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 único. 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

O trabalho aqui apresentado segue uma orientação clássica da investigação no domínio da computação paralela: as metodologias e modelos para programação de aplicações devem permitir a exploração eficiente do hardware disponível, mas também oferecer abstracções convenientes, que facilitem o processo de desenvolvimento. Mais especificamente, a abordagem desenvolvida pretende conjugar os seguintes objectivos:

- exploração eficiente de *clusters* que integram máquinas multiprocessador (nós SMP) e múltiplas tecnologias de interligação de elevado desempenho (redes SAN variadas);
- programação de aplicações com base em paradigmas bem conhecidos, mas integrando facilidades para a selecção/alocação de recursos físicos, em tempo de execução, em ambientes multi-aplicação e multi-utilizador, numa perspectiva cooperativa.

A utilização de 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 várias tecnologias disponíveis. A criação de vários *subclusters*, um por cada tecnologia, e a interligação destes através de nós multi-interface conduz 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 facilmente podem ser identificados três níveis de paralelismo: interprocessador (intranó), internó (*intrasubcluster*) e *intersubcluster*. A exploração eficiente destes sistemas obriga à utilização de metodologias específicas, que possibilitem explorar a localidade.

O desenvolvimento de aplicações paralelas está, normalmente, conotado com a resolução de problemas científicos com requisitos computacionais elevados. Nestes casos, o programador codifica uma única aplicação que tem ao seu dispor a totalidade dos recursos físicos

(hardware) do *cluster*. 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 científico, 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 único, 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 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.

Deste modo, é normal o programador dispor apenas de uma imagem de sistema único, que não permite explorar as várias localidades presentes num *cluster* SMP multi-SAN. Na verdade, é vulgar assumir que um *cluster* encerra um alto nível de homogeneidade e que deve ser visto como uma máquina com um elevado número de processadores.

Por outro lado, as abstracções oferecidas pelas plataformas tradicionais são altamente 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.

Portanto, é necessário definir uma nova arquitectura para a exploração de *clusters*, que permita atingir, simultaneamente, os dois objectivos apontados anteriormente.

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 único. 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 único minimalista (de baixo-nível). Esta garante que a possível inabilidade do programador, no que respeita ao manuseamento das particularidades do sistema de computação, não compromete o funcionamento das aplicações, embora sem garantias de elevados níveis de desempenho.

Em concreto foram desenvolvidas duas bibliotecas, que traduzem novos modelos para a comunicação e computação em *clusters*. A primeira, uma biblioteca de comunicação com suporte para múltiplas tecnologias SAN, introduz o conceito de comunicação orientada ao recurso, o qual se baseia na exploração de um sistema de directório de baixo-nível próprio. A segunda, uma biblioteca para programação de aplicações paralelas 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.

1.3 Organização da dissertação

O capítulo 2 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. É também apresentada a forma como os paradigmas da programação por memória partilhada, passagem de mensagens e memória global são suportados.

O capítulo 3 apresenta a biblioteca R_OCl e o modelo de comunicação orientada ao recurso. São explicados os vários componentes do sistema e são apresentados detalhes de implementação relativos à exploração das tecnologias Myrinet e Gigabit. Faz-se ainda uma breve exposição do interface disponível para o programador.

O capítulo 4 pode ser visto como um complemento ao capítulo 2. É apresentada a biblioteca $m_\varepsilon\mu$, que materializa as metodologias e modelos apresentados no capítulo 2, e explica-se de que forma as funcionalidades do R_OCl servem de base a esta biblioteca.

O capítulo 5 avalia o desempenho do $m_\varepsilon\mu$, sendo apresentados alguns modelos de avaliação próprios.

O capítulo 6 mostra a conveniência na utilização do $m_\varepsilon\mu$, no que respeita à programação de aplicações de acordo com metodologias e técnicas convencionais.

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

1.3.1 Cronologia das etapas do trabalho

Com o intuito de validar o caminho seguido neste trabalho, houve a preocupação de publicar os resultados alcançados em cada etapa. Os vários artigos produzidos, publicados em conferências internacionais, constituem uma alternativa a este texto, para aqueles que, através de uma leitura rápida, pretendem tomar conhecimento das principais ideias preconizadas. Neste sentido, enumeram-se de seguida esses artigos, sendo indicada, para cada um deles, uma breve descrição dos assuntos tratados:

1. CoR's Faster Route over Myrinet. First Myrinet Users Group Conference (MUG 2000), INRIA, pp. 173-179, Lyon, 2000.
   ↪ Primeira abordagem à exploração de uma tecnologia de comunicação de elevado desempenho numa plataforma de programação com um elevado nível de abstração.

2. Scalable Multithreading in a Low Latency Myrinet Cluster. $5th$ International Meeting on High Performance Computing for Computational Science (VECPAR 2002) - Selected Papers and Invited Talks, LNCS 2565, Springer, pp. 579-592, Porto, 2002.
   ↪ 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. $7th$ International Conference on Applications of High-Performance Computers in Engineering (HPC'02), WITpress, pp. 241-250, Bologna, 2002.
   ↪ Concepção de sistemas de despacho de mensagens com recurso a 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. 3^{rd} International Conference on Computational Science (ICCS 2003), LNCS 2658, Springer, pp. 1022-1031, Saint Petersburg, 2003.
   ↪ 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. Euro-Par 2003, LNCS 2790, Springer, pp. 969-979, Klagenfurt, 2003.
   ↪ Introdução do conceito de comunicação orientada ao recurso e apresentação da biblioteca R_OCl.

6. Evaluating Applications Performance in a Multi-networked Cluster. 15^{th} IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003), ACTA Press, pp. 375-380, Marina del Rey, 2003.
   ↪ Avaliação exaustiva do desempenho da biblioteca R_OCl.

7. Mapping application-level components into hierarchical systems resources. The 2004 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '2004), CSREA Press, Las Vegas, 2004.
   ↪ 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. 5th Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS '2004), Kluwer Academic Publishers, Budapest, 2004.
   ↪ Apresentação de uma metodologia para programação de aplicações em *clusters* SMP multi-SAN.

Capítulo 2

Modelação de aplicações orientada ao recurso

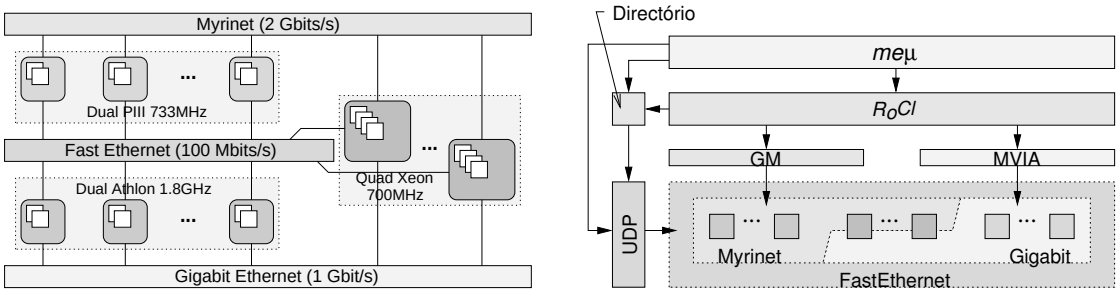
Num passado recente, os *clusters* assumiram um papel importante na execução de programas paralelos para resolução de problemas de índole científica. Essencialmente, os programadores procuravam ambientes de desenvolvimento e execução que permitissem a exploração do *cluster* como se se tratasse de uma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

2.1 Arquitectura do sistema de exploração

A utilização de múltiplas tecnologias SAN num *cluster* traduz-se na existência de nós multi-interface e de 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 2.1(a) apresenta um exemplo de um *cluster* SMP multi-SAN, combinando as tecnologias Myrinet e Gigabit, o qual constitui a plataforma de experimentação subjacente ao trabalho apresentado ao longo deste texto.



(a) (b)

Figura 2.1: Exploração de um *cluster* SMP multi-SAN.

2.1.1 Imagem de sistema único de nível comunicacional

Com o intuito de explorar eficientemente um *cluster* deste tipo, 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 MVIA, por 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, por norma, interligados por uma tecnologia de comunicação secundária, como Fast Ethernet, vulgarmente usada para operações de instalação, manutenção e gestão. Assim, o recurso à difusão de mensagens de localização permite a construção de um serviço de directório totalmente distribuído (ver secção 3.2).

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 único, suportada pelo directório e pela integração de diferentes tecnologias de comunicação (ver figura 2.1(b)).

É importante referir que, apesar de esta imagem de sistema único garantir a total conecti-

vidade 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. Na verdade, o R_OCl nem sequer oferece primitivas para a criação de entidades.

2.1.2 Abstracções para programação de aplicações

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 que tira partido (ver figura 2.1(b)) da funcionalidade do R_OCl – o $m_\varepsilon\mu$ (*Modelling applications and Exploiting clusters through a Unified abstraction layer*) – e 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 único 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 oferecido pelo R_OCl para armazenar os recursos físicos do *cluster* bem como a forma como estes 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 presentes num *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, incluindo as seguintes fases:

- representação da hierarquia de recursos físicos que constituem o *cluster*;
- modelação da aplicação ou do conjunto de aplicações cooperantes que constituem o sistema;
- correspondência entre componentes aplicacionais e recursos físicos.

2.2 Representação de recursos

A manipulação de recursos físicos, nomeadamente a selecção e a reserva de recursos, exige formalismos e mecanismos adequados para a sua representação e organização. Dada a natureza intrinsecamente hierárquica dos *clusters* SMP multi-SAN, uma árvore é naturalmente a estrutura mais indicada para dispor os distintos recursos físicos. A figura 2.2 mostra uma possível hierarquia de recursos, visando a representação do *cluster* apresentado na figura 2.1(a).

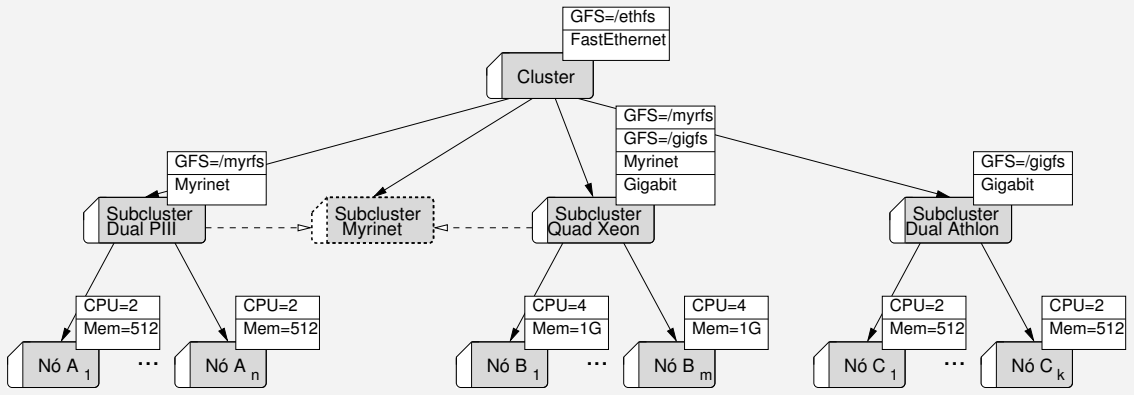


Figura 2.2: Hierarquia de recursos de um *cluster*.

2.2.1 Organização básica

Cada nó de uma árvore de recursos engloba um determinado conjunto de recursos hardware, caracterizados por uma lista de propriedades, a que se dá o nome de domínio. Os domínios de nível mais elevado (topo da árvore) introduzem recursos gerais, tal como uma tecnologia de interligação comum a vários nós do *cluster*, enquanto os domínios correspondentes às folhas da árvore englobam os recursos hardware mais específicos, mas ainda susceptíveis de serem manipulados pelo sistema de exploração.

As propriedades são usadas para evidenciar a presença de qualidades ou para estipular valores que clarificam ou quantificam recursos. Na figura 2.2, por exemplo, as propriedades `Myrinet` e `Gigabit` dividem os recursos do *cluster* em duas classes, enquanto as propriedades `GFS=...` e `CPU=...` estabelecem diferentes formas de acesso a um sistema de ficheiros global e quantificam, em domínios distintos, o recurso processador, respectivamente.

Dependendo da capacidade do sistema de exploração para manipular (alocar ou reservar, por exemplo) recursos a um nível mais fino, algumas propriedades podem ser transformadas em subdomínios. As propriedades `CPU=...`, por exemplo, usadas na caracterização de

domínios que representam máquinas, na figura 2.2, 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.

Cada nó de uma árvore de recursos herda as propriedades do seu ascendente, as quais são reunidas com aquelas 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 2.2 reunirá, portanto, as propriedades `GFS=/ethfs`, `FastEthernet`, `GFS=myrfs`, `Myrinet`, `CPU=2` e `Mem=512`.

A especificação dos recursos físicos exigidos por uma aplicação, através de uma lista de propriedades, permite que o programador solicite ao sistema de exploração que este percorra a árvore de recursos e localize um domínio cujas propriedades acumuladas satisfaçam os requisitos da aplicação. Em relação à figura 2.2, o domínio *Nó* A_1 dará resposta aos requisitos (`Myrinet`) $\wedge$ (`CPU=2`), visto que herda a propriedade `Myrinet` do seu ascendente.

Se os recursos necessários numa aplicação se encontram dispersos pelos domínios 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 ao mecanismo de sintetização. Assim, o domínio *Subcluster Quad Xeon* preenche os requisitos (`Myrinet`) $\wedge$ (`Gigabit`) $\wedge$ (`CPU=4*m`)[1].

2.2.2 Vistas

Os mecanismos de herança e de sintetização não são suficientes quando os recursos pretendidos não podem ser reunidos num domínio em particular. Ainda em relação à figura 2.2, nenhum domínio preenche cabalmente os requisitos (`Myrinet`) $\wedge$ (`CPU=2*n+4*m`)[2]. No entanto, os recursos em causa estão presentes no *cluster*, sendo apenas necessária uma nova vista.

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 do *cluster* de acordo com os interesses de cada aplicação. 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 o mecanismo de partilha de propriedades, como resposta à necessidade de criação dinâmica de vistas.

A criação de um pseudónimo implica designar um ascendente e um ou mais nós originais.

[1] m diz respeito ao número de nós do *subcluster Quad Xeon*.
[2] n e m dizem respeito ao número de nós dos *subclusters Dual PIII* e *Quad Xeon*, respectivamente.

Na figura 2.2, o domínio *Subcluster Myrinet* (representado por uma forma de contornos a tracejado) é um pseudónimo cujos originais (ligados a si por setas a tracejado) são os domínios *Subcluster Dual PIII* e *Subcluster Quad Xeon*. Este pseudónimo herdará as propriedades do domínio *Cluster* e partilhará as propriedades dos seus originais, isto é, reunirá as propriedades directamente associadas aos seus originais, bem como as propriedades previamente herdadas ou sintetizadas por eles.

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 plataforma RSD [3].

2.2.3 Construção da hierarquia de recursos

A disposição dos recursos físicos de um *cluster* segundo uma estrutura hierárquica carece de intervenção administrativa. Na verdade, o administrador do *cluster* será responsável por especificar os recursos, criando um ficheiro de descrição de recursos, de acordo com uma sintaxe predefinida (ver secção 4.1.2).

Para cada recurso, o nome, ascendente e lista de propriedades devem ser definidos. Opcionalmente, uma lista de originais (no caso dos pseudónimos) e uma etiqueta podem também ser indicados. As etiquetas são usadas para referenciar o ascendente ou os originais na especificação de um recurso.

Na fase de arranque do sistema de exploração, o ficheiro de recursos é processado por uma aplicação cliente do directório R_OCl, a qual procede ao registo dos recursos como domínios. O serviço de directório é responsável pela atribuição de identificadores globais aos vários domínios em causa. Os ascendentes e os originais, referenciados no ficheiro de recursos através de etiquetas da responsabilidade do administrador, passam a ser referenciados pelos identificadores devolvidos nas operações de registo.

Um domínio é armazenado como um triplo $\langle identificador, nome, lista\ de\ atributos \rangle$, conforme a funcionalidade oferecida pelo R_OCl (ver secção 3.1.1). Cada atributo corresponde a um par $\langle nome, valor \rangle$, possibilitando o armazenamento de informação de controlo, como o ascendente e cada um dos originais (caso existam) do domínio, para além das propriedades directamente associadas ao domínio.

Na medida em que o R_OCl fornece um serviço de directório global e distribuído, os recursos podem ser registados a partir de qualquer nó do *cluster*, ou seja, o processamento do ficheiro de recursos pode ser efectuado a partir de um único ponto escolhido arbitrariamente. No

entanto, por forma a dispersar as entradas no serviço de directório (correspondentes aos recursos registados) pelos vários servidores que constituem o sistema (ver secção 3.2), o administrador pode associar um nó do *cluster* à especificação de cada um dos recursos. O cliente do serviço de directório que processa o ficheiro de recursos será responsável por sincronizar o arranque dos nós do *cluster* e as operações de registo.

2.3 Modelação de aplicações

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. No seguimento do modelo de computação apresentado em [21], definiu-se um conjunto básico de entidades que visa, simultaneamente, facilitar o desenho de aplicações e manipular de forma eficiente os recursos físicos disponíveis.

2.3.1 Entidades para a modelação de aplicações

O modelo de programação subjacente ao $m_{\varepsilon}\mu$ combina os paradigmas da memória partilhada, memória global e passagem de mensagens, providenciando as seguintes seis entidades para a modelação de aplicações:

- 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 podem enviar/retirar 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.

Com o intuito de garantir uma fácil correspondência entre componentes aplicacionais e recursos físicos (ver secção 2.4), a modelação de uma aplicação compreende a definição de uma hierarquia de entidades, à imagem da abordagem adoptada para a representação e organização de recursos físicos. Os pseudónimos podem também ser usados pelo programador ou pelo sistema de suporte à execução para produzir vistas distintas das entidades

aplicacionais. No entanto, ao contrário da representação de recursos físicos, as hierarquias que representam componentes aplicacionais incluem múltiplas entidades de tipos distintos, as quais não são susceptíveis de serem organizadas arbitrariamente, visto que, por exemplo, as tarefas não podem ter descendentes.

Tal como na representação de recursos físicos, os programadores podem solicitar que o sistema de exploração localize uma entidade específica na hierarquia de um componente aplicacional. Na verdade, as entidades aplicacionais podem ser vistas como recursos lógicos ao dispor do programador (ver secção 2.6). Note-se que, as entidades aplicacionais, tal como os domínios que representam recursos físicos, são armazenadas no directório R_OCl.

A modelação hierárquica de aplicações está em conformidade com o modelo *dividir para conquistar*, o qual tem sido utilizado num ambiente de programação que visa explorar múltiplos *clusters* [25].

2.3.2 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.

Considerando 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. Na verdade, deverão ser usadas as aqui denominadas cadeias de pseudo-ascendência, que traduzem as várias cadeias de nós desde um dado ponto até raíz da árvore, tendo em consideração, para além da relação descendente-ascendente, as múltiplas relações pseudónimo-original, de cada entidade.

Na criação de pseudónimos existem ainda duas restrições importantes: *(i)* apenas os domínios admitem múltiplos originais e *(ii)* os originais têm que ser entidades do mesmo tipo dos respectivos pseudónimos, à excepção dos domínios. Estes poderão ter como originais, para além de outros domínios, caixas postal, operões e tarefas. Note-se que, a criação de entidades pseudónimo não obriga à verificação das segunda e terceira restrições acima indicadas.

A criação de entidades é sempre efectuada a partir de tarefas. Assim, é sempre possível obter o criador de uma entidade – identificador da tarefa responsável pela sua criação – o qual é armazenado como uma propriedade especial da entidade. O ascendente de cada entidade e os originais dos pseudónimos são também armazenados nos mesmos moldes. No que respeita aos descendentes e pseudónimos das entidades, não há qualquer armazenamento explícito. Por conseguinte, a obtenção dos descendentes ou pseudónimos de uma entidade trata-se de uma operação distribuída.

2.3.3 Suporte ao paradigma da memória partilhada

As arquitecturas de computadores actuais exploram o paralelismo ao nível do hardware, como acontece com o encadeamento e a execução simultânea de instruções, suportadas desde há muito pelos processadores, e as técnicas recentes de *hiperthreading*. Além disso, num sistema de computação, os vários componentes hardware constituintes têm um funcionamento inerentemente paralelo. No entanto, do ponto de vista de uma aplicação desenvolvida para tirar partido da funcionalidade oferecida pelos sistemas operativos vulgarmente usados num *cluster*, o primeiro nível de paralelismo que pode ser identificado num *cluster* SMP multi-SAN emerge dos múltiplos processadores existentes em cada nó.

