux*, 2003.

[Dubnicki 97] Cezary Dubnicki, Angelos Bilas, Kai Li & James Philbin. *Design and Implementation of Virtual Memory-Mapped Communication on Myrinet.* In 11^{th} International Parallel Processing Symposium (IPPS'97), pp. 388–396, 1997.

[Eisenhauer 98] Greg Eisenhauer, Bech Schroeder & Karsten Schwan. *DataExchange: High Performance Communications in Distributed Laboratories.* Parallel Computing, vol. 24, no. 12-13, pp. 1713–1733, 1998.

[Engelschall 00] Ralf Engelschall. *Portable Multithreading - The Signal Stack Trick For User-Space Thread Creation.* In USENIX Annual Technical Conference, 2000.

[Espenica 02] Roberto Espenica & Pedro Medeiros. *Porting PVM to the VIA architecture using a fast communication library.* In 9^{th} European PVM/MPI Users Group Meeting (EuroPVM/MPI'02), pp. 341–348, 2002.

[Etsion 99] Yoav Etsion, Mickael Raizman & Dror Feitelson. *Topology and Routing in Clusters: From Theory to Practice.* Technical report, The Hebrew University - Parallel Systems Lab, 1999.

[Farrell 00a] Paul Farrell & Hong Ong. *Communication Performance over a Gigabit Ethernet Network.* In 19^{th} IEEE International Performance, Computing, and Communications Conference (IPCCC'00), pp. 181–189, 2000.

[Farrell 00b] Paul Farrell, Hong Ong & Stephen Scott. *Enabling High Performance Data Transfer on Cluster Architecture.* In 2^{nd} IEEE International Conference on Cluster Computing (CLUSTER'00), pp. 347–348, 2000.

[Felten 92] Edward Felten & Dylan McNamee. *Improving the Performance of Message-Passing Applications by Multithreading.* In Scalable High Performance Computing Conference (SHPCC'92), pp. 84–89, 1992.

[Ferrari 95] Adam Ferrari & Vaidy Sunderam. *TPVM: Distributed Concurrent Computing with Lightweight Processes.* In 4^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC'95), 1995.

[Finucane 02] David Finucane. *Myrinet Mapping and Monitoring.* Myrinet User's Group Conference (MUG'02), 2002.

[Fischer 02] Markus Fischer. *Sockets-GM - Mapping Distributed Applications to Myrinet.* Myrinet User's Group Conference (MUG'02), 2002.

[Fitzgerald 97] Steven Fitzgerald, Ian Foster, Carl Kesselman, Gregor von Laszewski, Warren Smith & Steven Tuecke. *A Directory Service for Configuring High-Performance Distributed Computations*. In 6^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC'97), 1997.

[Flanery 00] Ray Flanery, Al Geist, Brian Luethke & Stephen Scott. *Cluster Command & Control (C3) Tools Suite*. In 3^{th} Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS'00), 2000.

[Foster 94] Ian Foster, Carl Kesselman & Steven Tuecke. *The Nexus Task-Parallel Runtime System*. In 1^{st} International Workshop on Parallel Processing (IWPP'94), pp. 457–462, 1994.

[Foster 97] Ian Foster & Carl Kesselman. *Globus: A Metacomputing Infrastructure Toolkit*. The International Journal of Supercomputer Applications and High Performance Computing, vol. 11, no. 2, pp. 115–128, 1997.

[Foster 98] Ian Foster & Nicholas Karonis. *A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems*. In IEEE/ACM Supercomputing Conference (SC'98), 1998.

[Frachten. 02] Eitan Frachtenberg, Fabrizio Petrini, Juan Fernandez, Scott Pakin & Salvador Coll. *STORM: Lightning-Fast Resource Management*. In IEEE/ACM Supercomputing Conference (SC'02), 2002.

[Frank 93] Matthew Frank & Mary Vernon. *A Hybrid Shared Memory/Message Passing Parallel Machine*. In 22^{th} International Conference on Parallel Processing (ICPP'93), pp. 232–237, 1993.

[Gallard 02] Pascal Gallard, Christine Morin & Renaud Lottiaux. *Dynamic Resource Management in a Cluster for High-Availability*. In Euro-Par 2002, pp. 589–592, 2002.

[Garcia 00] Felix Garcia & Javier Fernandez. *POSIX Thread Libraries*. Linux Journal, vol. 2000, no. 70es, 2000.

[Geist 94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek & Vaidy Sunderam. PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, 1994.

[Geist 96] Al Geist, James Kohl & Philip Papadopoulos. *PVM and MPI: a Comparison of Features*. Calculateurs Paralleles, vol. 8, no. 2, pp. 137–150, 1996.

[Geist 00] Al Geist, Stephen Scott & Jens Schwidder. *M3C - An Architecture for Monitoring and Managing Multiple Clusters.* In 12^{th} IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'00), 2000.

[Geoffray 99] Patrick Geoffray, Loïc Prylli & Bernard Tourancheau. *BIP-SMP : High Performance Message Passing over a Cluster of Commodity SMPs.* In IEEE/ACM Supercomputing Conference (SC'99), 1999.

[Geoffray 01] Patrick Geoffray, CongDuc Pham, Loïc Prylli, Bernard Tourancheau & Roland Westrelin. *Protocols and Software for Exploiting Myrinet Clusters.* In 1^{st} International Conference on Computational Science (ICCS'01), pp. 233–242, 2001.

[Geoffray 02] Patrick Geoffray. *MPICH-GM and VI-GM middlewares.* Myrinet User's Group Conference (MUG'02), 2002.

[George 00] William George, John Hagedorn & Judith Devaney. *IMPI: Making MPI Interoperable.* Journal of Research of the National Institute of Standands and Technology, vol. 105, no. 3, pp. 343–428, 2000.

[Ghormley 98] Douglas Ghormley, David Petrou, Steven Rodrigues, Amin Vahdat & Thomas Anderson. *GLUnix: a Global Layer Unix for a Network of Workstations.* Software Practice and Experience, vol. 28, no. 9, pp. 929–961, 1998.

[Giannini 98] Louis Giannini & Andrew Chien. *A Software Architecture for Global Address Space Communication on Clusters: Put/Get on Fast Messages.* In 7^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC'98), pp. 330–339, 1998.

[Gribble 00] Steven Gribble, Eric Brewer, Joseph Hellerstein & David Culler. *Scalable, Distributed Data Structures for Internet Service Construction.* In 4^{th} Symposium on Operating System Design and Implementation (OSDI'00), pp. 319–332, 2000.

[Gropp 95] William Gropp & Ewing Lusk. *A Taxonomy of Programming Models for Symmetric Multiprocessors and SMP Clusters.* In Programming Models for Massively Parallel Computers (PMMP'95), pp. 2–9, 1995.

[Gropp 96] William Gropp, Ewing Lusk, Nathan Doss & Anthony Skjellum. *A High-Performance, Portable Implementation of the MPI Message Pas-*

sing Interface Standard. Parallel Computing, vol. 22, no. 6, pp. 789–828, 1996.

[Gropp 98] William Gropp & Ewing Lusk. *PVM and MPI Are Completely Different*. Future Generation Computer Systems, 1998.

[Gusella 90] Riccardo Gusella. *A measurement study of diskless workstation traffic on an Ethernet*. IEEE Transactions on Communications, vol. 38, no. 9, pp. 1557–1568, 1990.

[Gustafson 88] John Gustafson. *Reevaluating Amdahl's Law*. Communications of the ACM, vol. 31, no. 5, pp. 532–533, 1988.

[Haines 94] Matthew Haines, David Cronk & Piyush Mehrotra. *On the Design of Chant: A Talking Threads Package*. In IEEE/ACM Supercomputing Conference (SC'94), pp. 350–359, 1994.

[Hansen 00] Jørgen Hansen & Eric Jul. *Latency Reduction using a Polling Scheduler*. In 2^{nd} Workshop on Cluster-Based Computing (WCBC'00), pp. 27–31, 2000.

[He 02] Yun He & Chris Ding. *MPI and OpenMP Paradigms on Cluster of SMP Architectures: the Vacancy Tracking Algorithm for Multi-Dimensional Array Transposition*. In IEEE/ACM Supercomputing Conference (SC'02), pp. 1–14, 2002.

[Hill 98] Jonathan Hill, Bill McColl, Dan Stefanescu, Mark Goudreau, Kevin Lang, Satish Rao, Tornsten Suel, Thanasis Tsantilas & Rob Bisseling. *BSPlib: The BSP Programming Library*. Parallel Computing, vol. 24, no. 14, pp. 1947–1980, 1998.