O paralelismo intranó é explorado pela criação de múltiplas tarefas no contexto de um operão. Cada tarefa corresponde a um fio-de-execução de nível sistema. Os fios-de-execução de nível sistema, ao contrário dos de nível utilizador, estão conotados com baixo desempenho, motivado pelos elevados tempos de comutação de contexto. No entanto, possibilitam a utilização dos vários processadores de um sistema SMP e suportam a execução assíncrona de primitivas de sistema. Acresce ainda que, com o aparecimento da biblioteca NPTL no RedHat Linux 9.0, os fios-de-execução de nível sistema tornaram-se uma alternativa interessante devido aos significativos melhoramentos ao nível do desempenho [24].

A cooperação entre tarefas de um operão faz-se através de estruturas de dados partilhadas, criadas no contexto do operão, o qual, num sistema Linux, é materializado num processo. A criação e utilização destas estruturas faz-se à margem do modelo de programação do $m_{\mathcal{E}}\mu$, visto não serem colocadas ao dispor do programador entidades específicas para esse efeito. De igual modo, a sincronização entre as tarefas, no que respeita ao acesso aos dados partilhados, é feita por meio das primitivas POSIX destinadas à 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.

2.3.4 Suporte ao paradigma da passagem de mensagens

A abordagem seguida no $m_\varepsilon\mu$ amplia o modelo tradicional da passagem de mensagens, com a integração de entidades de tipos variados, usadas na modelação de aplicações, no processo de comunicação. De facto, todas as mensagens são geradas por tarefas, que podem usar os seus identificadores ou identificadores correspondentes a caixas postal para preencher o campo origem da mensagem, mas o destino da mensagem pode ser uma tarefa, um operão, um domínio ou uma caixa postal.

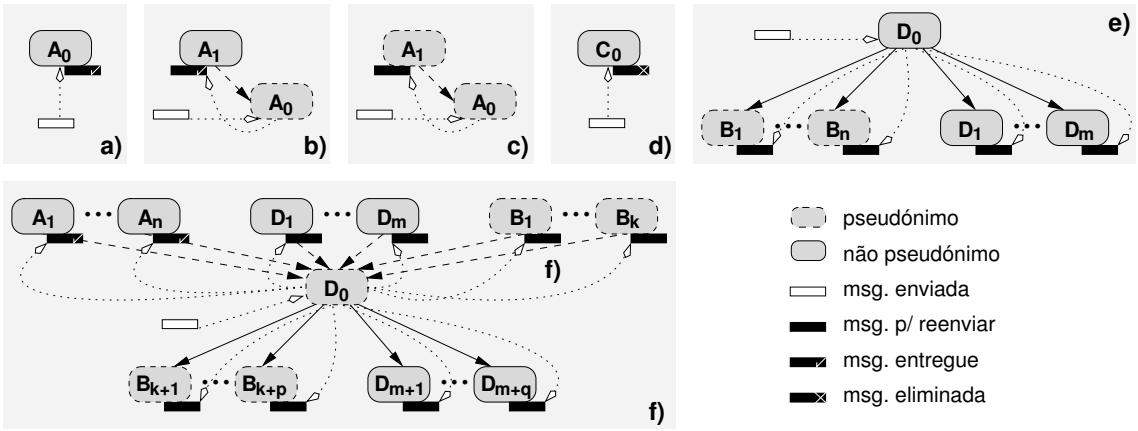


Figura 2.3: Cenários possíveis para a entrega de mensagens.

O envio de uma mensagem a uma tarefa é pacífico, sendo suficiente o mecanismo de comunicação oferecido pelo R_OCl (ver figura 2.3-*a)*). No entanto, 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 2.3-*b)*). Se o original for, por sua vez, um pseudónimo, então será necessário encaminhamento sucessivo (ver figura 2.3-*c)*).

Quando uma mensagem é destinada a um operão, então qualquer tarefa descendente desse operão que não seja um pseudónimo poderá competir pelo acesso à mensagem; o operão servirá de repositório da única cópia da mensagem e uma das tarefas descendentes poderá

consumir essa mensagem. Se o operão for um pseudónimo, o procedimento será similar ao descrito para as tarefas.

A caixa postal, tal como o operão, armazena a cópia única da mensagem, por forma a permitir que uma tarefa reclame a mensagem em causa. No entanto, 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 cada um dos seus originais (ver figuras 2.3-*e),f)*). Qualquer mensagem endereçada a um domínio que represente um recurso físico será descartada (ver figura 2.3-*d)*).

2.3.5 Suporte ao paradigma da 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 de paralelismo no domínio dos dados. A memória partilhada distribuída acarreta, normalmente, algoritmos pesados de sincronização. Deste modo, os mecanismos de memória global, baseados nas operações de leitura/escrita remota das bibliotecas de comunicação baixo-nível e em mecanismos de sincronização mínimos ou mesmo nulos, tem vindo a ganhar especial relevância.

Os blocos de memória criados ao nível dos operões podem ser acedidos remotamente, individualmente, e podem ainda ser unificados através de um agregador de memória. O mecanismo de agregação passa pela criação de um pseudónimo, para cada bloco de memória, como descendente do agregador (ver figura 2.4).

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 criada uma entrada $\langle in\acute{i}cio, tamanho, identificador\rangle$, onde $in\acute{i}cio$ diz respeito ao tamanho actual do espaço de endereçamento global (o qual é iniciado a zero no momento da criação do agregador), $tamanho$ corresponde à dimensão (em bytes) do bloco e $identificador$ é o identificador global do bloco. Desta forma, 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 no espaço de endereçamento global pode ser especificada. Também é 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 2.4, apenas uma fracção do bloco B_1, por exemplo, é

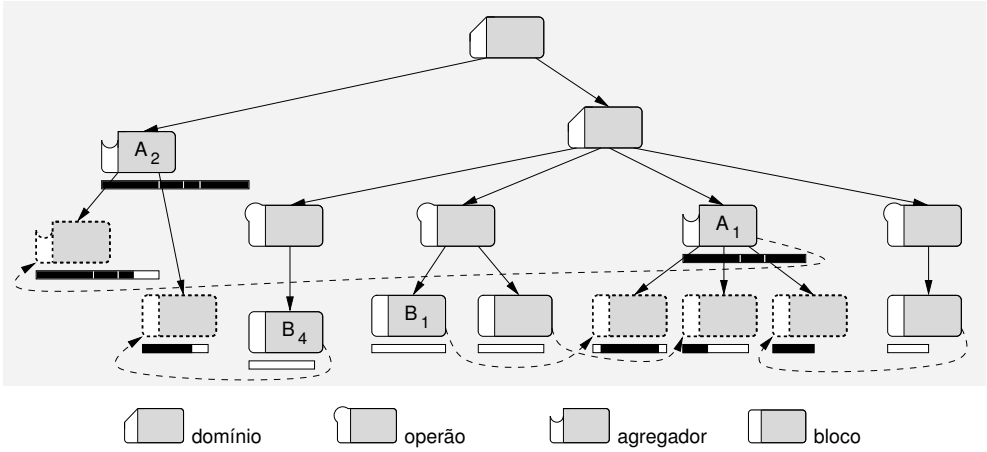


Figura 2.4: Exemplo de agregação de blocos de memória.

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 2.4, se ao agregador A_1 for adicionado um outro bloco, isso não deverá interferir com a posição ocupada pelo bloco B_4 no agregador A_2.

Para obter acesso a uma zona da memória global, antes de mais, um programador deverá 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*, as quais desencadearão as leituras remotas dos vários blocos de memória envolvidos. A actualização complementar é efectuada através de operações *put*.

Note-se que, o apontador referido permite ler e escrever zonas de memória local que é explicitamente sincronizada com os blocos de memória remotos por via de primitivas *get* e *put*. Estas primitivas tiram partido das operações RDMA de baixo-nível, suportadas pelo hardware de comunicação mais actual. Quando o programa deixa de utilizar o fragmento de memória global, o apontador local é libertado.

2.3.6 Um exemplo de modelação

A figura 2.5 apresenta 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 corresponde, na prática, a uma versão simplificada do projecto SIRe[3]. O modelo proposto visa exclusivamente a apresentação das abstracções propostas no $m_{\varepsilon}\mu$. Abordagens específicas, e naturalmente mais elaboradas, para recuperação de informação da *Web* podem ser encontradas em [2, 7].

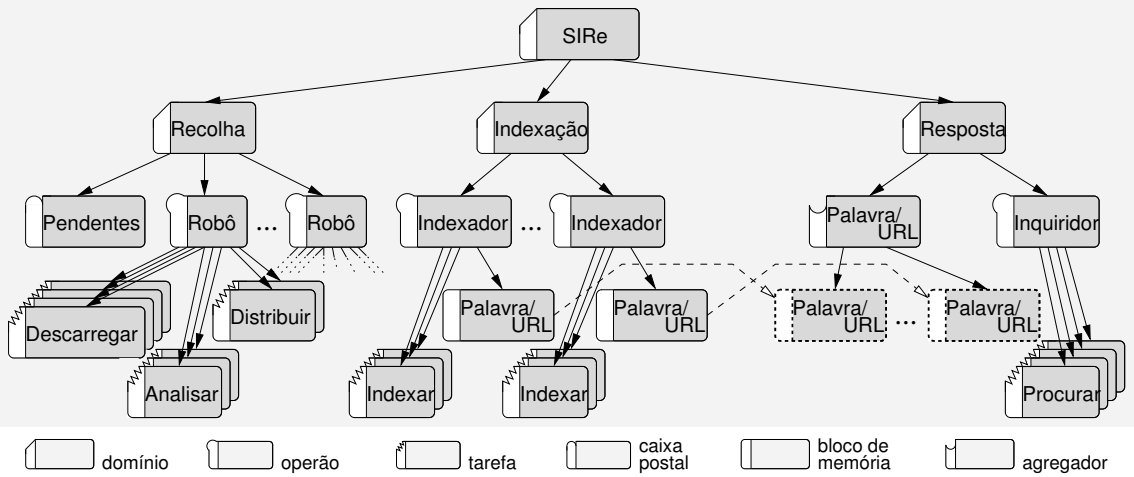


Figura 2.5: Exemplo de modelação do sistema SIRe.

Cada operão *Robô* representa uma réplica de um robô, em execução numa única máquina (um nó do *cluster*). Cada um dos estágios do varrimento da *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 encarregue de analisar o conteúdo das páginas, por exemplo, partem do estado de repouso e, logo que exista uma página descarregada, de acordo com a estratégia de escalonamento do sistema operativo, uma das tarefas inicia o processamento da página.

Os vários estágios de um robô, ou seja, a totalidade das tarefas que compõem os vários estágios do robô, são sincronizados através de estruturas de dados partilhadas ao nível do contexto do operão *Robô* e com base no recurso a 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.

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

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[4], a aplicar a cada um dos URLs. Obviamente, este tipo de estratégia requer protocolos adicionais, para suportar a variação do número de entidades cooperantes, por exemplo. No entanto, como está em causa apenas a apresentação de conceitos e metodologias $m_{\varepsilon}\mu$, o sistema aqui proposto abstrai-se, naturalmente, de muitos detalhes de funcionamento.

Por forma a entregar os URLs a quem de direito, após a análise do conteúdo de cada uma das páginas descarregadas, o estágio *Distribuir* envia a cada operão *Robô* uma mensagem com os URLs que, segundo a função de dispersão, lhe dizem respeito. Assim, às tarefas *Descarregar* de um dado operão *Robô* cabe o papel de processar, 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 respeita 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*.

O sistema de indexação, representado pelo domínio *Indexação*, tem como propósito manter um 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*.

Com o intuito de reunir múltiplos fragmentos de memória, são criados vários 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 apre-

[4] Função de *hash*.

sentados, 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.

Finalmente, o sistema de resposta determina os URLs relevantes para um 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.

O acesso à matriz, 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.

2.4 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 das duas hierarquias produzidas nas fases precedentes, isto é, a fusão da hierarquia representativa do hardware do *cluster* com a hierarquia representativa da aplicação (ou do sistema de aplicações). Fundamentalmente, trata-se de produzir uma única hierarquia que ilustre a correspondência entre recursos lógicos e físicos.

2.4.1 Disposição de recursos lógicos

A figura 2.6 apresenta uma hipótese de integração das entidades da aplicação ilustrada na figura 2.5 na hierarquia de recursos físicos da figura 2.2. 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.

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 respeita 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

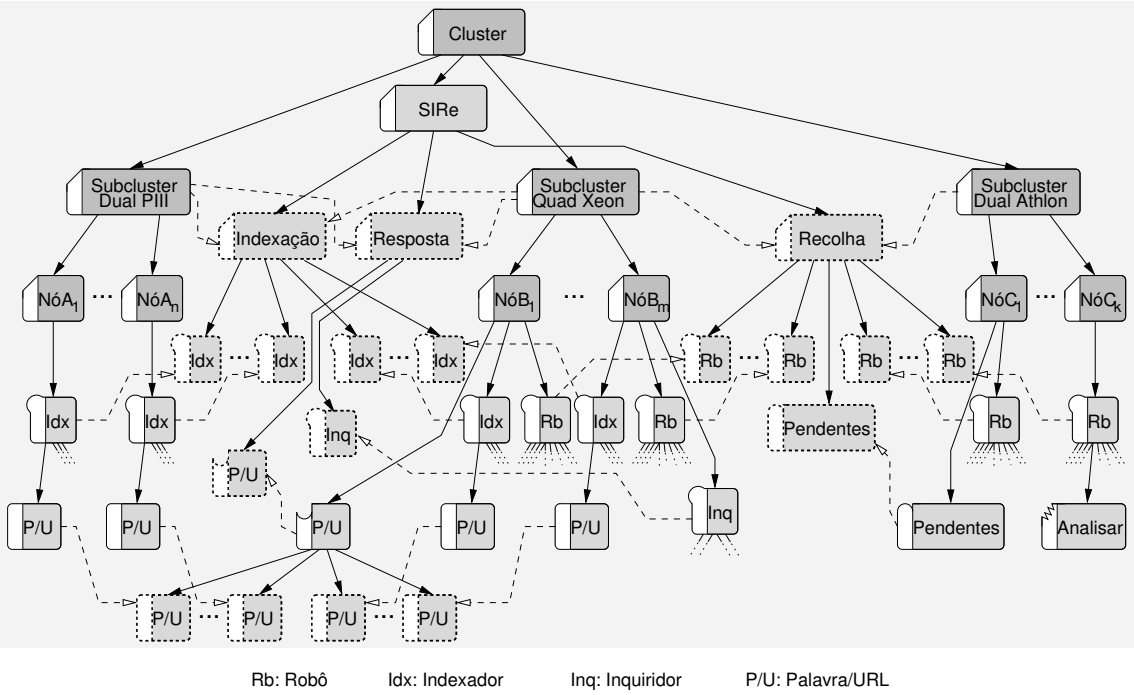


Figura 2.6: Correspondência entre hierarquias lógica e física.

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 2.6, o domínio lógico *Recolha* estabelece os recursos físicos usados pelo sistema de recolha, dado que os operões *Robô* são automaticamente distribuídos pelos nós do *cluster* dispostos sob os originais desse domínio. Na verdade, o domínio *Recolha* constitui um vista particular dos recursos físicos do *cluster*, combinando os recursos dos *subclusters Quad Xeon* e *Dual Athlon*.

Para preservar a hierarquia concebida pelo programador para a aplicação, o sistema de exploração procede à criação transparente 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 vistas distintas: uma vista do programador e a vista do sistema. Esta última é necessária para o normal funcionamento do $m_{\varepsilon}\mu$, que internamente interpreta o sistema de computação e as aplicações que nele executam como um conjunto de máquinas e fios-de-execução, sendo ainda adequada para efeitos de administração. Cada programador/utilizador terá interesse numa vista própria para componentes de software da sua responsabilidade, bem como dos recursos hardware que num dado momento utiliza, enquanto que o administrador terá interesse na vista única

de sistema que, no caso da figura 2.6, corresponderia à árvore constituída pelas entidades não pseudónimo.

A tarefa *Analisar* da figura 2.6, por exemplo, poderá ser alcançada através de dois percursos distintos: *Cluster* → *Dual Athlon* → *Nó* C_k → *Robô* → *Analisar* – vista de sistema – e *Cluster* → *SIRe* → *Recolha* → $Robô^{(Pseudo)}$ → *Analisar* – vista do programador. Note-se que, nenhum pseudónimo é criado para a tarefa *Analisar*, porque as duas vistas estão já integradas pelo domínio pseudónimo *Robô*. Na prática, os pseudónimos permitem saltar entre vistas.

As capacidades 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, na raíz do *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. No entanto, o sistema de exploração será capaz de garantir que a aplicação executa, apesar de o desempenho esperado ser reduzido.

2.4.2 Criação dinâmica de recursos

A criação de recursos lógicos ocorre em dois cenários distintos: no arranque da aplicação, dado que o sistema de exploração cria automaticamente um operão e uma tarefa inicias , e quando as tarefas executam primitivas especificamente com esse propósito. Isto significa que, na sua generalidade, os recursos lógicos são criados durante a execução das aplicações, de acordo com o código escrito pelo programador.

O procedimento para 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 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.

Assim, a criação do domínio pseudónimo *Recolha*, na figura 2.6, exigirá o conhecimento dos identificadores do domínio lógico *SIRe* e os dos domínios físicos *Subcluster Dual Athlon* e *Subcluster Quad Xeon*. O primeiro será, naturalmente, devolvido pela primitiva que criou o domínio *SIRe*, executada numa fase anterior da aplicação. Os outros serão obtidos pelas primitivas de localização de recursos, com base em características fornecidas, como por exemplo, estar presente a tecnologia de comunicação Gigabit.

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 repre-

sente 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 2.6, 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 chega aos domínios originais, que integram a hierarquia de recursos físicos. No entanto, será ainda necessário chegar a um dos domínios *Nó...*, onde efectivamente poderá ser criado o operão.

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 um domínio adequado 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 lógicos, facilmente se conclui que a criação e registo de recursos lógicos se fazem de forma completamente distribuída e assíncrona. O serviço de directório distribuído, suportando o registo local de todos as entidades criadas num dado nó do *cluster*, contribui decisivamente para o bom desempenho deste sistema.

2.5 Sistemas de aplicações

O exemplo de modelação apresentado na secção 2.3.6 constitui uma única aplicação – uma macro-aplicação – subentendendo-se que todos os componentes foram desenhados pelo mesmo programador. No entanto, o sistema alvo – um sistema escalável para recuperação de informação – poderia ser concebido como um conjunto de aplicações mais simples – um sistema de aplicações.

2.5.1 Cooperação interaplicação

No contexto de um sistema de aplicações, vários programadores deverão ser capazes de desenvolver aplicações cooperantes, tendo conhecimento, unicamente, das características e funcionalidades básicas das aplicações desenvolvidas por outros. As distintas aplicações poderão, inclusivamente, no âmbito de um sistema operativo multi-utilizador, executar sob o controlo de vários utilizadores. O sistema de aplicações, como um todo, será suportado por um única hierarquia $m_{\varepsilon}\mu$. Aliás, quaisquer aplicações em execução num *cluster*, num dado momento, serão traduzidas numa única hierarquia $m_{\varepsilon}\mu$, independentemente de

integrarem o mesmo sistema de aplicações ou não, isto é, independentemente do seu nível de cooperação.

Relativamente ao exemplo de modelação referido, robôs distintos, desenvolvidos por vários programadores e utilizando algoritmos de operação variados, poderão cooperar apenas com base em dois pressupostos: cada robô deverá procurar potenciais parceiros e deverá garantir mecanismos para o intercâmbio de URLs. Inclusivamente, os subsistemas de indexação e de resposta, apesar de partilharem algumas entidades lógicas, poderiam ser modelados como aplicações separadas (mas cooperantes), sem que fosse necessário alterar a sua arquitectura.

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, por via da 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. Na figura 2.6, se considerássemos os subsistemas de indexação e resposta como duas aplicações cooperantes distintas, o agregador *Palavra/URL* da aplicação representada pelo domínio *Resposta* teria como descendentes pseudónimos de blocos de memória pertencentes à aplicação representada pelo domínio *Indexação*.