[Hori 99] Atsushi Hori, Hiroshi Tezuka & Yutaka Ishikawa. *An Implementation of Parallel Operating System for Clustered Commodity Computers*. In USENIX 99, 1999.

[Hunt 98] Guerney Hunt, Germán Goldszmidt, Richard King & Rajat Mukherjee. *Network Dispatcher: a connection router for scalable Internet services*. Computer Networks and ISDN Systems, vol. 30, no. 1-7, pp. 347–357, 1998.

[Huse 99] Lars Huse. *Collective Communication on Dedicated Clusters of Workstations*. In 6^{th} European PVM/MPI Users Group Meeting (EuroPVM/MPI'99), pp. 469–476, 1999.

[Huss-Led. 97] Steve Huss-Lederman (Editor). *MPI-2: Extensions to the Message-Passing Interface*. Technical report, MPI Forum, 1997.

[IBTA 02] InfiniBand Trade Association. *InfiniBand Architecture Specification*. Volume 1, Release 1.1, 2002.

[Intel 98] Intel Corporation. *Intel Virtual Interface (VI) Architecture - Conformance Suite User's Guide*. Preliminary Version V0.5, 1998.

[Ishikawa 99] Yutaka Ishikawa, Atsushi Hori, Hiroshi Tezuka, Shinji Sumimoto, Toshiyuki Takahashi & Hiroshi Harada. *Parallel C++ Programming System on Cluster of Heterogeneous Computers*. In 8^{th} Heterogeneous Computing Workshop (HCW'99), pp. 73–82, 1999.

[Jones 01] James Jones (editor). *PBS Pro Release 5.1 - User Guide*. Veridian Systems, Inc., 2001.

[Karl 98] Holger Karl. *Bridging the Gap between Distributed Shared Memory and Message Passing*. Concurrency: Practice and Experience, vol. 10, no. 11-13, pp. 887–900, 1998.

[Kavi 99] Krishna Kavi. *Multithreaded System Implementations*. IASTED Journal of Microcomputer Applications, vol. 18, no. 2, 1999.

[Keleher 92] Pete Keleher, Alan Cox & Willy Zwaenepoel. *Lazy Release Consistency for Software Distributed Shared Memory*. In 19^{th} International Symposium on Computer Architecture (ISCA'92), pp. 13–21, 1992.

[Keller 01] Axel Keller & Alexander Reinefeld. *Anatomy of a Resource Management System for HPC Clusters*. Annual Review of Scalable Computing, vol. 3, pp. 1–31, 2001.

[Kim 01] Jin-Soo Kim, Kangho Kim & Sung-In Jung. *SOVIA: A User-level Sockets Layer Over Virtual Interface Architecture*. In 3^{rd} IEEE International Conference on Cluster Computing (CLUSTER'01), 2001.

[Klaiber 94] Alexander Klaiber & Henry Levy. *A Comparison of Message Passing and Shared Memory Architectures for Data Parallel Programs*. In 21^{st} International Symposium on Computer Architecture (ISACA'94), pp. 94–105, 1994.

[Kranz 93] David Kranz, Kirk Johnson, Anant Agarwal, John Kubiatowicz & Beng-Hong Lim. *Integrating Message-Passing and Shared-Memory: Early Ex-*

perience. In 4^{th} ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'93), pp. 54–63, 1993.

[Kuhns 99] Fred Kuhns, Carlos O'Ryan, Douglas Schmidt, Ossama Othman & Jeff Parsons. *The Design and Performance of a Pluggable Protocols Framework for Object Request Broker Middleware*. In 6^{th} International Workshop on Protocols For High-Speed Networks (PfHSN'99), pp. 81–98, 1999.

[Kurmann 03] Christian Kurmann & Thomas Stricker. *Zero-Copy for CORBA - Efficient Communication for Distributed Object Middleware*. In 12^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC'03), pp. 4–13, 2003.

[Kutluğ 01] Sencer Kutluğ, Mohammad Banikazemi, Dhabaleswar Panda & Ponnuswamy Sadayappan. *VIBe: A Micro-benchmark Suite for Evaluating Virtual Interface Architecture (VIA) Implementations*. In 15^{th} International Parallel and Distributed Processing Symposium (IPDPS'01), 2001.

[Langendoen 96] Koen Langendoen, John Romein, Raoul Bhoedjang & Henri Bal. *Integrating Polling, Interrupts, and Thread Management*. In 6^{th} Symposium on the Frontiers of Massively Parallel Computing (Frontiers'96), 1996.

[Langendoen 98] Koen Langendoen, Rutger Hofman & Henri Bal. *Challenging Applications on Fast Networks*. In 4^{th} International Symposium on High-Performance Computer Architecture (HPCA'98), pp. 68–79, 1998.

[Lu 95] Honghui Lu, Sandhya Dwarkadas, Alan Cox & Willy Zwaenepoel. *Message Passing Versus Distributed Shared Memory on Networks of Workstations*. In IEEE/ACM Supercomputing Conference (SC'95), 1995.

[Maassen 02] Jason Maassen, Thilo Kielmann & Henri Bal. *GMI: Flexible and Efficient Group Method Invocation for Parallel Programming*. In 6^{th} Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR'02), 2002.

[Meijers 02] Frans Meijers. *Evaluation of Myrinet for the Data Acquisition of the CMS Experiment at the Large Hadron Collider*. Myrinet User's Group Conference (MUG'02), 2002.

[Meijers 03] Frans Meijers. *Studies for the CMS Event Builder*. Conference for Computing in High Energy and Nuclear Physics (CHEP'03), 2003.

[Mellanox 03] Mellanox Technologies. *Leadership in InfiniBand Solutions*. InfiniBand HPC Clustering Solutions, 2003.

[Microsoft 00] Microsoft Corporation. *Network Load Balancing Technical Overview*. White Paper, 2000.

[Moreira 01] Cecília Moreira. CoRes - Computação Orientada ao Recurso - uma Especificação. Master's thesis, Universidade do Minho, 2001.

[MSC 01] MSC.Software Corporation. *Overview of MSC.Linux 2001*, 2001.

[Myricom 00] Myricom, Inc. *The GM Message Passing System*, 2000.

[Myricom 02] Myricom, Inc. *VI-GM Reference Manual*. 1.0, 2002.

[Namyst 95] Raymond Namyst & Jean-François Méhaut. *PM^2: Parallel Multithreaded Machine - A computing environment for distributed architectures*. In Parallel Computing Conference (ParCo'95), pp. 279–285, 1995.

[Nieplocha 96a] Jarek Nieplocha & Ian Foster. *Disk Resident Arrays: An Array-Oriented I/O Library for Out-of-Core Computations*. In 6^{th} Symposium on the Frontiers of Massively Parallel Computation (Frontiers'96), 1996.

[Nieplocha 96b] Jarek Nieplocha, Robert Harrison & Ian Foster. *Explicit Management of Memory Hierarchy*. Advances in High Performance Computing, pp. 185–198, 1996.

[Nieplocha 99] Jarek Nieplocha & Jialin Ju. *ARMCI: A Portable Aggregate Remote Memory Copy Interface*. In 3^{rd} Workshop on Run-Time Systems for Parallel Programming (RTSPP'99), pp. 533–546, 1999.

[Nieplocha 01] Jarek Nieplocha, Jialin Ju & Edoardo Apra. *One-sided Communication on the Myrinet-based SMP Clusters using the GM Message-Passing Library*. In 1^{st} Workshop on Communication Architecture for Clusters (CAC'01), 2001.

[Nieplocha 02] Jarek Nieplocha, Robert Harrison, Mukul Kumar, Bruce Palmer, Vinod Tipparaju & Harold Trease. *Combining Distributed and Shared Memory Models: Approach and Evolution of the Global Arrays Toolkit*. In Workshop on Performance Optimization for High-Level Languages and Libraries (POHLL'02), 2002.

[Nieuwpoort 01] Rob van Nieuwpoort, Thilo Kielmann & Henri Bal. *Efficient Load Balancing for Wide-Area Divide-and-Conquer Applications*. In 8^{th} ACM

SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'01), pp. 34–43, 2001.

[Nieuwpoort 02] Rob van Nieuwpoort, Jason Maassen, Rutger Hofman, Thilo Kielmann & Henri Bal. *Ibis: an Efficient Java-based Grid Programming Environment.* In Joint ACM Java Grande - ISCOPE 2002, pp. 18–27, 2002.

[Nikolopou. 99a] Dimitrios Nikolopoulos, Eleftherios Polychronopoulos & Theodore Papatheodorou. *Fine-Grain and Multiprogramming-Conscious Nanothreading with the Solaris Operating System.* In 5^{th} International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), pp. 1797–1803, 1999.