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, conseguindo uma outra forma de cooperação. 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), conseguirá obter uma cópia de todas as mensagens endereçadas a esse domínio, permitindo algum tipo de cooperação. Na secção 6.2 serão apresentadas, com

maior detalhe, algumas abordagens que tiram partido deste mecanismo.

2.5.2 Partilha de recursos físicos

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

Da análise da figura 2.6 depreende-se que os recursos físicos do *cluster*, mais concretamente os nós do *cluster*, são 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.

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.

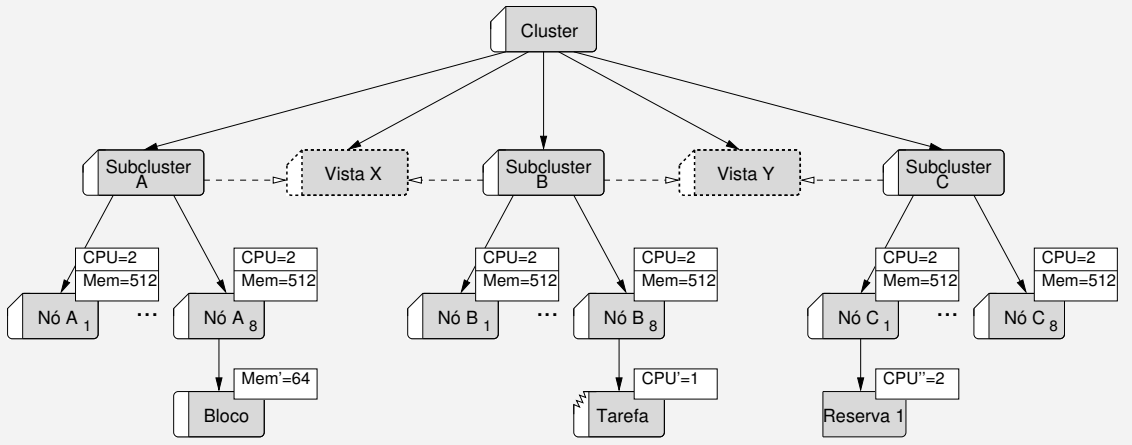


Figura 2.7: Alocação e reserva de recursos físicos.

O mecanismo de alocação de recursos do $m_{\varepsilon}\mu$ é bastante simples. O acto de criar uma entidade lógica sob um determinado domínio físico corresponde, automaticamente, à apropriação de recursos. A criação de um bloco de memória sob um domínio físico correspondente a um nó do *cluster*, por exemplo, significará um decréscimo no valor da propriedade que traduz a memória desse nó, em conformidade com uma propriedade implícita do bloco, que simboliza a memória por ele consumida.

Relativamente à figura 2.7, a criação de uma vista com garantia de 8192 unidades de memória deverá ter por base os requisitos (`Mem=8192`) $\wedge$ (`Mem'=0`), significando que a totalidade da memória deverá estar disponível. Deste modo, o domínio *Vista X* não cumprirá tais requisitos, em virtude de o bloco descendente do domínio *Nó* A_8 incluir uma propriedade implícita 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. A solução adoptada no $m_{\varepsilon}\mu$ passa pela introdução de uma entidade específica – o subtractor – que, ao ser criado 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 2.7, a criação de uma vista com garantia de 31 processadores, excluindo também os não ocupados mas reservados, deverá ter por base os requisitos (`CPU=31`) $\wedge$ (`CPU'=0`) $\wedge$ (`CPU''=0`), 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), dado que dois processadores do domínio *Nó* A_1 estão reservados (com base no subtractor *Reserva*).

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.

A alocação e a reserva de recursos computacionais têm sido amplamente investigadas, fundamentalmente no âmbito da *Grid* [8, 9]. No entanto, a abordagem aqui descrita, que pontualmente poderá comportar algumas das técnicas usadas na *Grid*, tem a vantagem de estar perfeitamente integrada no modelo de programação de aplicações $m_{\varepsilon}\mu$.

2.5.3 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 respeita ao destinatário das operações em causa.

2.6 Orientação ao recurso

O leque de entidades registadas por uma aplicação pode ser visto como o conjunto de recursos lógicos que a aplicação coloca ao dispor de outras. Neste sentido, pode-se dizer que um sistema de aplicações não é mais que um conjunto de recursos lógicos relacionados entre si.

Os domínios físicos que representam o hardware do *cluster* são também registados no directório com um intuito similar: colocar à disposição das aplicações a totalidade dos recursos físicos do *cluster*.

O processo de execução de uma aplicação compreende a associação de recursos lógicos a recursos físicos e a interacção entre recursos lógicos. Em ambos os casos é fundamental o papel do directório para seleccionar/localizar recursos. Neste sentido, o $m_{\varepsilon}\mu$ constitui 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).

O sistema de suporte ao $m_{\varepsilon}\mu$ – o R_OCl – é responsável por garantir os meios necessários ao registo e à localização de recursos físicos e lógicos. Por parte do R_OCl não há qualquer distinção entre os dois tipos de recursos, justificando-se o conceito genérico de recurso.

Do ponto de vista sintáctico, um recurso é caracterizado por um nome, um identificador global (atribuído pelo R_OCl), uma lista de propriedades, um ascendente, uma lista de descendentes, uma lista de pseudónimos, uma lista de originais, uma lista de membros

Tabela 2.1: Caracterização de recursos

domínio	::	R · [(as · im) · or∗] · ds∗ · (pd · im)∗ · oz · pt · ec
operão	::	R · (as · im) · (pd · im)∗ · (or · ec · ds∗) | (ct · rp · ds+) · oz · pt
tarefa	::	R · (as · im) · (pd · im)∗ · (or · ec) | (ex · rp) · pt
cx. postal	::	R · (as · im) · (pd · im)∗ · (or · ec) | rp · pt
bloco	::	R · (as · im) · (pd · im)∗ · (or · rd) | sq · pt
agregador	::	R · (as · im) · (pd · im)∗ · (or · rd) | (ds+ · oz · sz) · pt
subtractor	::	R · (as · im)
R	::	nm · id · pr∗

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, ec=encaminhador, sq=sequência de valores, sz=sincronizador, rd=redireccionador

e uma lista de identificações de membro. O conceito de membro surge da necessidade de manipular descendentes e originais de forma não diferenciada, como acontece no reencaminhamento de mensagens, por parte dos domínios (ver secção 2.3.4). A reunião dos descendentes com os originais de uma entidade é designada por conjunto de membros da entidade, os quais são identificados por um número de ordem. Dado que, uma determinada entidade é membro não só do seu ascendente como também de todos os seus pseudónimos, terá, implicitamente, um conjunto de identificadores de membro, de acordo com os vários contextos onde é referenciada.

Do ponto de vista semântico, um recurso é caracterizado pela presença, ou não, de:

- um organizador – um mecanismo para gestão de membros, quando em número superior a 1;
- um contentor – um contexto para a execução de rotinas e armazenamento de dados;
- um porto – um ponto de acesso ao sistema de comunicação, para envio/recepção de mensagens e leitura/escrita de zonas de memória remotas;
- um encaminhador – um mecanismo para reenvio de mensagens a descendentes e originais;
- um repositório – um conjunto de estruturas de dados que possibilitam o armazenamento de mensagens e a sua recuperação com base em critérios de selecção;
- um executor – um fio-de-execução que executa uma função indicada;
- um redireccionador – um mecanismo para redireccionar leituras/escritas de memória remota;

- uma sequência de valores – uma zona de memória contígua com dados não estruturados;
- um sincronizador – um mecanismo para harmonização de acessos concorrentes.

A tabela 2.1 apresenta a caracterização formal dos vários tipos de recursos suportados no $m_{\varepsilon}\mu$, considerando quer os elementos sintácticos quer os semânticos.

2.7 Epílogo

Por norma, e no contexto dos sistemas de computação mais vulgares, a execução de aplicações, com fortes exigências ao nível do desempenho, é suportada por imagens de sistema único poderosas, como é o caso do Gobelins [14], que, de forma transparente, fazem a gestão dos recursos do *cluster*. Estes sistemas, essencialmente, visam 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 único 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, se assim o entenderem, por forma a potenciar níveis de desempenho mais elevados nas aplicações.

Quando comparado com um *cluster* SMP multi-SAN, um sistema de metacomputação é, necessariamente, um sistema substancialmente mais complexo. No contexto destes sistemas, têm sido propostas arquitecturas genéricas para a gestão de recursos, como se pode constatar no trabalho apresentado em [8]. 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 constitui uma inovação importante.

Capítulo 3

Biblioteca para comunicação inter-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 particular tecnologia de comunicação.

Em alguns casos, como acontece com a biblioteca Madeleine e o sistema DECK, é dada a possibilidade de as aplicações explorarem *clusters* heterogéneos, onde alguns nós dispõem de múltiplos interfaces de comunicação, de tecnologias variadas, e onde dois ou mais *sub-clusters* podem ser identificados, pelo facto de alguns nós disporem de um único interface de comunicação.

No entanto, a necessidade de contemplar a utilização maciça de fios-de-execução, não só por forma a explorar nós SMP mas também para facilitar a modelação de um leque importante de aplicações, bem como a pretensão de ir de encontro ao paradigma da computação orientada ao recurso levaram ao desenvolvimento da biblioteca R_OCl – *Resource oriented Communication library*.

3.1 Comunicação orientada ao recurso

A biblioteca R_OCl, tendo como objectivo suportar o desenvolvimento de ambientes de programação de alto-nível segundo o paradigma da computação orientada ao recurso, introduz um novo modelo de comunicação. Na verdade, a abordagem seguida passa pela inclusão,

nesta biblioteca, de muitas das funcionalidades necessárias à computação orientada ao recurso, facilitando assim o desenvolvimento de abstracções de alto-nível.

3.1.1 Conceitos gerais

O modelo de comunicação do R_OCl define três entidades fundamentais – contextos, recursos e tampões – e recorre às funcionalidades de um serviço de directório de baixo-nível.

Um contexto é criado quando a biblioteca arranca, como resposta à evocação da primitiva de iniciação do R_OCl. Qualquer contexto detém um ou mais portos de comunicação baixo-nível, para envio e recepção de mensagens, comportando-se como um macro-repositório de mensagens.

O recurso constitui uma abstracção usada para a modelação tanto de entidades comunicantes como de entidades computacionais. Os recursos são publicados mediante uma operação de registo num serviço de directório global e distribuído. Qualquer recurso é associado a um contexto preexistente e possui um identificador único, no âmbito do *cluster*. 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 tuplo $\langle nome, valor \rangle$, onde *nome* é uma cadeia de caracteres e *valor* é uma sequência de bytes. Um recurso R com n atributos é definido pela expressão $R = \{\langle nome_1, valor_1 \rangle, ..., \langle nome_n, valor_n \rangle\}$. O programador deverá, portanto, no momento do registo do recurso, enumerar a lista de atributos que caracterizam esse recurso (ver secção 3.2.1).

Com o intuito de minimizar o número de operações de alocação e registo de memória, o R_OCl inclui um sistema de gestão de tampões. As mensagens são mantidas em áreas de memória registada específicas, para permitir comunicação sem cópias de memória. Antes do envio de mensagens, o programador deverá obter tampões de dimensões adequadas. Para a recepção, a biblioteca é responsável por fornecer aos subsistemas de comunicação os necessários blocos de memória pré-registada.

Os identificadores globais atribuídos aos recursos são usados para determinar a origem e o destino das mensagens. A identidade de um recurso poderá ser previamente conhecida pela aplicação ou poderá ser obtida a partir de uma pesquisa no directório.

A figura 3.1 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.

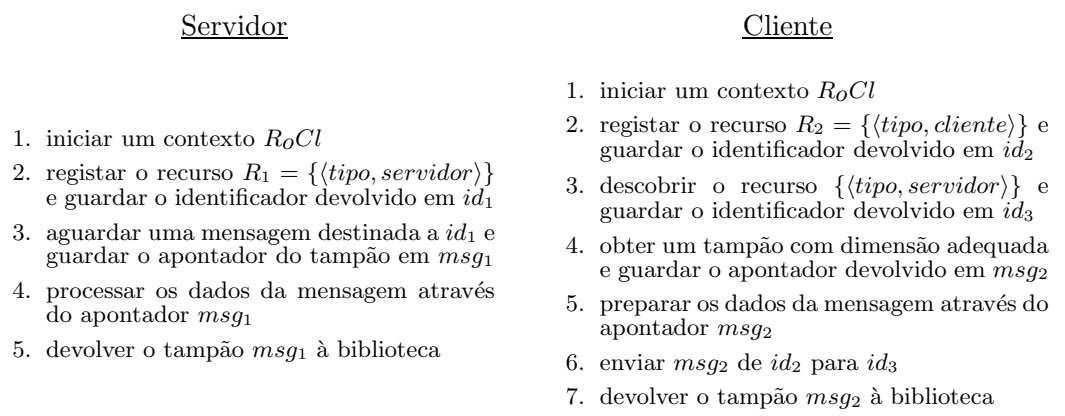


Figura 3.1: Um exemplo básico de comunicação entre recursos.

3.1.2 Interface básico

O conjunto básico de primitivas, que os programadores podem usar para explorar o R_OCl, é apresentado na tabela 3.1, organizado de acordo como as entidades R_OCl envolvidas. O funcioamento geral destas primitivas será discutido ao longo das próximas secções.

Tabela 3.1: Primitivas R_OCl básicas.

Contextos
`int rocl_init(bool bridge)`
`rocl_exit()`
Recursos
`int rocl_register(const rocl_attrl_t *attrs, bool unique)`
`rocl_delete(int gid)`
`int rocl_query(int gid, rocl_attrl_t *attrs, enum rocl_scope scope)`
Tampões
`void * rocl_bfget(int len)`
`rocl_bfret(void *ptr)`
`rocl_bftoret(void *ptr)`
`int rocl_bfstat(const void *ptr, int timeout)`
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)`

De momento 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. A utilização de bibliotecas de comunicação de baixo-nível, tal como GM e 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 [18] e o PVM sobre VIA [11], 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.

3.2 Serviço de directório

O R_OCl cria um sistema totalmente dinâmico, onde as entidades comunicantes podem aparecer e desaparecer, a qualquer momento, durante a execução de uma aplicação. Para suportar esta característica, recorre-se à abstracção "recurso" e á funcionalidade de um serviço de directório distribuído global. A importância de um mecanismo com estas características foi já realçada, por outros autores, em [1] e [12].

O serviço de directório do R_OCl é um sistema distribuído, que oferece acesso eficiente e escalável à informação correspondente a recursos registados. Este serviço facilita o desenvolvimento de plataformas e aplicações de computação distribuída mais flexíveis.

3.2.1 Listas de 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 3.2.

Tabela 3.2: 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_kill_attrl(rocl_attrl_t *attrs)
```

As listas de atributos são usadas tanto no registo como na pesquisa de recursos. Para o

registo correcto 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, os valores dos atributos podem ser omitidos (valores *NULL*), apesar de o tamanho indicado para esses valores ser diferente de zero, por forma a indicar à biblioteca que atributos em particular se pretende obter relativamente a um recurso específico.

Uma lista de atributos é armazenada numa zona de memória contígua, com o objectivo de evitar cópias de memória aquando do seu envio para um servidor (ver secção 3.2.2). Na verdade, a própria lista de atributos é usada como mensagem (pacote) de pedido ou resposta, obrigando, inclusivamente, à reserva de algum espaço à cabeça da lista, para que possa ser incluída informação de controlo.

3.2.2 Operação local

Os recursos R_OCl são registados em cada nó do *cluster* usando um servidor local (ver figura 3.2), o qual gera identificadores globais com base no seu número de série, atribuído na fase de arranque, e num contador de registos mantido localmente. Desta forma, as operações de registo evitam a comunicação entre nós. A base de dados de recursos locais (BD, na figura 3.2) é mantida na memória principal, sendo usadas técnicas de *hashing*, para acelerar as operações de pesquisa.

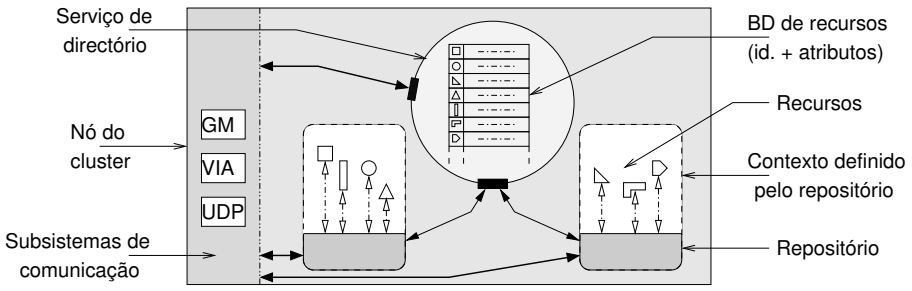


Figura 3.2: Registo de recursos locais.

Os atributos associados a cada recurso, e registados no directório, são inalteráveis, estando apenas disponível, para o programador, uma primitiva destinada a eliminação do registo de um dado recurso. Esta abordagem permite implementar, de forma mais fácil, 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 ordem de pesquisa 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; caso contrário, os atributos completamente especificados serão usados para gerar índices *hash*. Depois de encontrado 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.

3.2.3 Operação global

Quando uma pesquisa em particular não pode ser satisfeita pelo servidor local, é desencadeado um processo de procura global. Numa procura global, todos os servidores em execução no *cluster*, um em cada nó, recebem o pedido, mas apenas responde aquele onde a pesquisa tiver sucesso.

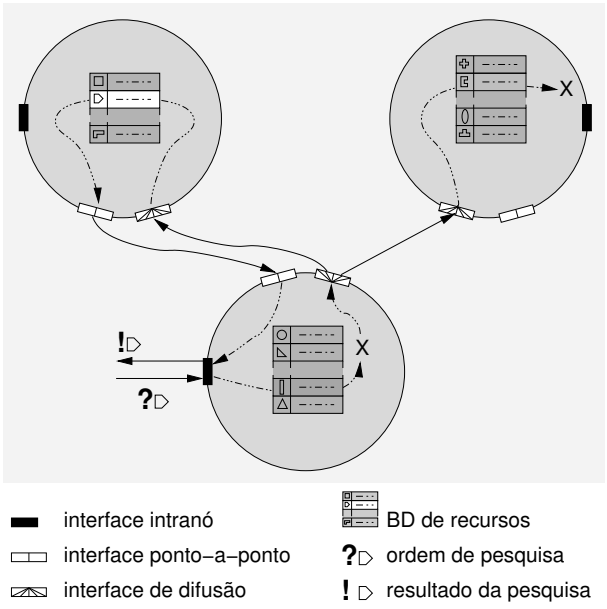


Figura 3.3: Mecanismo de pesquisa global.

O mecanismo de procura global baseia-se na difusão UDP, explorando o facto de a totali-

dade dos nós de um *cluster* estarem, por norma, 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 *subcluster* (uma partição tecnológica), os pedidos poderão ser entregues mediante a combinação de difusão UDP e árvores de dispersão para a tecnologia de comunicação desse *subcluster*. O funcionamento geral 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. O recurso a 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, usado, por exemplo, para explorar hardware Gigabit, devido à orientação à conexão).

3.2.4 Pesquisas com múltiplas respostas

As ordens de pesquisa que não especificam o identificador de um recurso poderão resultar na devolução de múltiplas respostas, oriundas 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 que não são específicos de um único recurso 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 3.3). Este interface não deverá ser usado quando se pretende um única resposta, independentemente de vários recursos poderem cumprir o critério de procura, ou quando se sabe de antemão que será devolvida uma única resposta, para a ordem de pesquisa em causa.

Através deste interface, os resultados de uma pesquisa são obtidos a partir do servidor local, um de cada vez, tal como se se tratasse de uma sequência de pesquisas simples independentes. Cada pacote de pedido/resposta movimenta a informação – uma lista de atributos – correspondente a um único recurso.

Tabela 3.3: 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, os quais mantêm alguma informação de controlo/estado, por cada operação de pesquisa em curso, por forma a poderem decidir: procurar uma resposta na base de dados local, difundir a ordem de pesquisa, 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.

Cada servidor remoto pode devolver um único resultado (informação relativa a um único recurso) como resposta a um pedido recebido por via de uma determinada operação de difusão. 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.

3.3 Troca de mensagens inter-recurso

As aplicações R_OCl endereçam mensagens a recursos, cuja identificação é previamente obtida através do serviço de directório. A passagem de mensagens inter-recurso levanta, essencialmente, dois problemas: o endereçamento de mensagens, dado que não existe uma relação directa entre recursos e endereços de portos dos subsistemas de comunicação, e o despacho de mensagens, dado que, por um lado, os recursos são animados por fios-de-execução, que concorrem pelo acesso a mensagens, e, por outro lado, vários portos de comunicação, associados a distintas tecnologias de comunicação de baixo-nível, têm de ser multiplexados.

3.3.1 Endereçamento de mensagens

Os contextos R_OCl são as únicas 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. Desta forma, tem de ser estabelecida a corres-

pondência entre recursos e contextos.

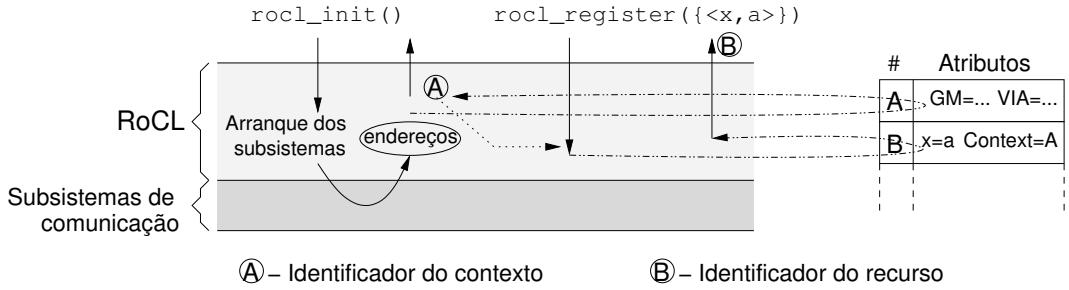


Figura 3.4: Correspondência entre recursos e contextos.

A correspondência entre recursos e contextos é efectuada conforme representado na figura 3.4. Os contextos são registados automaticamente no arranque da biblioteca, sendo tratados como recursos de sistema. Os endereços de portos dos subsistemas de comunicação são usados como atributos dos contextos. A biblioteca R_OCl usa o identificador global de um contexto, devolvido pelo serviço de directório, como um atributo implícito de qualquer recurso registado pela aplicação. Assim, todos os recursos são etiquetados com o identificador do contexto no qual estão inseridos.

Esta abordagem é, à primeira vista, particularmente ineficiente, dado que são necessários três passos para o envio de uma simples mensagem: *1)* o contexto associado ao recurso deverá ser obtido através de uma consulta ao directório, *2)* os endereços de comunicação baixo-nível associados ao contexto deverão ser obtidos analogamente e, finalmente, *3)* a mensagem poderá ser enviada, 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, para guardar 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.

3.3.2 Despacho de mensagens

Os recursos R_OCl são animados por fios-de-execução, o que se traduz num acesso concorrente/paralelo aos mecanismos de comunicação. Além disso, a recepção de mensagens, por parte de uma 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 de sistema e filas de mensagens. Os fios-de-execução de sistema, um por cada subsistema de comunicação suportado, aguardam mensagens, recorrendo à sondagem e a mecanismos de tratamento de interrupções, e armazenam as mensagens recebidas numa fila de recepção. Os recursos acedem a esta fila e obtêm mensagens, com base em alguns critérios de selecção – o destino, a origem e a etiqueta da mensagem. A primitiva de recepção oferecida pelo R_OCl possibilita recepções bloqueantes e temporizadas, com base no parâmetro `timeout` (ver tabela 3.1); um valor negativo para este parâmetro especifica o comportamento bloqueante.

A primitiva de envio entrega uma mensagem directamente a um subsistema de comunicação, o qual, com recurso ao hardware do interface de comunicação, se encarrega do envio efectivo da mensagem. Deste modo, a aplicação fica de imediato livre para prosseguir a sua execução. No entanto, 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 de sistema, para além de tratarem das mensagens recebidas, também monitorizam a fila de envio, por forma a processar envios pendentes.

É importante referir que, devido ao facto de as filas de envio e recepção apenas manipularem registos descritores de mensagens, os quais contêm um apontador para os dados da mensagem, nenhuma cópia adicional de memória é introduzida.

Detalhes da multiplexagem

A generalidade dos sistemas de comunicação baixo-nível oferecem mecanismos para comunicação entre nós de um *cluster*, e não entre entidades variadas do mundo aplicacional. Como tal, por norma, são oferecidas abstracções, para identificação de origens/destinos de mensagens, em número reduzido. No caso do GM, por exemplo, por cada interface de comunicação, apenas são suportados oito portos de comunicação, o que dificulta o desenvolvimento de aplicações onde múltiplas entidades lógicas necessitam trocar mensagens. Além disso, algumas bibliotecas de comunicação baixo-nível nem sequer suportam o acesso concorrente/paralelo a um ponto de comunicação, através de múltiplos fios-de-execução.

O mecanismo de despacho do R_OCl possibilita que um único porto de comunicação de um dado subsistema seja usado por vários recursos, animados por vários fios-de-execução concorrentes. No que respeita 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 baixo-nível.

Como já referido, a recepção de mensagens, por parte dos recursos aplicacionais, é feita por intermédio de uma fila de recepção. O mecanismo de despacho insere na fila as mensagens destinadas aos vário recursos e, no caso de existir algum fio-de-execução bloqueado, à espera de uma mensagem em particular, e se a mensagem recebida servir esse propósito, encarrega-se de alertar 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 serviço específico de multiplexagem, visto que, por um lado, não é possível que vários fios-de-execução fiquem bloqueados, aguardando mensagens, apenas com recurso às primitivas de uma biblioteca de comunicação 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 penalizadores para o tempo de execução de uma aplicação. De facto, 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, uma baixa frequência de sondagem, a qual aumenta o tempo de resposta na troca de mensagens, influencia drasticamente o desempenho global da aplicação.

O recurso a primitivas bloqueantes das bibliotecas de comunicação baixo-nível, as quais exploram mecanismos de tratamento de interrupções, também podem acarretar alguma degradação de desempenho. Na verdade, este método obriga a comutações de contexto, aumentando o tempo necessário para efectivamente tratar a chegada de uma mensagem; um fio-de-execução bloqueado do sistema de despacho terá que ser escalonado e, posteriormente, ainda haverá necessidade de escalonar um fio-de-execução da aplicação.

No R_OCl houve a preocupação de minimizar o impacto do sistema de despacho de mensagens. Em primeiro lugar, o R_OCl inclui mecanismos para comutar entre a sondagem e o recurso à recepção bloqueante. O funcionamento geral é baseado no modo bloqueante, mas, quando todos os fio-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, por forma a permitir que os fios-de-execução da aplicação

também cooperem no processo de despacho de mensagens, o R_OCl disponibiliza uma primitiva – `rocl_dispatch()` – que pode ser usada pelo programador, ao longo do código da aplicação. A evocação de outras quaisquer 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).

No sentido de minimizar o impacto do despacho de mensagens nas aplicações, alguns autores ([19], [16], [10]) propuseram a inclusão de mecanismos de sondagem no próprio escalonador de fios-de-execução. Apesar dos elevados níveis de desempenho que esta abordagem permite alcançar, tem como inconveniente a necessidade de alterar, ou desenhar de raíz, o escalonador. O uso de um sistema de suporte a fios-de-execução próprio, implementado no nível utilizador, inviabiliza o recurso indiscriminado a bibliotecas de rotinas preexistentes, visto que na sua maioria foram desenhadas para ambientes POSIX.

3.3.3 Escrita e leitura remotas

Em geral, a passagem de mensagens envolve dois intervenientes: o emissor e o receptor. Ao receptor cabe a tarefa de, por cada envio, desencadear a execução de uma operação para consumo da mensagem. Em abordagens do género da adoptada nas mensagens activas [26], não há a execução explícita de uma operação de recepção, no destino, mas uma entidade de sistema encarrega-se do processamento associado à entrega da mensagem.

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

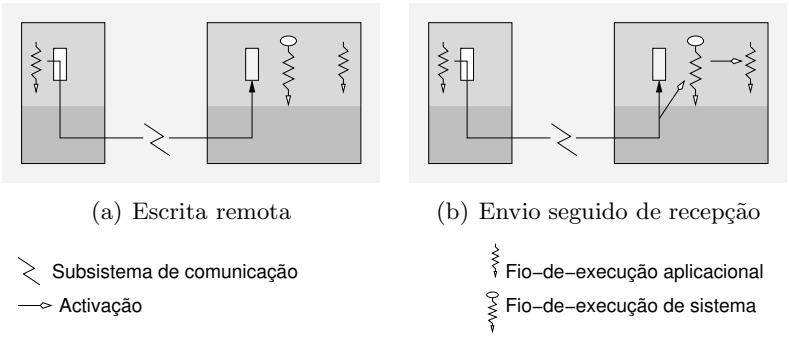


Figura 3.5: Escrita remota *vs* envio-recepção.

No caso do R_OCl, a tradicional passagem de mensagens obriga, no destino, à activação de um fio-de-execução do sistema de despacho, o qual, por sua vez, irá proceder à activação do fio-de-execução que anima o recurso correspondente ao destino da mensagem (ver figura

3.5). Com a integração de operações RDMA, o sistema R_OCl passa a dispor de primitivas de comunicação com tempos de resposta muito próximos do hardware de interligação.

As operações RDMA no R_OCl usam os conceitos recurso e tampão de memória, sem introduzir novas abstracções. As primitivas desenhadas para o efeito (ver tabela 3.4) permitem: *(i)* associar a um recurso um determinado tampão, tornando-o acessível remotamente, isto é, permitindo a escrita e leitura remotas, *(ii)* movimentar um fragmento de um dado tampão local para uma determinada zona do tampão remoto associado ao recurso indicado como destino e *(iii)* movimentar um fragmento do tampão remoto associado ao recurso indicado como origem para uma determinada zona de um dado tampão local.

Tabela 3.4: 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. Obviamente, o sistema R_OCl também mantém uma estrutura de armazenamento intermédio, para associar identificadores de recursos a endereços de tampões, por forma a minimizar estas trocas de informação entre contextos.

Esta abordagem introduz um nível de abstracção superior ao oferecido pelas tradicionais bibliotecas de comunicação baixo-nível, como é o caso do GM e do VIA. Com efeito, os modelos de comunicação destes subsistemas impõe 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.

3.3.4 Controlo de acesso

No modelo de comunicação do R_OCl, um recurso poderá endereçar mensagens a qualquer outro registado no directório, independentemente do nó do *cluster*, da aplicação ou mesmo do utilizador. Na verdade, a estratégia de identificação global coloca num só 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 multiprogramado e multi-utilizador haverá situações onde se torna necessário impor algumas limitações a interacção entre recursos; o R_OCl terá que colocar à disposição dos programadores mecanismos de controlo de acesso, por forma a impedir que a um dado recurso sejam enviadas mensagens indesejáveis.

O mecanismo de controlo de acesso do R_OCl baseia-se num requisito fundamental: 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 subsistemas de comunicação. Se o serviço de directório não responder a pedidos efectuados pela aplicação de um dado utilizador, relativamente a recursos que este não deve ter acesso, a comunicação não terá 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.

A funcionamento deste mecanismo 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 3.1), a qual designa os utilizadores, ou grupos de utilizadores, que podem interactuar com o recurso. Esta lista é armazenada como um conjunto de atributos de sistema, aos quais apenas a biblioteca R_OCl tem acesso.

Os pedidos de pesquisa, enviados pelas primitivas `rocl_query` e `rocl_query_start` ao servidor local, incluem as identificações do utilizador e do seu grupo, as quais são acrescentadas, automaticamente, pela biblioteca R_OCl. Deste modo, o serviço de directório poderá decidir se deve ou não responder a uma dada ordem de pesquisa, 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.

3.4 Implementação sobre GM e VIA

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.

Numa fase inicial foram definidas como tecnologias alvo: Myrinet, Gigabit Ethernet e Fast Ethernet. Dado que o R_OCl se assume como uma biblioteca de nível intermédio, a exploração destas tecnologias faz-se com recurso a bibliotecas de comunicação de baixo-nível, nomeadamente, o GM, para exploração da Myrinet, e o MVIA, para exploração da Gigabit e Fast Ethernet.

3.4.1 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 MVIA 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 logo ao nível do modelo de comunicação, partilham uma característica fundamental: são bibliotecas de nível utilizador, permitindo contornar o sistema operativo.

Comunicação sem cópias

Tanto o GM como o VIA obrigam ao registo de zonas de memória, por forma a que o processo de troca de mensagens não envolva cópias de memória. O registo de regiões memória envolve a marcação das páginas de memória associadas e a facilitação dos endereços reais ao subsistema de comunicação. O R_OCl inclui uma primitiva para as aplicações obterem previamente um tampão, o qual é usado em envios posteriores. Esse tampão é registado pelo R_OCl, tendo em conta os dois subsistemas de comunicação. Isto significa que as aplicações, antes da evocação de uma primitiva de envio, desencadeiam o processo de registo da zona de memória que contém os dados a enviar.

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. Estes blocos deverão ser suficientemente grandes para as mensagens recebidas pelo hardware de comunicação, 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 antecipar qualquer informação sobre as mensagens que a aplicação espera vir a receber, a tarefa de munir os subsistemas de comunicação de blocos de memória pré-registada 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 constantemente reabastecer os subsistemas de comunicação, por forma a que nenhuma mensagem deixe de ser entregue.

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, por forma a 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 tuplo $\langle controlador, porto \rangle$, para identificar cada porto. Cada controlador possui um identificador numérico único (no contexto do *cluster*), atribuído por um serviço GM, enquanto que os portos são numerados de 1 a 8, no contexto de 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*); duas aplicações (ou componentes aplicacionais) que pretendam trocar mensagens deverão possuir um VI, cada uma, e esses VIs deverão estar emparelhados.

Por norma, 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, o qual está conectado a um dado VI remoto.

No MVIA, os VIs estão limitados 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.

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 recepção em particular, nomeadamente, endereços de tampões, tamanho dos dados a enviar/receber e outra 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 o destino 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, o que simplifica o processo de recepção de mensagens de tamanhos variados.

As primitivas disponibilizadas pelo GM, para o envio e recepção de mensagens, levantam um problema adicional, pelo facto de não ser segura a sua utilização em ambientes com múltiplos fios-de-execução. No caso do VIA tal não constitui um problema, mas a implementação MVIA 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, os quais não levantam dificuldades nas primitivas não bloqueantes, mas requerem 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.

3.4.2 Gestão de tampões

O R_OCl utiliza uma colecção de tampões pré-alocados e pré-registados, por forma a assegurar a comunicação sem cópias. Dado que, tanto a alocação como o registo de zonas de memória são operações pesadas, 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 3.6 apresenta o sistema de gestão de tampões usado pelo R_OCl.

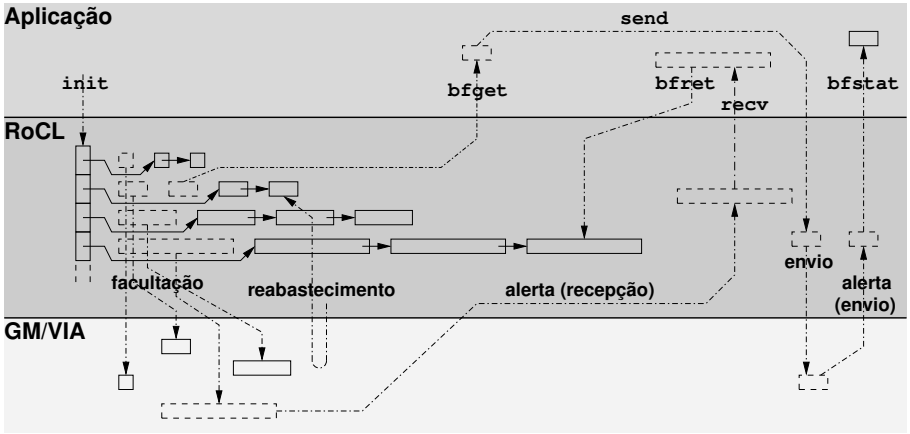


Figura 3.6: Gestão de tampões.

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 sua colecção, sendo assim evitados 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, que o poderá utilizar para efectuar outro envio.

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 proceder à sua devolução à biblioteca R_OCl, a qual o integrará na colecção de tampões que gere (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`), para que esta proceda à reintegração automática do tampão, logo após o alerta de conclusão de envio. Desta forma a aplicação não terá que evocar as primitivas `rocl_bfstat` e `rocl_bfret`.

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, ficando a saber qual o tampão (endereço) onde a mensagem se encontra. 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, o R_OCl será capaz, desde que a aplicação devolva tampões atempadamente, de 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.

Tamanhos de tampões

Para envio de mensagens, a aplicação solicita tampões com tamanhos adequados, conforme os dados a enviar. O R_OCl apenas terá que reservar algum espaço no tampão, para que possa incluir informação de controlo.

Na recepção, a biblioteca terá que lidar com mensagens de tamanho variável, sendo difícil 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 assim a variedade de tampões manipulados; o tamanho de um tampão corresponderá a uma potência de base 2, estando

o expoente 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 tamanho de tampão possível.

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, tal que: $2^{y-1} < (x + inf.controlo) \leq 2^y$.

Formato dos tampões