[Nikolopou. 99b] Dimitrios Nikolopoulos, Eleftherios Polychronopoulos, Theodore Papatheodorou, Christos Antonopoulos, Ioannis Venetis & Panagiotis Hadjidoukas. *Achieving Multiprogramming Scalability on Intel SMP Platforms: Nanothreading in the Linux Kernel.* In Parallel Computing Conference (ParCo'99), pp. 623–630, 1999.

[O'Carroll 98] Francis O'Carroll, Hiroshi Tezuka, Atsushi Hori & Yutaka Ishikawa. *MPICH-PM: Design and Implementation of Zero Copy MPI for PM.* Technical report TR-97011, Real-World Computing Partnership, 1998.

[OCG 01] The Open Cluster Group. *OSCAR: A packaged Cluster software stack for High Performance Computing*, 2001.

[Oliveira 97] Vitor Oliveira. *Memória Partilhada, Passagem de Mensagens e Míltiplos Fios-de-Execução.* Technical report, Universidade do Minho, 1997.

[Olson 03] Ryan Olson, Michael Schmidt, Mark Gordon & Alistair Rendell. *Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model.* In IEEE/ACM Supercomputing Conference (SC'03), 2003.

[Pant 99] Avneesh Pant. *A High Performance MPI Implementation on the NTSC VIA Cluster.* Technical report, NCSA, University of Illinois at Urbana-Champaign, 1999.

[Papadopou. 01] Philip Papadopoulos, Mason Katz & Greg Bruno. *NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters.* In 3^{rd} IEEE International Conference on Cluster Computing (CLUSTER'01), pp. 258–270, 2001.

[ParGrpKU 01] Parallel Research Group. *SCE: Scalable Cluster Environment 1.0.* Kasetsart University, Thailand, 2001.

[Peisert 01] Sean Peisert & Scott Baden. *A Programming Model for Automated Decomposition on Heterogeneous Clusters of Multiprocessors.* Technical report, UCSD CSE, 2001.

[Pendarakis 01] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma & Marcel Waldvogel. *ALMI: An application Level Multicast Infrastructure.* In 3^{rd} Usenix Symposium on Internet Technologies & Systems (USITS'01), pp. 49–60, 2001.

[Petrini 02] Fabrizio Petrini, Wu chun Feng, Adolfy Hoisie, Salvador Coll & Eitan Frachtenberg. *The Quadrics Network: High-Performance Clustering Technology.* IEEE Micro, vol. 22, no. 1, pp. 46–57, 2002.

[Philippsen 00] Michael Philippsen, Bernhard Haumacher & Christian Nester. *More efficient serialization and RMI for Java.* Concurrency: Practice and Experience, vol. 12, pp. 495–518, 2000.

[Pina 97] António Pina. *MC^2 - Modelo de Computação Celular: Origem e Evolução.* PhD thesis, Universidade do Minho, 1997.

[Pina 00] António Pina, Albano Alves, Vítor Oliveira & Cecília Moreira. *CoR's Faster Route over Myrinet.* In 1^{st} Myrinet Users Group Conference (MUG'00), pp. 173–179. INRIA, 2000.

[Pina 02] António Pina, Vitor Oliveira, Cecília Moreira & Albano Alves. *pCoR - a Protoype for Resource Oriented Computing.* In 7*th* International Conference on Applications of High-Performance Computers in Engineering (HPC'02), pp. 251–262, 2002.

[Planquelle 99] Benoît Planquelle, Jean-François Méhaut & Nathalie Revol. *Multi-protocol communications and high speed networks.* In Euro-Par 1999, pp. 139–143, 1999.

[Price 03] Gregory Price & David Lowenthal. *A Comparative Analysis of Fine-Grain Threads Packages.* Journal of Parallel and Distributed Computing, vol. 63, no. 11, pp. 1050–1063, 2003.

[Protopopov 01] Boris Protopopov & Anthony Skjellum. *A multi-threaded Message Passing Interface (MPI) arcgitecture: performance and program issues.* Journal of Parallel and Distributed Computing, vol. 61, no. 4, pp. 449–466, 2001.

[Ramamoor. 97] Ravi Ramamoorthi, Adam Rifkin, Boris Dimitrov & Mani Chandy. *A General Resource Reservation Framework for Scientific Computing.*

In Scientific Computing in Object-Oriented Parallel Environments (ISCOPE'97), pp. 283–290, 1997.

[RegentsUC 02] Regents of the University of California. *Ganglia Toolkit*, 2002.