A manipulação de um tampão, no R_OCl, pressupõe a delimitação de quatro zonas distintas, conforme se mostra na figura 3.7. Uma zona inicial é usada para guardar informação de controlo relativa ao tampão, informação esta que será de carácter genérico, de interesse para qualquer subsistema de comunicação em utilização, ou de carácter específico (um fragmento por cada subsistema). Uma outra zona, em princípio correspondente à maior fracção do tampão, é destinada ao armazenamento da mensagem propriamente dita, sendo de seguida incluída informação de controlo relativa à mensagem. Dependendo do tamanho dos dados da mensagem, haverá a considerar uma última zona do tampão que ficará livre.

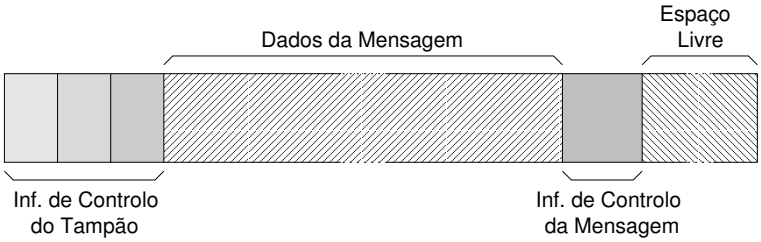


Figura 3.7: Formato de um tampão.

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.

3.4.3 Despacho de mensagens

O mecanismo de despacho do R_OCl terá que gerir um número reduzido de abstracções fornecidas pelas bibliotecas de baixo-nível, por forma a multiplexar mensagens de inúmeras entidades do nível aplicacional – recursos. 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 não.

Endereçamento baixo-nível

O interface do R_OCl com os vários 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. Como já foi referido, o objectivo é fazer chegar a mensagem ao contexto R_OCl, no qual o recurso alvo foi criado.

GM: Na fase de arranque do contexto, é aberto um porto, o qual é anunciado através do registo dos identificadores numéricos do controlador e do porto no serviço de directório. Deste modo, o R_OCl será capaz de converter identificadores de recursos em portos GM.

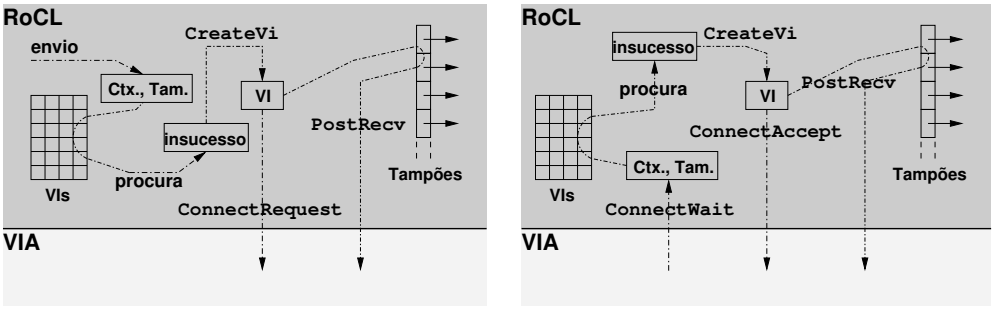
VIA: Na fase de arranque do contexto, é registado no directório o endereço de rede VIA, o qual é composto por um endereço MAC – o endereço físico do controlador de rede – e um discriminador. O discriminador VIA pode ser entendido como um porto UDP ou TCP, mas o programador é livre de usar qualquer sequência 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, o R_OCl poderá solicitar o estabelecimento de uma conexão com a contexto destino adequado, por forma a fazer chegar uma mensagem a um dado recurso.

Detalhes das conexões VIA

Dado que, o estabelecimento de uma conexão é uma operação pesada, 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 uma única vez: quando, pela primeira vez, é endereçada uma mensagem a um recurso desse contexto remoto.

O recurso a 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, em cada contexto, mantém um conjunto de conexões por cada contexto remoto – uma conexão para cada tamanho de tampão possível.

A figura 3.8 apresenta os passos associados ao estabelecimento de uma conexão entre dois contextos.



(a) Origem (b) Destino

Figura 3.8: Gestão de conexões VIA no R_OCl.

No contexto origem (figura 3.8(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 faculta 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 3.8(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.

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, a qual 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 respeita 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.

3.4.4 Controlo de fluxo

As bibliotecas de comunicação 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. No entanto, 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, de um tampão de dimensão adequada, no destino, 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 MVIA, 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 significa que a mensagem não tenha sido descartada, pelo controlador, pelo facto de não existir um tampão.

Apesar de o mecanismo de despacho do R_OCl 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 atempadamente 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 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 encarrega-se 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 MVIA, não existe nenhum mecanismo que facilite a implementação de um mecanismo 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 (MVIA), terá como efeito a inserção da mensagem na fila de envio do mecanismo de despacho.

3.4.5 Reencaminhamento de mensagens

No *cluster* representado na figura 2.1(a), 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 MVIA.

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.

O reencaminhamento propriamente dito, do qual é apresentado um exemplo na figura 3.9, 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 ou-

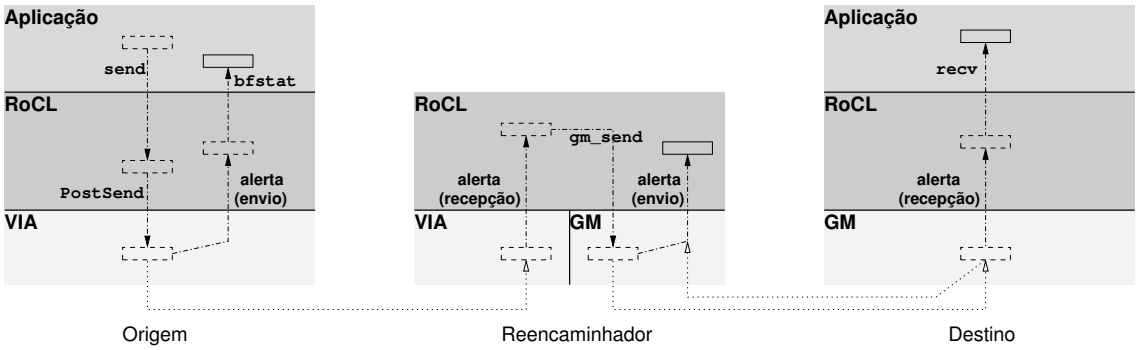


Figura 3.9: Exemplo de reencaminhamento de mensagens.

tro 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 MVIA, independentemente de existir, no tampão, 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 3.1). 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 do R_OCl é, obviamente, limitado, tolerando apenas um único nó intermédio, entre a origem e o destino. No entanto, este nível de reencaminhamento é 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.

3.4.6 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 tem uma decisão importante a tomar, sempre que a aplicação evoca a primitiva de envio: qual o subsistema de comunicação a usar?

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 MVIA. O critério fundamental de selecção usado no R_OCl é o desempenho. Neste sentido, por norma, é usado o GM, o qual garante melhor desempenho (ver secção 5.2).

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 o R_OCl opta 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.

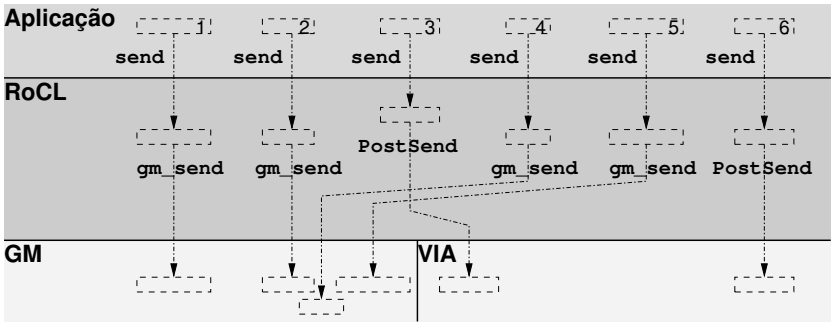


Figura 3.10: Selecção do subsistema em envios consecutivos.

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, fazendo actuar o mecanismo de controlo de fluxo, e só então recorrer ao MVIA, se procedesse ao envio alternado de mensagens através dos dois subsistemas. Na figura 3.10

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.

3.4.7 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 MVIA, esse tamanho é fixado em 32kbytes, tamanho esse que corresponde à dimensão máxima de um segmento de dados no MVIA. Segundo estudos apresentados em [15] e mais tarde corroborados em [6], 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 recurso 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 MVIA 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 arbitráro.

A abordagem seguida no R_OCl compreende a fragmentação da mensagem correspondente à zona de memória indicada pelo programador, conforme se mostra na figura 3.11. 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 do R_OCl. 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

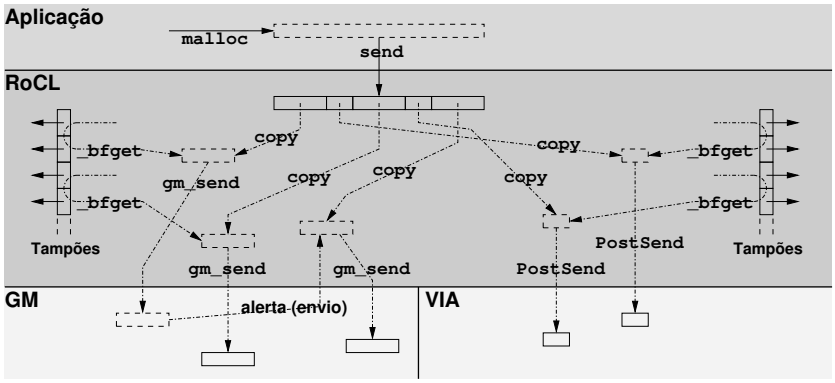


Figura 3.11: Despacho dos fragmentos de uma mensagem.

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.

3.4.8 Escrita e leitura remotas

Como já referido, 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 MVIA, apesar de a especificação VIA contemplar ambas as operações (leitura e escrita), apenas são suportadas operações de 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 de desempenho elevados. 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 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.

3.5 Difusão selectiva

As bibliotecas de comunicação baixo-nível, por norma, apenas oferecem primitivas para comunicação ponto-a-ponto. Em alguns casos, são ainda fornecidas 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 endereçadas mensagens, com o intuito de suportar difusão selectiva; é o caso dos grupos PVM ou dos comunicadores MPI. Esta funcionalidade é normalmente implementada à custa da comunicação ponto-a-ponto oferecida pelas bibliotecas de comunicação baixo-nível, com a preocupação de proceder a optimizações no sentido de explorar particularidades do hardware de comunicação normalmente usado em *clusters* [17].

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.

3.5.1 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. No entanto, o R_OCl inclui uma primitiva – `rocl_set_rel(int ogid, int dgid, enum rocl_rel rel)` – que 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 mensagens endereçadas ao ascendente também lhe são entregues (representado pela seta contínua e unidireccional do esquema da figura 3.12(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 3.12(b)).

Os esquemas 3.12(c) e 3.12(d) apresentam o encadeamento de relações de filiação e de emparelhamento, respectivamente, enquanto o esquema 3.12(e) exemplifica a combinação dos dois tipos de relações. X e Y representam recursos que endereçam mensagens a outros

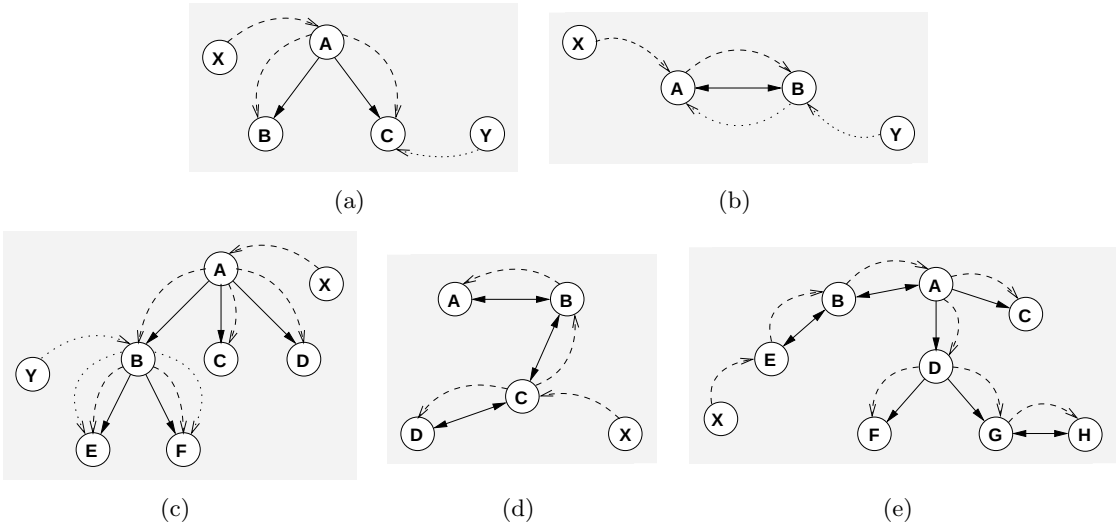


Figura 3.12: Exemplos de relacionamento de recursos e sequente entrega de mensagens.

recursos com relações estabelecidas, enquanto que as setas curvas a tracejado ou ponteado representam a entrega das respectivas mensagens.

Esta abordagem permite que o R_OCl disponha de facilidades de comunicação multiponto, sem que sejam introduzidas novas abstracções; os recursos continuam a ser a única entidade endereçável, no que respeita 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 nome do grupo (o recurso 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 está ao nível das utilizadas no sistemas ALMI [23] e SCRIBE [5], as quais constituem infra-estruturas de difusão selectiva de nível aplicacional.

3.5.2 Pressupostos da abordagem

O esquema de suporte ao relacionamento de recursos, com o intuito de delimitar os múltiplos destinos de uma operação de difusão selectiva, levanta algumas questões, cuja resolução é fundamental.

Unicidade

O mecanismo de registo de recursos oferecido pelo R_OCl, por omissão, não garante a unicidade dos recursos registados. Isto significa que é possível registar vários recursos usando os mesmos atributos. Note-se que, preceder a operação de registo de uma simples operação de consulta, para verificar se o recurso ainda não existe e pode, portanto, ser registado, 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, num sistema paralelo onde vários componentes cooperam, torna-se difícil decidir quem deve criar este recurso.

Neste sentido, a primitiva de registo `rocl_register` inclui o parâmetro `unique` (ver tabela 3.1), o qual permite indicar se se pretende que o registo não tenha efectivamente lugar, no caso de já existir registado um recurso equivalente. 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 membros destes são essenciais no suporte à comunicação colectiva. No R_OCl, a dificuldade surge na gestão de relações, ou seja, a gestão dos membros de um grupo, visto que o problema da nomeação é facilmente ultrapassado devido à utilização do recurso como abstracção única e dado 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, ao contrário do PVM, por exemplo.

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 `recl_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 de *hash* 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, a forma imediata de implementar a difusão selectiva passa pelo envio da mensagem ao recurso ascendente (que pode ser entendido como gestor do grupo) o qual, pelo facto de conhecer todos os descendentes, pode proceder ao envio de uma cópia da mensagem a cada um deles. No caso de um descendente ser ascendente de um outro conjunto de recursos, uma cópia da cópia recebida será enviada a cada um deles. Esta abordagem é semelhante à seguida no DECK [4], com a vantagem de, na realidade, existirem vários pontos centralizadores distribuídos pelos vários nós do *cluster*.

No caso do emparelhamento, o mecanismo de entrega é mais óbvio. Visto que as relações são estabelecidas entre pares, sem que exista a criação de uma entidade destinada a nomear um determinado conjunto de recursos, o mecanismo de entrega funcionará como um reencaminhamento de mensagens.

Note-se que, o mecanismo flexível de estabelecimento de relações oferecido pelo R_OCl não impede a criação de ciclos. A utilização de um tempo de vida para as mensagens 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, o R_OCl verifica se a mensagem é uma repetição, com base em informação de controlo incluída pela origem.

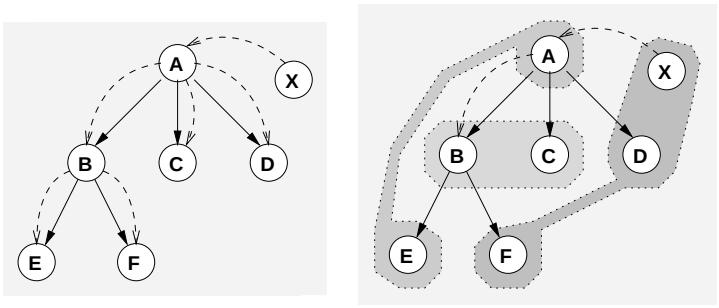
3.5.3 Difusão selectiva optimizada

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 3.13(a) apresenta um cenário de entrega de uma mensagem a vários recursos encadeados por via de relações de filiação, sendo efectuados seis envios. Na figura 3.13(b), os mesmos recursos encontram-se distribuídos por três contextos e a entrega da mesma mensagem envolve apenas dois envios. Este exemplo permite realçar duas optimizações importantes:

- quando mais que um descendente directo pertencem ao mesmo contexto (é o caso de B e C), deve ser enviada uma única mensagem;
- quando um descendente pertence a um contexto no seio do qual já existe uma cópia da mensagem (é o caso de D, E e F), deve ser evitado o envio.



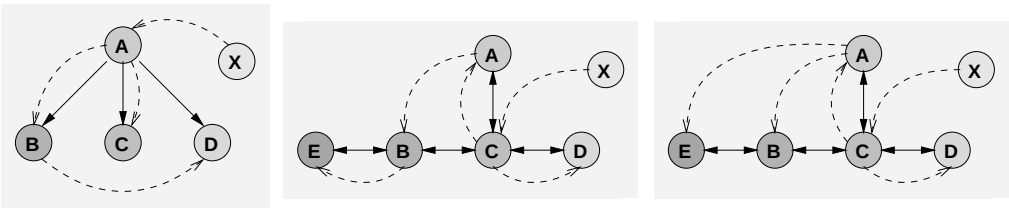
(a) (b)

Figura 3.13: Difusão por contextos.

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.

Envios em paralelo

Tendo como base a difusão por contextos, existe ainda a possibilidade de recorrer a técnicas que garantem menores tempos de latência, como é o caso das árvores de dispersão.



(a) (b) (c)

Figura 3.14: Delegação de responsabilidade de envio.

A figura 3.14(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.

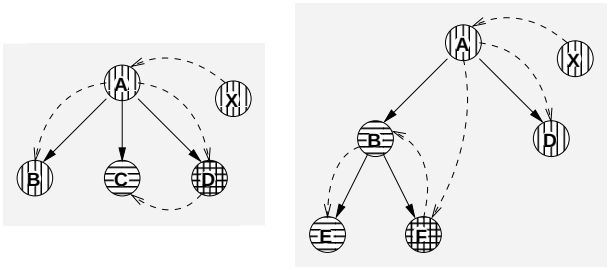
A figura 3.14(b) mostra uma situação análoga para os recursos pares do recurso C.

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 (figura 3.14(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 sistemas de despacho. Na figura 3.14(c), o sistema de despacho do contexto do recurso A, 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 3.4.5). Na difusão de mensagens, o recurso aos serviços de reencaminhamento pode, em alguns casos, ser evitado, 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 3.15: Difusão por tecnologias.

Na figura 3.15 são usados padrões com linhas verticais ou horizontais, para distinguir recursos com acesso a duas tecnologias de comunicação distintas. 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 recurso a 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 de *hash* usadas para gestão de relações. No entanto, os casos que envolvem recursos que distam mais que um relacionamento, como acontece na figura 3.15(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.

3.6 Operação em ambiente *multicluster*

O principal objectivo do R_OCl é a operação em ambientes baseados num único *cluster*, composto por vários *subclusters*. Neste sentido, o R_OCl implementa funcionalidades de reencaminhamento, para garantir a conectividade total dos recursos, independentemente da tecnologia disponível em cada subcluster, e recorre ao UDP para garantir o funcionamento global do serviço de directório, no pressuposto de que todos os nós do *cluster* estão interligados por Fast Ethernet.

No entanto, devido à necessidade concreta de interligação de dois *clusters* geograficamente distantes, o R_OCl também inclui funcionalidades para suporte à operação em ambiente *multicluster*.

3.6.1 Directório multinível

O primeiro obstáculo à operação em ambiente *multicluster* prende-se com a estratégia de identificação global dos recursos – geração de identificadores únicos – e com as pesquisas globais.

Quanto ao primeiro aspecto, optou-se por reservar 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 respeita às pesquisas, dado que o R_OCl recorre à difusão UDP, ao nível do *cluster*, a qual não é viável quando o universo de acção é alargado a múltiplos *clusters*, torna-se necessário organizar o serviço de directório por níveis. Note-se que, regra geral, os nós de um *cluster* não podem ser alcançados a partir do exterior, pelo facto de possuírem endereços IP inválidos.

Pesquisas *multicluster*

O serviço de directório do R_OCl contempla a utilização de um representante por *cluster*, instalado numa única máquina possuidora de um endereço IP válido, para efeitos de interface com o exterior. Desta forma, os servidores de directório poderão redireccionar ordens de pesquisa para *clusters* remotos, através do envio dos pedidos correspondentes aos representantes desses *clusters*.

A figura 3.16 apresenta o mecanismo de pesquisa *multicluster*. Sempre que um servidor difunde, ao nível do seu *cluster*, um pedido, devido ao facto de não o poder satisfazer localmente, e se nenhuma resposta remota for recebida num determinado período de tempo,

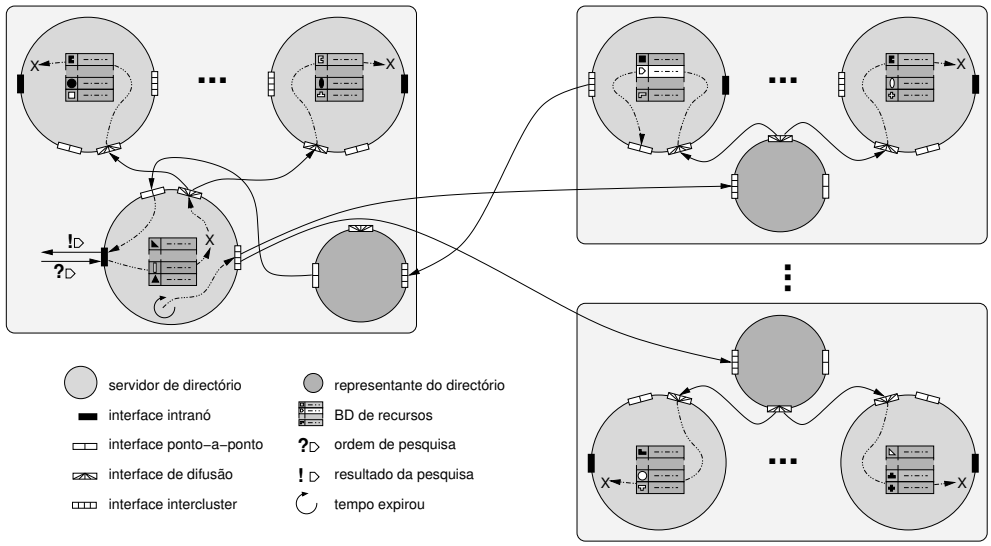


Figura 3.16: Mecanismo de pesquisa *multicluster*.

é desencadeado um processo de pesquisa *multicluster*. São então enviados pedidos, em sequência, a cada um dos representantes de *clusters* remotos conhecidos, representantes esses 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. Esse representante reenviará a resposta ao servidor que despoletou a pesquisa *multicluster*, o qual 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 3.2.4, tendo agora em conta que algumas repostas serão oriundas de servidores em execução em nós de *clusters* remotos.

Esta abordagem 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) [28]. 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 [22] e pelo NLB (Network Load Balancing) [20] 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 3.1 e 3.3), 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 tentar encontrar respostas;
- `SUBCLUSTER` – no caso de a base de dados local não possuir um 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.

3.6.2 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 do *subcluster* do representante 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 contexto 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 algumas 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 esgotada a capacidade da tabela de conexões que mantém. Os servidores de directório também implementam o mesmo mecanismo de gestão de conexões.

3.7 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 constituir um sistema heterogéneo, com vários *subclusters*, 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 – operação em ambiente *multicluster*.

A implementação actual da sistema R_OCl tomou por base as bibliotecas de comunicação baixo-nível GM e MVIA. É oferecido um interface único, o qual permite tirar partido das funcionalidades mais relevantes de ambas as bibliotecas e ultrapassar o desempenho alcançável com recurso unicamente a uma delas, através da utilização conjugada de ambas, em nós multiconectados. No entanto, a arquitectura do R_OCl foi mantida suficientemente flexível, por forma a permitir incorporar facilmente novos subsistemas de comunicação.

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*, visto que, o papel do R_OCl começa apenas na fase de registo dos recursos. Tais mecanismos são deixados a cargo das camadas de software de níveis superiores.

Capítulo 4

Biblioteca para organização de recursos

Os conceitos da modelação de aplicações orientada ao recurso foram apresentados no capítulo 2, sem que houvesse qualquer preocupação em expor as ferramentas ao dispor dos administradores e programadores, para o desenvolvimento e execução de aplicações. Na verdade, a plataforma $m_{\varepsilon}\mu$ corresponde a uma elevação 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 no capítulo 3, é agora possível expor os detalhes relativos ao interface $m_{\varepsilon}\mu$, nomeadamente, a forma como algumas primitvas são implementadas, explorando as 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 do $m_{\varepsilon}\mu$, principalmente no que respeita à manipulação das entidades apresentadas no capítulo 2, para manipulação de recursos físicos e lógicos.

4.1 Configuração do sistema

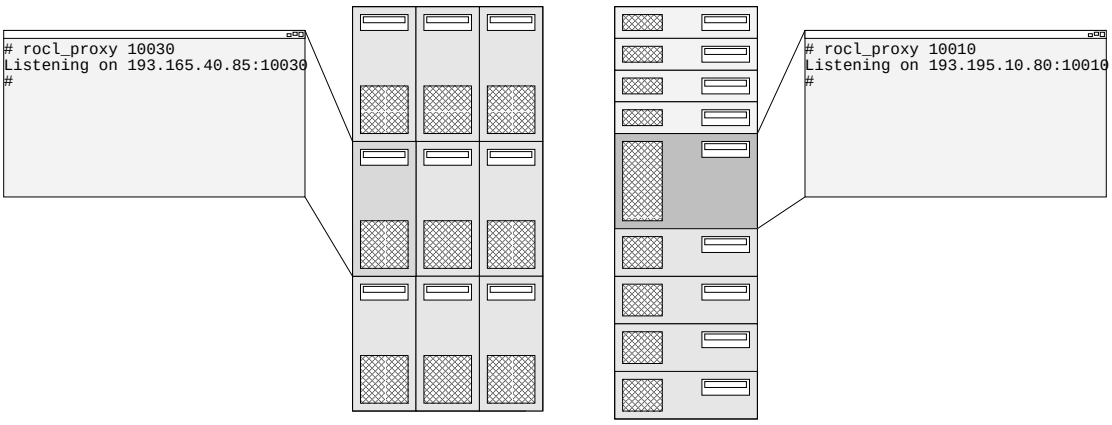
A plataforma $m_{\varepsilon}\mu$ apresenta-se ao programador como uma biblioteca de primitivas, que permitem codificar aplicações em concordância com o paradigma da orientação ao recurso.

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.

4.1.1 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.

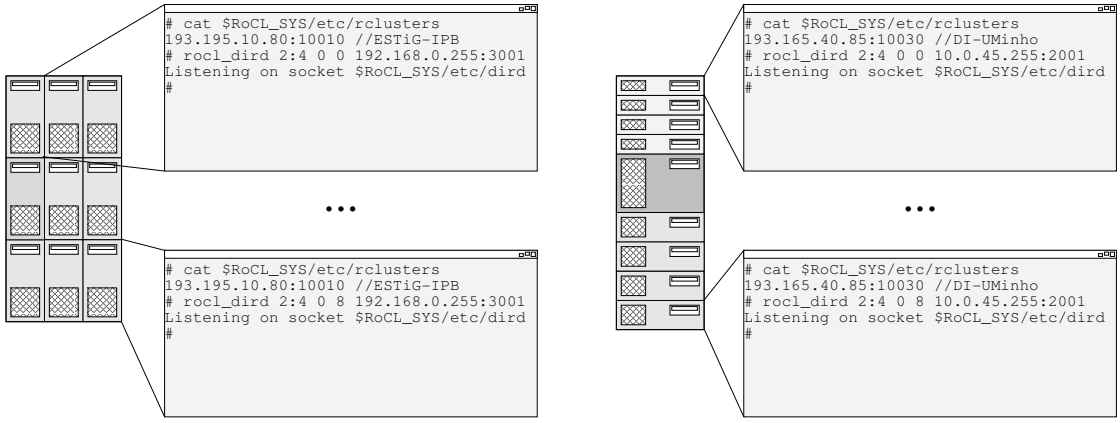
No caso de se pretender a exploração de vários *clusters*, será necessário, antes de mais, instalar um representante R_OCl em cada *cluster*. Para tal, deverá ser seleccionada, em cada *cluster*, uma máquina com um endereço IP válido e, preferencialmente, com múltiplos interfaces de comunicação, no caso de se tratar de um *cluster* heterogéneo. A figura 4.1 mostra o arranque de representantes em dois *clusters*, localizados em Braga e Bragança.



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

Figura 4.1: Arranque de representantes.

De seguida deverão ser colocados em execução os servidores de directório, um por cada máquina de cada *cluster*. 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 4.2.



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

Figura 4.2: Arranque de servidores de directório.

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 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 $m_\mathcal{E}\mu$ 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.

4.1.2 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 4.3.

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

Figura 4.3: Sintaxe para especificação de um domínio físico.

Com base nessa linguagem, a hierarquia de recursos apresentada na figura 2.2 pode ser facilmente especificada, conforme se constata pela figura 4.4. Essa especificação é processada pelo programa `meu_phyparse`, o qual se encarrega de registar, no directório R_OCl, cada um dos domínios especificados.

```
/* 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 4.4: Especificação dos recursos físicos de um *cluster*.

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 4.4, por exemplo, será constituído, ao nível do R_OCl, como um recurso $R = \{\langle"tipo","domínio"\rangle, \langle"nome","Nó\ \ A1"\rangle, \langle"ascend", 1003\rangle, \langle"CPU", 2\rangle, \langle"Mem", 512\rangle\}$, assumindo que 1003 foi o identificador global atribuído pelo directório ao domínio *Subcluster Dual PIII*.

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 4.4, 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.

4.2 Navegação na hierarquia de recursos

A hierarquia de recursos, iniciada a partir a partir da 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 4.1. 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.

4.2.1 Pesquisas imediatas

Excluindo as primitivas `meu_get_properties`, `meu_lookup` e `meu_lookup_aggreg`, as restantes primitivas constantes da tabela 4.1 traduzem-se em simples ordens de pesquisa, a desencadear à custa de evocações às primitivas `rocl_query` e `rocl_query_start`.

A primitiva `meu_getmy_gid`, que permite às tarefas, as únicas entidades activas do sistema

Tabela 4.1: Primitivas $m_{\varepsilon}\mu$ para navegação na hierarquia de recursos.

```
int meu_getmy_gid()
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)
```

$m_{\varepsilon}\mu$, 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. Este valores são armazenados com base 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, asscociados a um recurso. A informação em causa, para cada uma destas primitvas, 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 4.2, também é evocada uma única vez a primitiva primitiva `rocl_query`. De facto, a lista de originais de um recurso é armazenada como um único atributo R_OCl.

Tabela 4.2: Primitivas 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_kill_gidl(meu_gidl_t *gids)
```

Já no que respeita 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.

4.2.2 Pesquisas baseadas em propriedades

As primitivas `meu_get_properties` e `meu_lookup` requerem o cálculo de 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 4.3.

Tabela 4.3: Primitivas 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_kill_prpl(meu_prpl_t *prps)
```

As propriedades que incluem um valor diferente *NULL*, para além de um nome, 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 extraordinário 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 do código da operação de acumulação, juntamente com a demais informação da propriedade, no directório R_OCl.

Cálculo de propriedades

Os vários passos associados à obtenção das propriedades de uma entidade são os representados pela operação *P()*, descrita em linguagem algorítmica, por questões de brevidade, na figura 4.5.

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

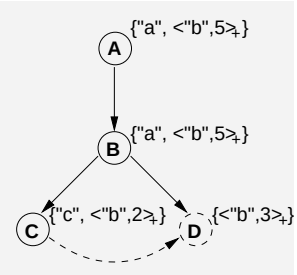
$\underline{IP(x)}$
$\quad y \leftarrow ascendant(x);$
$\quad foreach\ z \in (\{y\} \cup originals(y))$
$\quad\quad p \leftarrow p \cup OP(z) \cup IP(z);$
$\quad return(p);$

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

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

Figura 4.5: Cálculo das propriedades de uma entidade.

No algoritmo aprsentado, a operação *OP()* designa a obtenção das propriedades directamente associadas a uma entidade, operação essa que se traduzirá na evocação da primitiva `rocl_query`, enquanto a operação $\bigtriangledown$ representa a acumulação de propriedades (com base nas operações de acumulação associadas). A figura 4.6 apresenta o funcionamento desta última operação, 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.



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

Figura 4.6: Reunião de propriedades.

Dado que, a existência de pseudónimos pode levar a que uma entidade "receba", 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 falsa repetições, visto que a normal reunião de propriedades elimina aquelas com nomes e valores idênticos.

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.

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

Figura 4.7: Localização de uma entidade.

A figura 4.7 apresenta o algoritmo correspondente à primitiva `meu_lookup`. Quando se conlui 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 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 4.7; 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 acumulação inversa, por forma a implementar a subtracção de conjuntos de propriedades (operador $\setminus$).

4.2.3 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 no conjunto de todoas as entidades.

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

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

Figura 4.8: Localização de entidades com eventual agregação.

O funcionamento da primitiva `meu_lookup_aggreg`, descrito pelo algoritmo da figura 4.8, envolve dois passos. Em primeiro lugar é produzida uma lista de entidades, directa ou indirectamente descendentes da entidade indicada como parãmetro, que cumprem os requisitos da primeira lista de propriedades especificada, isto é, que garantem todas as propridades

indicadas nas várias subárvores que representam. 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 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 noma e lista de propriedades indicados.

4.3 Epílogo

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

No seu estado actual, são apenas expostos os procedimentos para a preparação inical do ambiente de suporte à execução e os mecanismos e estratégias que permitem a navegação na hierarquia de recursos físicos e lógicos.

Serão, posteriormente, adicionadas as secções respeitantes à criação de entidades lógicas, ao envio de mensagens e ao acesso a memória global. Existirá uma última secção, que apresentará, de forma sucinta, algumas funcionalidades avançadas da plataforma.

Capítulo 5

Avaliação de desempenho

A avaliação da biblioteca $m_{\mathcal{E}}\mu$ e, consequentemente, do R_OCl, foi efectuada através de um conjunto de testes sintéticos, que permitiram obter informação relativa ao funcionamento de variados componentes e ao funcionamento do sistema, como um todo.

Neste capítulo são apenas avaliadas as funcionalidades $m_{\mathcal{E}}\mu$ que não dependem de decisões do programador, ou seja, aquelas funcionalidades que, basicamente, correspondem a evocações directas de primitivas R_OCl. De facto, a panóplia de primitivas que o $m_{\mathcal{E}}\mu$ oferece, para organização de entidades lógicas e selecção de recursos físicos, insere-se num modelo de programação próprio e, portanto, proceder à sua avaliação é uma tarefa complexa. Refira-se que, o aparecimento do $m_{\mathcal{E}}\mu$ é justificado pela necessidade de oferecer aos programadores funcionalidades mais convenientes, por comparação com as disponibilizadas pelas plataformas de programação paralela tradicionais, e a avaliação da conveniência de uma plataforma requer um período longo de utilização da mesma.

Todos os testes que conduziram aos resultados aqui apresentados foram realizados num *cluster* composto por 9 máquinas, distribuídas por 3 *subclusters*, cujas características são apresentadas na tabela 5.1.

Tabela 5.1: 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+1.28 Gbits/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 MVIA, são usadas as versões 2.0.8 e 1.2, respectivamente.

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

5.1 Serviço de directório

O desempenho global do $m_{\varepsilon}\mu$ depende, em grande parte, do desempenho do serviço de directório implementado ao nível do R_OCl. Neste sentido, foi desenvolvido um pequeno programa que, através de sucessivas evocações da primitiva `meu_get_info`, permite avaliar as taxas máximas de pesquisa suportadas pelo directório.

Note-se que, as pesquisas em causa são simples, envolvendo uma única resposta que, neste caso, tem um tamanho de 256bytes. Deste modo, é eliminada toda a complexidade subjacente à navegação na hierarquia de recursos, a qual dependeria da aplicação e das opções do programador. As pesquisas simples são também aquelas que o próprio sistema R_OCl desencadeia, por forma a descobrir endereços de comunicação associados a recursos.

5.1.1 Operação local

Nas situações mais favoráveis, as pesquisas no directório, desencadeadas pela evocação de primitivas R_OCl, pelo sistema $m_{\varepsilon}\mu$ ou pelo próprio sistema R_OCl, são resolvidas no contexto do nó onde são desencadeadas. Nestes casos, apenas o servidor de directório local é contactado, sem que haja troca de mensagens entre nós. Deste modo, 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 5.1 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 questionar o servidor local. São apresentados dois cenários, correspondentes a dois tipos de máquinas: do lado esquerdo da figura 5.1 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.

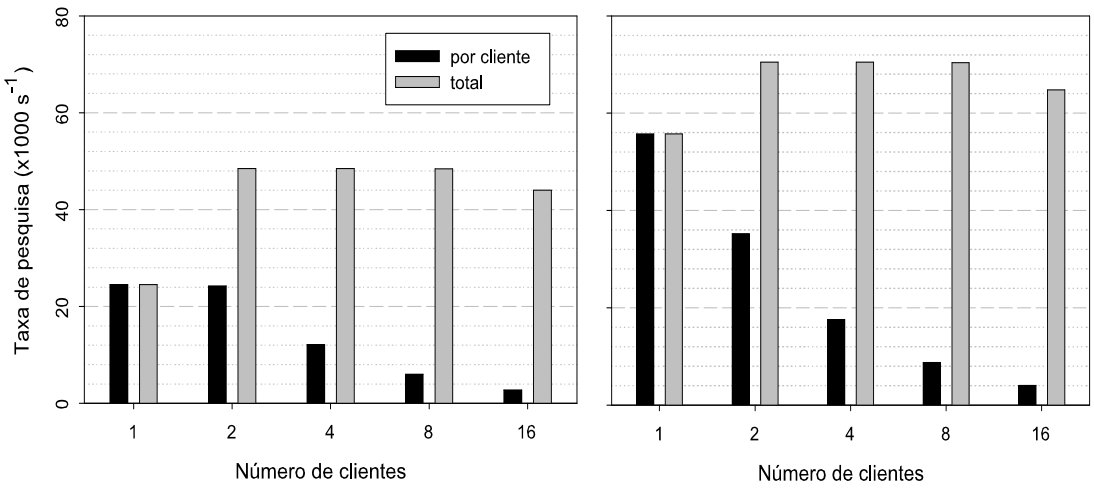


Figura 5.1: Taxas de pesquisa máximas na operação local.

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 suficientes 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 um acesso equitativo ao 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.

5.1.2 Operação global

As ordens de pesquisa 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 um recurso (através do seu identificador) que, garantidamente, se encontra 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.

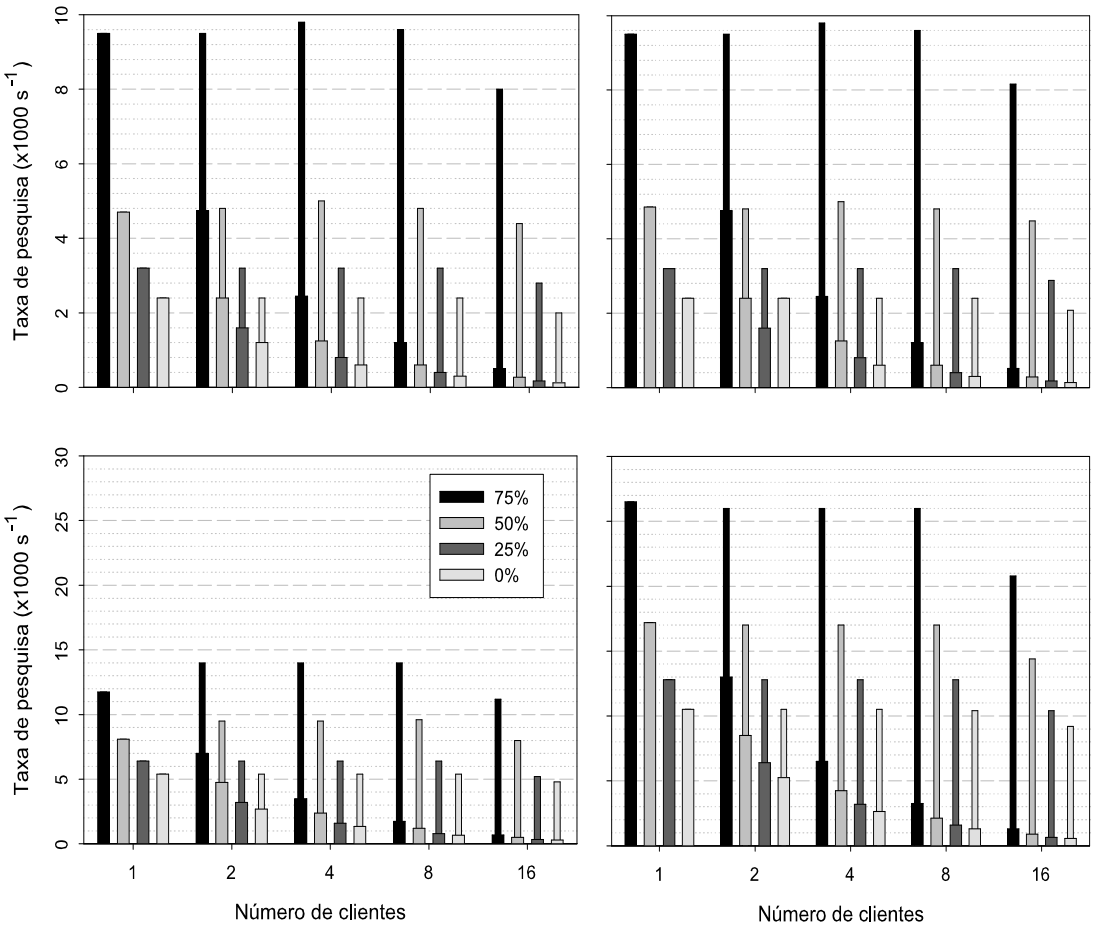


Figura 5.2: Taxas de pesquisa máximas na operação global (4 nós).

A figura 5.2 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). 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, o sistema R_OCl foi parametrizado por forma a não usar 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 5.2), 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 5.2) 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 necessárias máquinas com elevados níveis de desempenho, por forma a levar à exaustão a rede Fast Ethernet. Ainda mais surpreendente é verificar 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, em particular a difusão UDP, requer 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.