[Reinhardt 94] Steven Reinhardt, James Larus & David Wood. *Tempest and Typhoon: User-Level Shared Memory.* In 21^{st} International Symposium on Computer Architecture (ISCA'94), pp. 325–336, 1994.

[Roch 03] Jean-Louis Roch, Thierry Gautier & Rémi Revire. *Athapascan : an API for Asynchronous Parallel Programming.* Technical report RT-0276, INRIA, 2003.

[Rühl 96] Tim Rühl, Henri Bal, Raoul Bhoedjang, Koen Langendoen & Gregory Benson. *Experience with a Portability Layer for Implementing Parallel Programming Systems.* In 2^{nd} International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'96), pp. 1477–1488, 1996.

[Scyld 01] Scyld Computing Corporation. *Scyld Beowulf Clustering for High Performance Computing*, 2001.

[Seifert 00] Friedrich Seifert, Daniel Balkanski & Wolfgang Rehm. *Comparing MPI Performance of SCI and VIA.* In 3^{rd} International Conference on SCI-based Technology and Research (SCI-Europe'00), 2000.

[Seitz 01] Charles Seitz. *Recent Advances in Cluster Networks.* In 3^{rd} IEEE International Conference on Cluster Computing (CLUSTER'01), 2001.

[Shivam 01] Piyush Shivam, Pete Wyckoff & Dhabaleswar Panda. *EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing.* In IEEE/ACM Supercomputing Conference (SC'01), 2001.

[Shivam 02] Piyush Shivam, Pete Wyckoff & Dhabaleswar Panda. *Can User Level Protocols Take Advantage of Multi-CPU NICs?* In 16^{th} International Parallel and Distributed Processing Symposium (IPDPS'02), pp. 64–69, 2002.

[Silveira 00] André Silveira, Rafael Ávila, Marcos Barreto & Philippe Navaux. *DPC++: Object-Oriented Programming Applied to Cluster Computing.* In 6^{th} International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'00), volume 5, pp. 2515–2521, 2000.

[Singh 98] Gurdip Singh & Kusuma Vellanki. *A Distributed Protocol for Constructing Multicast Trees*. In 2^{nd} International Conference On Principles Of Distributed Systems (OPODIS'98), pp. 61–76, 1998.

[Sistare 02] Steven Sistare & Christopher Jackson. *Ultra-High Performance Communication with MPI and the Sun Fire Link Interconnect*. In IEEE/ACM Supercomputing Conference (SC'02), 2002.

[Snir 95] Marc Snir, Steve Otto, David Walker, Jack Dongarra & Steven Huss-Lederman. MPI: The Complete Reference. MIT Press, 1995.

[Speight 97] Evan Speight & John Bennett. *Brazos: A Third Generation DSM System*. In 1^{st} USENIX Windows NT Workshop, 1997.

[Speight 98a] Evan Speight, Hazim Abdel-Shafi & John Bennett. *An Integrated Shared-Memory / Message Passing API for Cluster-Based Multicomputing*. In 2^{nd} IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN'98), pp. 146–153, 1998.

[Speight 98b] Evan Speight & John Bennett. *Using Multicast and Multithreading to Reduce Communication in Software DSM Systems*. In 4^{th} International Symposium on High-Performance Computer Architecture (HPCA'98), 1998.

[Speight 99] Evan Speight, Hazim Abdel-Shafi & John Bennett. *Realizing the Performance Potential of the Virtual Interface Architecture*. In 13^{th} International Conference on Supercomputing (ICS'99), pp. 184–192, 1999.

[Strumpen 95] Volker Strumpen & Thomas Casavant. *Implementing Communication Latency Hiding in High-Latency Computer Networks*. In 3^{rd} International Conference on High Performance Computing and Networking Europe (HPCN Europe'95), pp. 86–93, 1995.

[Strumpen 96] Volker Strumpen. *Software-Based Communication Latency Hiding for Commodity Workstation Networks*. In 25^{th} International Conference on Parallel Processing (ICPP'96), pp. 146–153, 1996.

[Sumimoto 99] Shinji Sumimoto, Hiroshi Tezuka, Atsushi Hori, Hiroshi Harada, Toshi yuki Takahashi & Yutaka Ishikawa. *The Design and Evaluation of High Performance Communication using a Gigabit Ethernet*. In International Conference on Supercomputing, pp. 260–267, 1999.