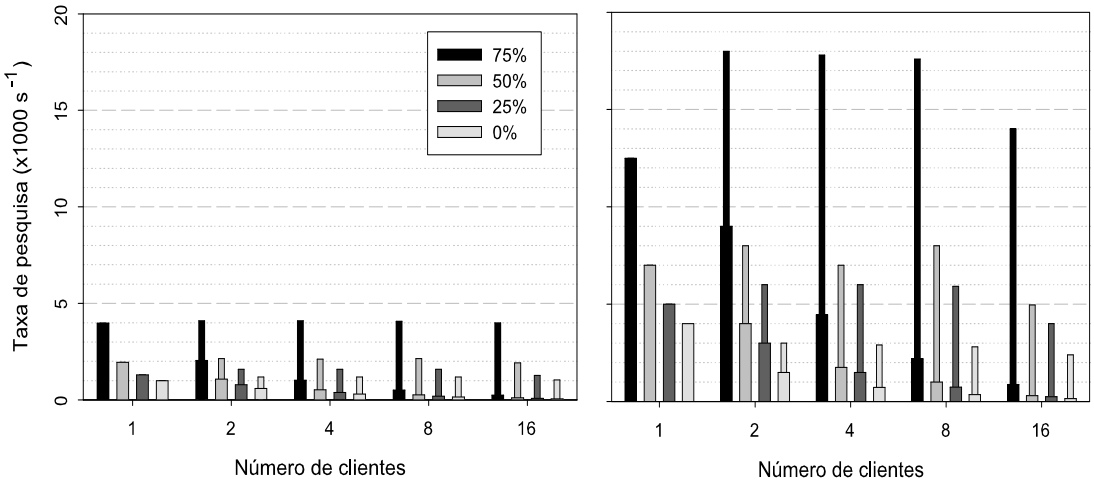


Figura 5.3: Taxas de pesquisa máximas na operação global (8 nós).

Por fim, a figura 5.3 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 via Fast Ethernet (gráfico do lado esquerdo da figura 5.3) ou Gigabit Ethernet (gráfico do lado direito da figura 5.3).

Importa realçar que o aumento do número 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 5.3, com as dos gráficos na parte superior da figura 5.2). 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 5.3, com as do gráfico no canto inferior esquerdo da figura 5.2). Se, em vez dos resultados obtidos no *subcluster* Athlon, for usada, como referência, uma média grosseira dos resultados apresentados nos gráficos da parte inferior da figura 5.2, então o decréscimo seria apenas de cerca de 10%. Deste modo, pode-se concluir que a tecnologia Gigabit proporciona uma boa escalabilidade ao serviço de directório, ao contrário da tecnologia Fast Ethernet.

5.2 Comunicação ponto-a-ponto

A comunicação ponto-a-ponto no $m_{\varepsilon}\mu$ 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.

5.2.1 Troca de mensagens intranó

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

Deste modo, o envio de uma mensagem a uma tarefa ou caixa postal, a partir de uma tarefa local (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 se enquadrar no operão ao qual pertence a tarefa origem, mas se estiver inserido no mesmo nó do *cluster*, então, o sistema R_OCl explora os mecanismos de *loopback* implementados pelo MVIA e pelo UDP[1], os quais garantem desempenhos equivalentes ao IPC e facilitam, significativamente, a implementação da biblioteca.

A figura 5.4 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

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

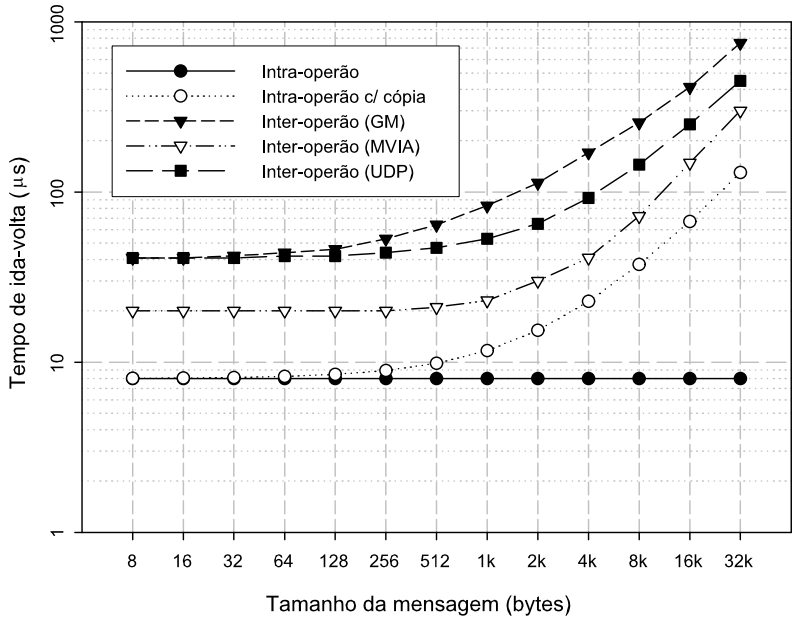


Figura 5.4: Tempo de ida-volta intranó.

mínimo 8μs, que corresponde às operações de sincronização e escalonamento de fios-de-execução do utilizador (usados para implementar as tarefas) e do sistema de despacho. Pode-se ainda concluir que o mecanismo de *loopback* do MVIA 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.

5.2.2 Troca de mensagens *intrasubcluster*

No contexto de um *subcluster*, o $m_{\mathcal{E}}\mu$ explora uma determinada tecnologia de comunicação de elevado desempenho, através do recurso a bibliotecas de comunicação de baixo-nível, ou seja, Myrinet e Gigabit, através das bibliotecas GM e MVIA. Os testes de desempenho levados a cabo neste âmbito também incluíram a utilização do protocolo UDP, para efeitos de comparação, o qual é suportado pelo R_OCl apenas com o intuito de permitir a experimentação em ambientes que não disponham de redes SAN.

Por uma questão de uniformização, todos os resultados apresentados foram obtidos com recurso a 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.

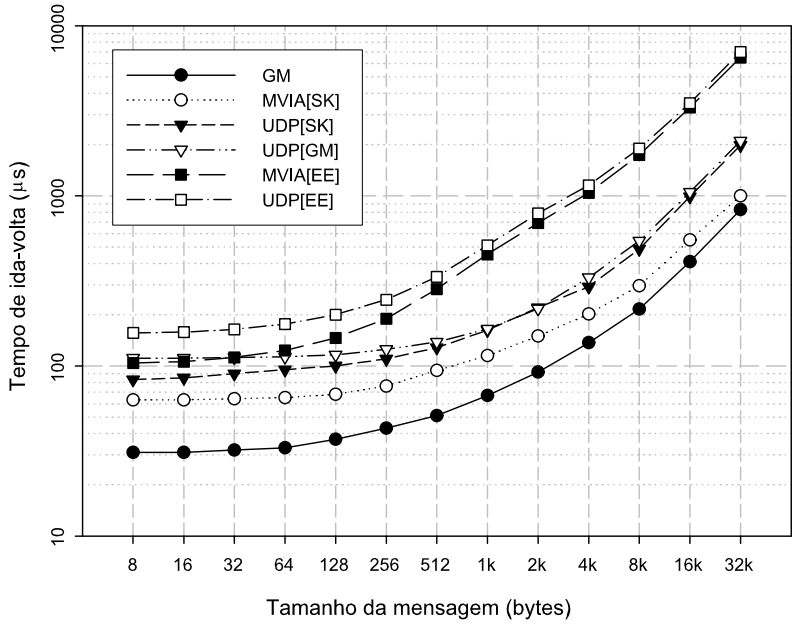


Figura 5.5: Tempo de ida-volta internó, *intrasubcluster*.

A figura 5.5 apresenta os tempos de ida-volta obtidos para a troca de mensagens entre tarefas em execução em máquinas distintas, do *subcluster* i686^{+}. Essencialmente, foi testado o suporte R_OCl para três subsistemas de comunicação: GM, MVIA e UDP. O GM é, evidentemente, usado para explorar a tecnologia Myrinet. O MVIA, como foi referido, é usado para explorar a tecnologia Gigabit (os controladores SysKonnect 9821 são suportados pelo MVIA), mas também permite explorar a tecnologia Fast Ethernet (os controladores Intel Ether Express Pro100, ao contrário dos 3Com 3c905C Tornado, também são suportados). O UDP, por via do suporte incluído no Linux para variados controladores, permite 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 5.5 permite confirmar duas coisas: 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.

5.2.3 Troca de mensagens *intersubcluster*

Um dos objectivos principais do $m_{\mathcal{E}}\mu$ 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 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 recurso ao mecanismo de reencaminhamento do R_OCl.

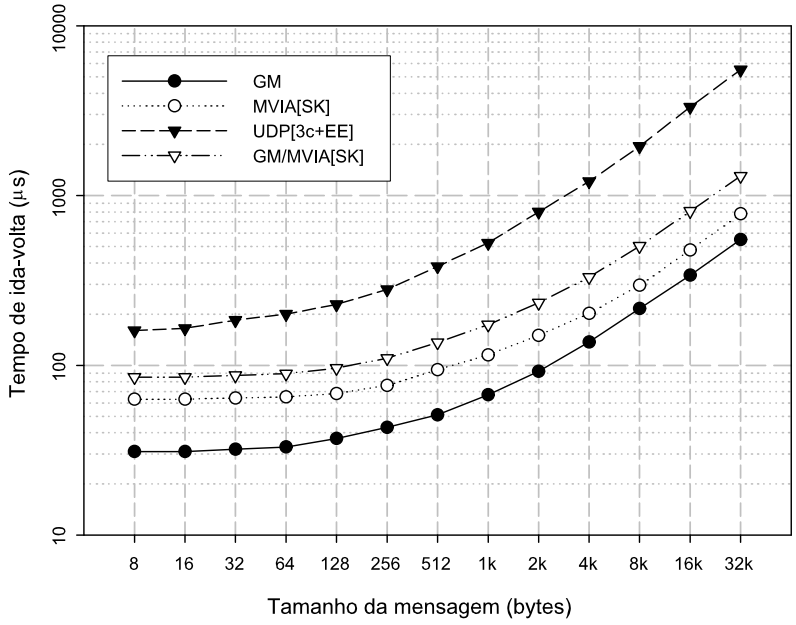


Figura 5.6: Tempo de ida-volta *intersubcluster*.

A figura 5.6 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 da máquina do *cluster* Xeon, na qual executa o serviço de reencaminhamento R_OCl.

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 MVIA). São também apresentados os resultados que se pode alcançar pela utilização da rede Fast Ethernet e do subsistema UDP, os quais constituem um meio de comunicação alternativo entre máquinas dos dois

subclusters[2].

Pode-se concluir que, o reencaminhamento R_OCl permite alcançar desempenhos bastante melhores que aqueles 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.

5.2.4 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 MVIA, de forma transparente às aplicações, nomeadamente, alternando entre os dois no envio de mensagens consecutivas.

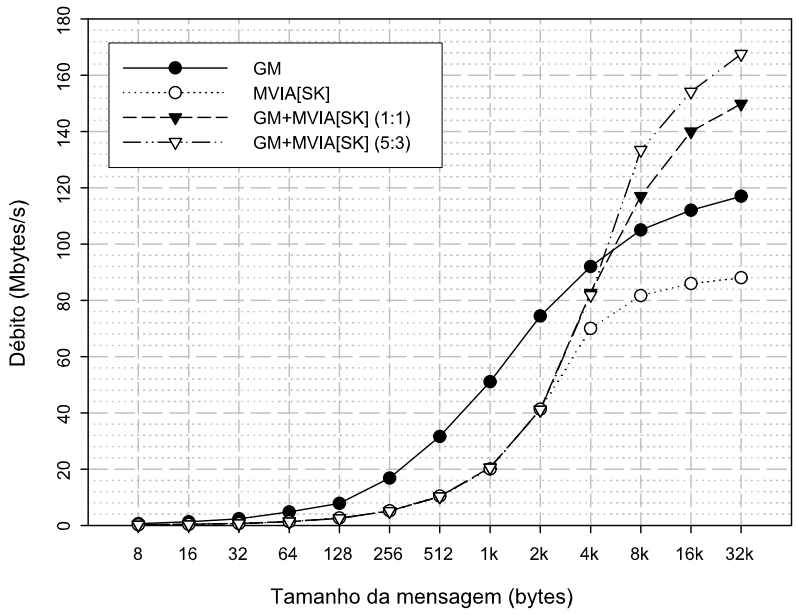


Figura 5.7: Débito na agregação de tecnologias.

A figura 5.7 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 MVIA), juntamente com os valores relativos a

[2]Note-se que, os controladores 3Com 3c905C Tornado não são suportados pelo MVIA.

duas estratégias de agregação de tecnologias.

Como se pode verificar, a simples alternância entre o GM e o MVIA (1 mensagem enviada através do GM, 1 mensagem enviada através do MVIA e assim sucessivamente) já permite ultrapassar o desempenho alcançado usando unicamente o MVIA, 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 advém 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 (de forma mais espaçada no tempo).

Se forem enviadas 5 mensagens através do GM seguidas de 3 através do MVIA 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 este subsistema seja usado para enviar um maior número de mensagens. Uma simples análise às curvas do GM e do MVIA permite concluir que o GM permite atingir um débito cerca de 45% mais elevado, para mensagens de 32kbytes.

5.2.5 Acesso a memória global

O mecanismo de acesso a 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.

A figura 5.8 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 MVIA. 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

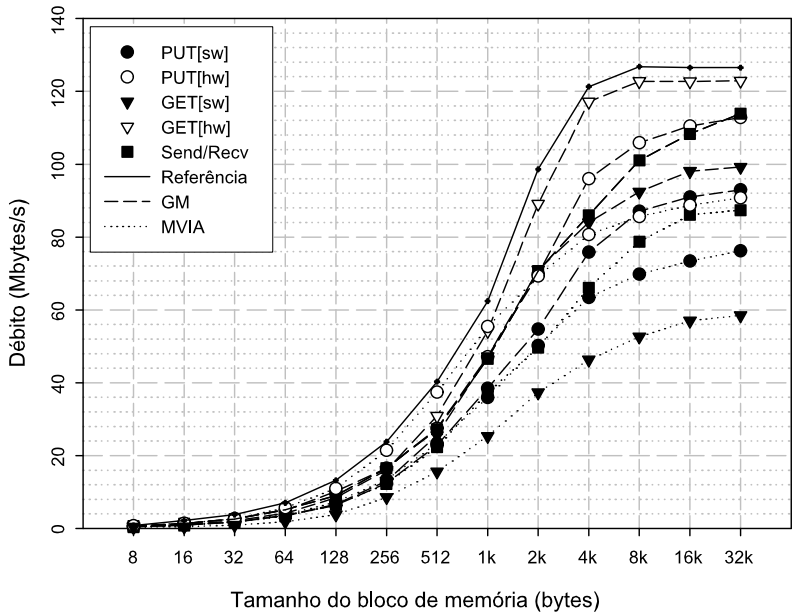


Figura 5.8: Débito nas operações de escrita e leitura remotas.

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 MVIA, como já foi referido, apenas existe suporte RDMA para a operação de escrita.

5.3 Comunicação e múltiplos fios-de-execução

Uma das características mais relevantes do $m_{\varepsilon}\mu$ e do R_OCl 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 do $m_{\varepsilon}\mu$ assumem os fios-de-execução POSIX, oferecidos pelo sistema operativo, como elementos essenciais para a modelação de aplicações e a exploração de nós SMP.

5.3.1 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 funcionali-

dades limitadas, 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 (actualmente já incluída no Linux RedHat 9.0) veio mostrar que a utilização de fios-de-execução POSIX não compromete, necessariamente, o desempenho do sistema de passagem de mensagens.

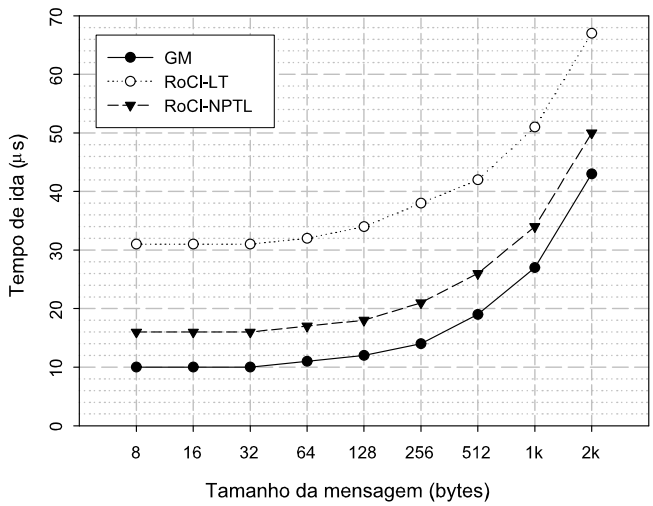


Figura 5.9: Impacto da comutação de contextos no tempo de resposta do envio.

A figura 5.9 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 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, com a introdução do NPTL, que se deve, essencialmente, à diminuição dos tempos de comutação e de sincronização entre fios-de-execução. Note-se que, os 5μs adicionais, que o R_OCl introduz em relação ao GM, são o preço a pagar pela flexibilidade oferecida, isto é, a orientação ao recurso e o modelo de comunicação totalmente assíncrono.

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 vários fios-de-execução consumidores, que tratam esses eventos. Desta forma, na prática, é reproduzido o ambiente subjacente a uma aplicação $m_{\varepsilon}\mu$, do ponto de vista da 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. Os consumidores aguardam por eventos bloqueando-se em variáveis de condição. Como resposta a um evento, um consumidor apenas actualiza uma variável de estado associada a esse evento, indicando que este foi efectivamente tratado.

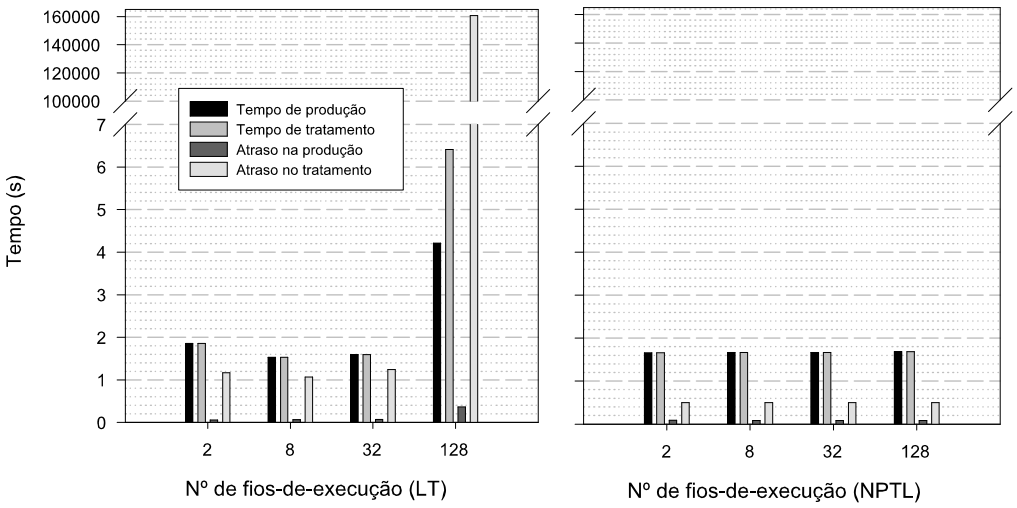


Figura 5.10: Tratamento de eventos com fios-de-execução LT e NPTL.

A figura 5.10 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 calculando 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.

5.3.2 Troca de mensagens concorrente

Como já foi referido, um pressuposto importante nas aplicações $m_{\varepsilon}\mu$ é a utilização de múltiplos fios-de-execução. Portanto, será de esperar que, em determinados momentos da fase de execução de uma aplicação, ocorram acessos concorrentes aos mecanismos de comunicação. Nomeadamente, será natural existirem, num nó computacional, várias tarefas $m_{\varepsilon}\mu$ a trocar mensagens com tarefas em execução num outro nó, por exemplo.

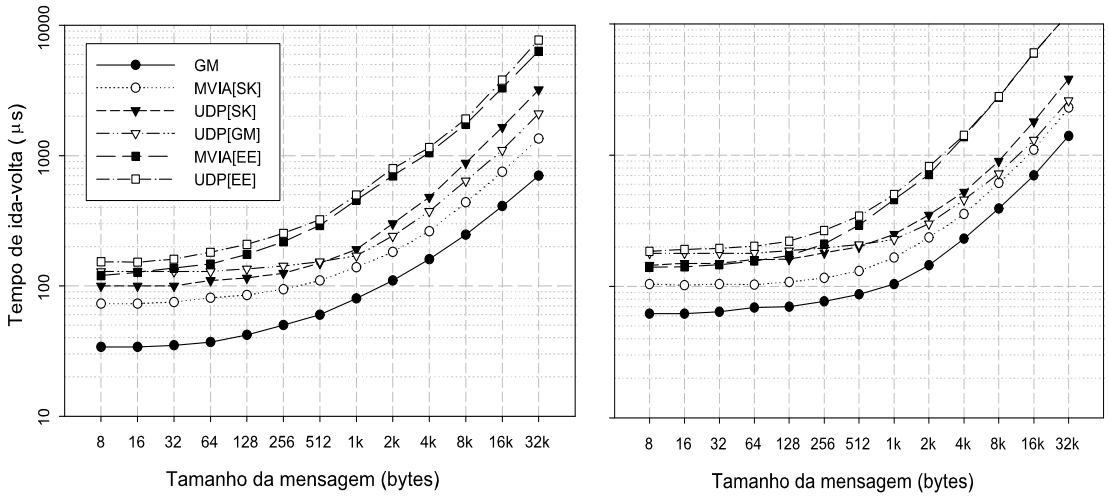


Figura 5.11: Tempos de ida-volta na troca de mensagens concorrente.