[Sun 02] Yuzhong Sun, David Bader, Xiaola Lin & Yibei Ling. *Broadcast on Clusters of SMPs with Optimal Concurrency.* In 8^{th} International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02), 2002.

[Takahashi 99] Toshiyuki Takahashi, Francis O'Carroll, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Hiroshi Harada, Yutaka Ishikawa & Peter Beckman. *Implementation and Evaluation of MPI on an SMP Cluster.* In Parallel and Distributed Processing - IPPS/SPDP'99 Workshops, pp. 1178–1192, 1999.

[Tam 00] Anthony Tam & Cho-Li Wang. *Contention-free Complete Exchange Algorithm on Clusters.* In 2^{nd} IEEE International Conference on Cluster Computing (CLUSTER'00), pp. 57–64, 2000.

[Tezuka 97] Hiroshi Tezuka, Atsushi Hori, Yutaka Ishikawa & Mitsuhisa Sato. *PM: An Operating System Coordinated High Performance Communication Library.* In 5^{th} International Conference on High Performance Computing and Networking Europe (HPCN Europe'97), pp. 708–717, 1997.

[Thakur 03] Rajeev Thakur & William Gropp. *Improving the Performance of Collective Operations in MPICH.* In 10^{th} European PVM/MPI Users' Group Meeting (EuroPVM/MPI'03), pp. 257–267, 2003.

[TOP500 04] *The Top500 SuperComputer Sites.* http://www.top500.org, 2004.

[Turner 01] Dave Turner, Weiyi Chen & Ricky Kendall. *Performance of the MP_Lite message-passing library on Linux clusters.* In 2^{nd} International Conference on Linux Clusters: The HPC Revolution, 2001.

[Verstoep 96] Kees Verstoep, Koen Langendoen & Henri Bal. *Efficient Reliable Multicast on Myrinet.* In 25^{th} International Conference on Parallel Processing (ICPP'96), volume III, pp. 156–165, 1996.

[von Eicken 92] Thorsten von Eicken, David Culler, Seth Goldstein & Klaus Schauser. *Active Messages: a Mechanism for Integrated Communication and Computation.* In 19^{th} International Symposium on Computer Architecture (ISCA'92), pp. 256–266, 1992.

[Wang 02] Cho-Li Wang, Anthony Tam, Benny Cheung, Wenzhang Zhu & David Lee. *Directed Point: A Communication Subsystem for Commodity Supercomputing with Gigabit Ethernet.* Future Generation Computer Systems, vol. 18, no. 3, pp. 401–420, 2002.

[Warschko 99a] Thomas Warschko, Joachim Blum & Walter Tichy. *A Reliable Transmission Protocol for Myrinet.* In 2^{nd} Workshop on Cluster-Computing, pp. 135–144, 1999.

[Warschko 99b] Thomas Warschko, Joachim Blum & Walter Ticky. *On the Design and Semantics of User-Space Communication Subsystems.* In 5^{th} International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), pp. 2344–2350, 1999.

[Welsh 97] Matt Welsh, Anindya Basu & Thorsten von Eicken. *Incorporating Memory Management into User-Level Network Interfaces.* Technical report TR97-1620, Cornell University, 1997.

[Welsh 00] Matt Welsh, Steven Gribble, Eric Brewer & David Culler. *A Design Framework for Highly Concurrent Systems.* Technical report, University of California at Berkeley, 2000.

[Wong 99] Kwan-Po Wong & Cho-Li Wang. *Push-Pull Messaging: A High-Performance Communication Mechanism for Commodity SMP Clusters.* In 28^{th} International Conference on Parallel Processing (ICPP'99), pp. 12–21, 1999.

[Worley 02] Patrick Worley. *Scaling the Unscalable: A Case Study on the AlphaServer SC.* In IEEE/ACM Supercomputing Conference (SC'02), 2002.

[Xu 99] Cheng-Zhong Xu, Brian Wims & Ramzi Jamal. *Distributed Shared Array: An Integration of Message Passing and Multi-Threading on SMP Clusters.* In 11^{th} IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'99), pp. 305–310, 1999.

[Zhang 00] Wensong Zhang. *Linux Virtual Server for Scalable Network Services.* In Ottawa Linux Symposium, 2000.

[Zhou 98] Honbo Zhou & Al Geist. *LPVM: A Step Towards Multithreaded PVM.* Concurrency: Practice and Experience, vol. 10, no. 5, pp. 407–416, 1998.