A figura 5.11 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. Por forma a facilitar a comparação com os resultados apresentados na secção 5.2.2, relativos à troca de mensagens sem concorrência, foi usada a configuração de hardware aí descrita.

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 acentuada (caso da 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.

5.3.3 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, assincronamente, efectuam 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 esta operação, a máquina em causa, não executou qualquer tarefa adicional (comunicação ou computação). De seguida foram executados os testes de avaliação do tempo de ida-volta, cujos resultados, em condições óptimas, foram apresentados na secção 5.2.2, 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, MVIA sobre Gigabit, etc) e TE_0 representa a taxa de execução obtida inicialmente, sem que tivesse ocorrido qualquer operação de comunicação em si-

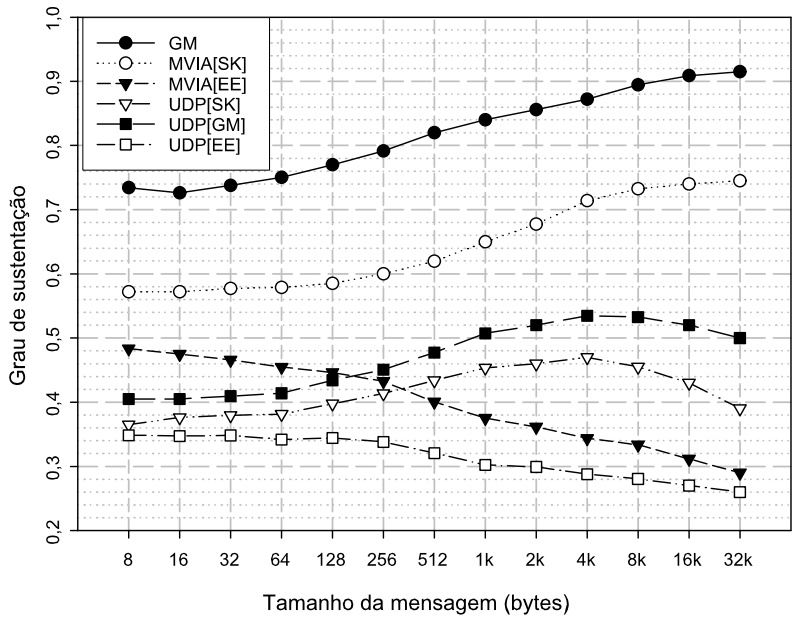


Figura 5.12: Grau de sustentação do desempenho na troca de mensagens.

multâneo. De igual forma, o rácio IV_{SC_0}/IV_{SC}, onde IV_{SC} e IV_{SC_0} representam os tempos de ida-volta obtidos para um dado subsistema de comunicação, com ou sem computação a decorrer em simultâneo, respectivamente, expressa o impacto na comunicação. 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 fundo, para o subsistema GM, o qual garante os tempos de ida-volta mais baixos, no R_OCl.

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

A figura 5.12 apresenta o grau de sustentação do desempenho global 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 mais tempo de processador que o MVIA e o GM, fazendo com que o desempenho caia.

Leitura e escrita remotas

Em relação ao acesso a 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 $(TE_{SC}^{orig} + TE_{SC}^{dest})/(2 \times TE_0)$.

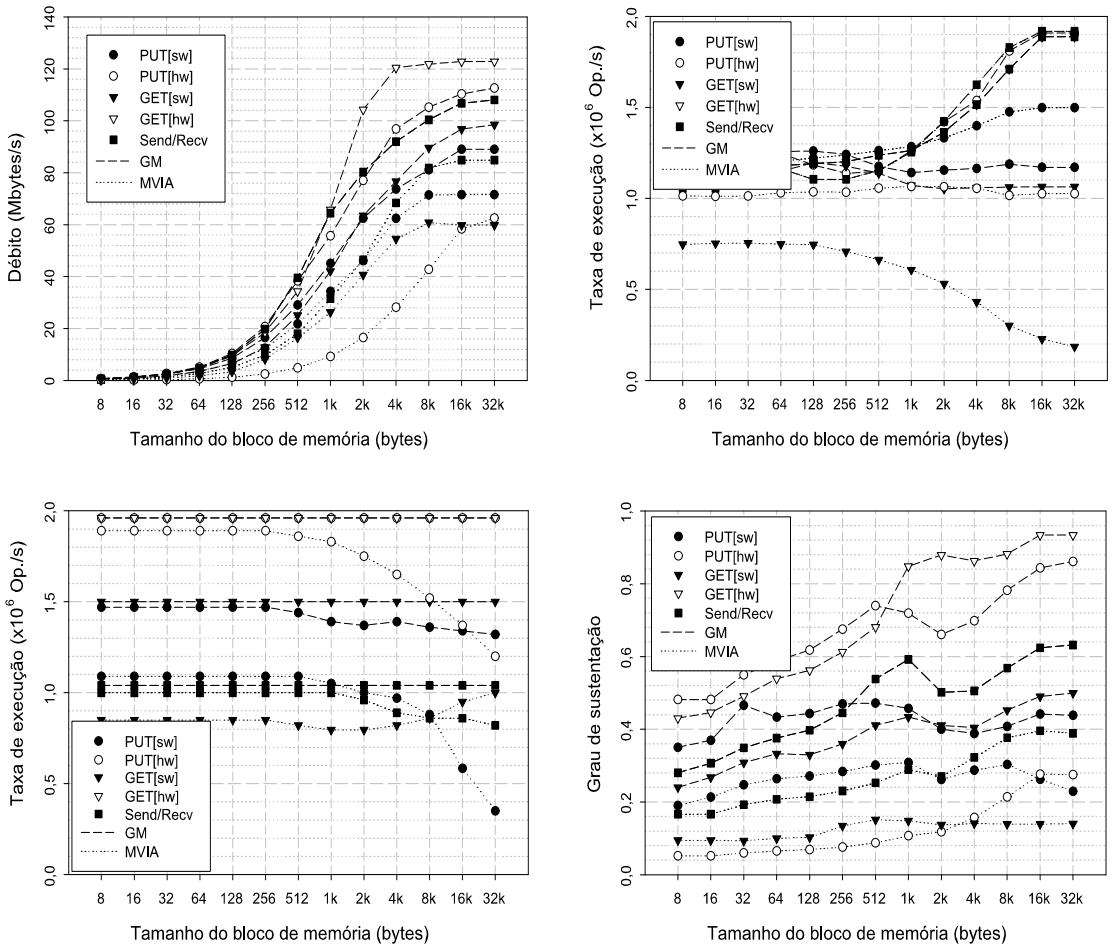


Figura 5.13: Grau de sustentação do desempenho no acesso a memória remota.

A figura 5.13 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 (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 5.8, 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 respeita ao MVIA/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 MVIA, 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[3].

5.4 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 de 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.

Foram apresentados testes de avaliação de desempenho que vão muito para além dos tradicionais testes de largura de banda e tempo de ida-volta que normalmente se usam. Falta ainda apresentar duas aplicações de teste que, em certa medida, capturam a essência de muitas aplicações paralelas, podendo, portanto, ser usadas para obter informação de referência sobre o comportamento que as aplicações do utilizador poderão vir a ter.

[3]Note-se que, a implementação $R_OCl$para a escrita remota, à custa de operações de envio e recepção, para as primitivas de leitura e escrita remotas, implica cópias de memória.

Capítulo 6

Adequação à programação convencional

O modelo de programação do $m_{\varepsilon}\mu$ introduz uma nova abordagem, para o desenho de sistemas de aplicações, mas mantém funcionalidades e conceitos presentes em abordagens convencionais. Na verdade, a maioria dos programadores não está aberta à aprendizagem de uma nova linguagem ou paradigma de programação e, portanto, qualquer nova abordagem deverá adequar-se às capacidades adequiridas pelos programadores ao longo de vários anos de utilização de tecnologias convencionais, como é o caso do MPI e do PVM.

Neste capítulo mostra-se de que forma o $m_{\varepsilon}\mu$ pode ser usado para desenvolver aplicações que recorrem a abstracções e modelos convencionais. A vantagem de, nestes casos, optar pela utilização do $m_{\varepsilon}\mu$, em detrimento de outras plataformas amplamente divulgadas, prende-se com o facto de o $m_{\varepsilon}\mu$ expandir os mecanismos convencionais, integrando-os no paradigma da orientação ao recurso, para além da possibilidade de se poderem usar múltiplos modelos de programação em simultâneo (memória partilhada, passagem de mensagens e memória global).

6.1 Programação com fios-de-execução escalável

A desenvolvimento de programas que usam múltiplos fios-de-execução, recorrendo ao interface POSIX, é uma tarefa difícil, essencialmente devido à necessidade de sincronização explícita no acesso a variáveis partilhadas. Além disso, os sistemas SMP, onde a programação com fios-de-execução assumiu um papel relevante, deixaram de ser uma alternativa atraente para a execução de programas paralelos.

No entanto, algumas aplicações são facilmente modeladas com recurso a 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 $m_{\varepsilon}\mu$.

6.1.1 Um problema de armazenamento e computação

Com o intuito de realçar a importância da integração dos mecanismos de passagem de mensagens com as facilidades para a utilização de fios-de-execução, apresenta-se aqui, como exemplo, 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.

6.1.2 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, por forma a acelerar a conversão de imagens JPEG em mapas de bits e o sequente redimensionamento desses mapas de bits.

A figura 6.1 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.

6.1.3 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-à 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.

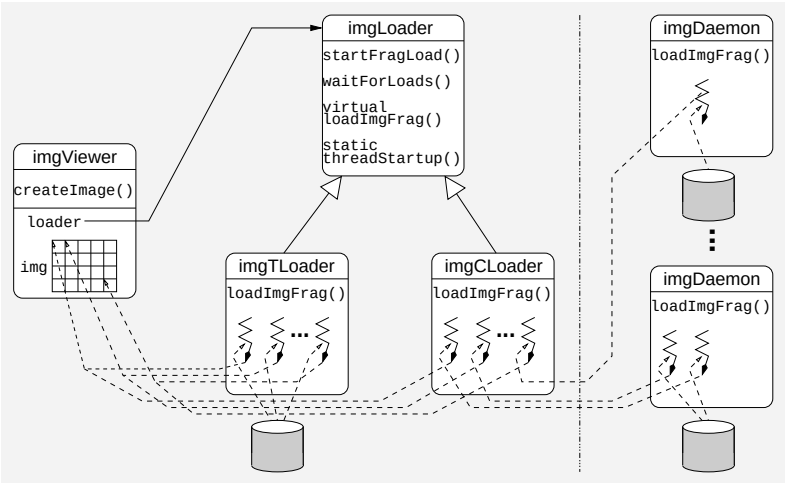


Figura 6.1: Modelo de objectos para o visionamento de paisagens.

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 6.1 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 um tarefa simples:

- 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, com recurso ao $m_{\mathcal{E}}\mu$, é bastante simples, dado que existem primitivas para criação de tarefas remotas e existe a possibilidade de endereçar mensagens, directamente, a tarefas.

6.1.4 Análise da escalabilidade

Por forma a comprovar a viabilidade desta abordagem, foram efectuados alguns testes, envolvendo um *cluster* Myrinet com 4 nós bi-processador e uma máquina SMP com 4 processadores, também conectada ao *cluster*. A tabela 6.1 apresenta o hardware envolvido.

Tabela 6.1: Hardware usado na validação do escalamento de uma solução SMP.

Especificações	Servidor SMP	Nó do *cluster*
Processador	4x Xeon 700MHz	2x PIII 733MHz
Memoria	1Gbyte	512Mbytes
Disco Rígido	Ultra SCSI 160	UDMA 66
Rede	LANai9 Myrinet, 64bits/66MHz	

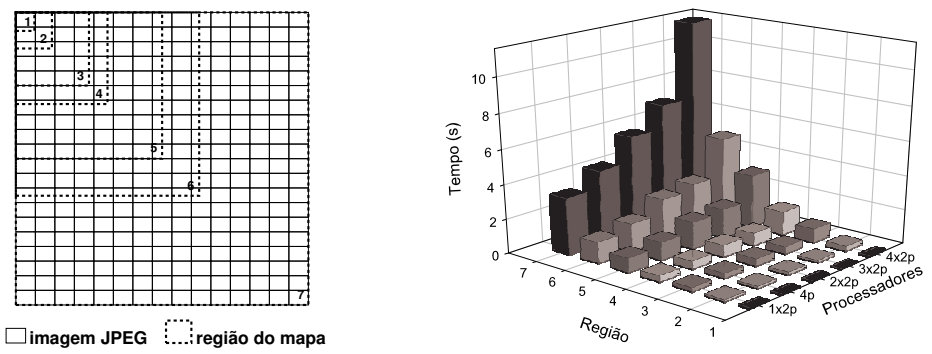


Figura 6.2: Desempenho da aplicação de visionamento em vários cenários.

A figura 6.2 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 o servidor SMP, sendo usado um objecto `imgTLoader` pela aplicação (designação `4p`);
- de 1 a 4 nós do *cluster*, 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 recurso a 2 nós (4 processadores) já consegue ultrapassar os resultados obtidos com o servidor SMP, o qual também inclui 4 processadores. A principal razão é a maior largura de banda disponível para carregamento de imagens JPEG, a partir de disco.

6.2 Passagem de mensagens de alto-nível

O paradigma da passagem de mensagens é, indiscutivelmente, o mais vulgarizado entre os programadores que desenvolvem aplicações paralelas. Para estes programadores, o PVM e/ou o MPI constituem a referência em termos de conceitos e funcionalidade, independentemente de poderem ser usados outros sistemas para o desenvolvimento e execução de aplicações. Deste modo, a apresentação de uma qualquer nova abordagem que integre a passagem de mensagens deverá contemplar o estabelecimento de um paralelo com estes dois ambientes.

Em [13] e [27] são examinados alguns tópicos 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, encerrando conceitos e abstracções fundamentais para qualquer sistema baseado na passagem de mensagens. O modelo de programação do $m_{\varepsilon}\mu$ integra mecanismos que, quando analisados segundo estas vertentes, permitem concluir que, para além da funcionalidade padrão dos sistemas tradicionais de passagem de mensagens, o $m_{\varepsilon}\mu$ proporciona abstracções de alto-nível singulares.

6.2.1 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, por forma a impedir a recepção inadvertida de mensagens. O suporte à definição de contextos (universos para a troca de mensagens) é pois o mecanismo usado para garantir a modularidade básica das aplicações, isto é, 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 ser substituído por outro mais eficiente (ou mais funcional) numa fase posterior, já na fase de execução da aplicação. Note-se que, regra geral, o programador só conclui que um dado módulo não satisfaz as necessidades da aplicação durante a sua execução.

Enquanto a definição de contextos é comum aos sistemas tradicionais, a substituição de componentes não é contemplada por nenhuma abordagem conhecida. No modelo de programação do $m_{\varepsilon}\mu$, as duas vertentes da modularidade são facilmente integradas.

Bibliotecas de funções

Numa aplicação $m_{\varepsilon}\mu$, um contexto pode ser definido de duas formas distintas:

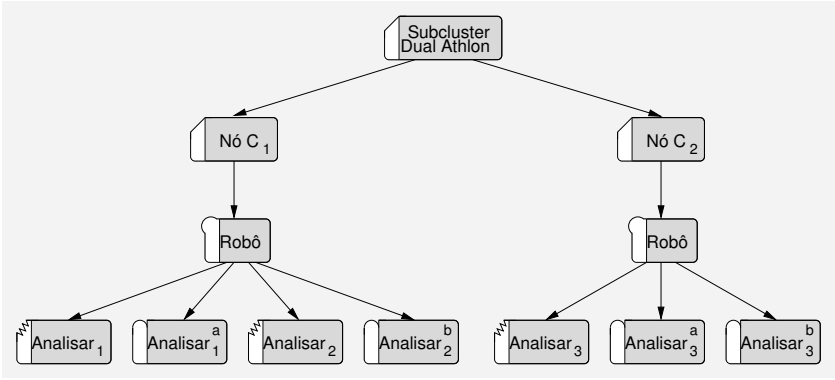
- através da criação de uma caixa postal por cada tarefa envolvida na biblioteca;
- mediante a criação de um domínio contextualizador, abrangendo todas as tarefas envolvidas na biblioteca.

O recurso a 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.

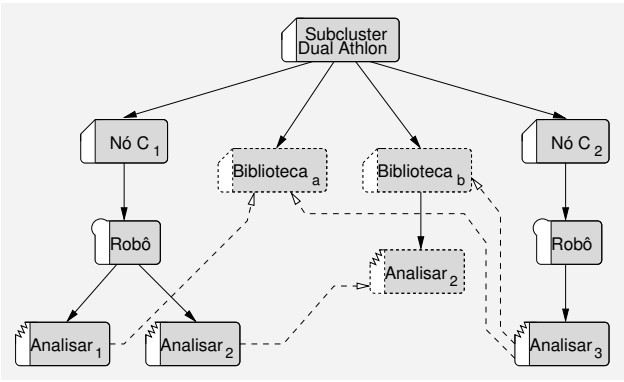
A figura 6.3(a) apresenta um cenário onde três tarefas criam quatro caixas postal, de forma a usarem duas bibliotecas: as tarefas $Analisar_1$ e $Analisar_2$, por fazerem uso das bibliotecas a e b, respectivamente, criam as caixas postal $Analisar_1^a$ e $Analisar_2^b$, enquanto a tarefa $Analisar_3$, 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, cabendo ao programador garantir os emparelhamentos adequados. No entanto, através do mecanismo de controlo de acesso, pode ser indicado que apenas a tarefa criadora tem acesso à caixa postal criada para fazer face a uma dada biblioteca.

Os domínios contextualizadores 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.



(a)



(b)

Figura 6.3: Definição de contextos através de caixas postal e domínios.

Na figura 6.3(b), é apresentada uma solução para o mesmo caso da figura 6.3(a), mas onde a criação de contextos se faz através de domínios pseudónimo. Assim, para cada uma das bibliotecas, é criado um domínio pseudónimo e as tarefas que usam uma determinada biblioteca são feitas membros do domínio correspondente.

Note-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 estejam criadas no momento em que esse contexto é criado, pois os membros de um domínio – os originais e os descendentes – 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 $Analisar_2$.

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 *(2)*, na impossibilidade de entrega de uma mensagem a uma determinada entidade, o sistema de comunicação notifica a aplicação, através do estado associado à mensagem (ver secção 3.4.2), e esta poderá desencadear um processo de reenvio, podendo recorrer ao directório para localizar novamente a entidade. Isto significa que, se uma entidade X (uma tarefa, por exemplo) for desactivada/eliminada e uma Y for criada e registada com as características da primeira, 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), acabarão por recorrer ao directório e encontrarão Y, com quem passarão a comunicar.

A figura 6.4 apresenta dois casos exemplificativos das implicações, do ponto de vista da passagem de mensagens, associadas à substituição de uma entidade simples (uma tarefa) por outras mais complexas (operões ou domínios). Partindo do cenário da figura 6.4(a), onde as tarefas *Descarregar* e *Analisar* estão envolvidas na troca de mensagens, as figuras 6.4(b) e 6.4(c) mostram de que forma a tarefa *Analisar* pode ser substituída por um operão, integrando várias tarefas em execução numa segunda máquina, ou mesmo por um domínio, com tarefas espalhadas por dois operões em máquinas distintas.

No primeiro caso (figura 6.4(b)), a tarefa *Descarregar* passará a enviar mensagens para o operão *Analisar*, podendo qualquer uma das tarefas $Tarefa_1$ e $Tarefa_2$ 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 não deverão contemplar a identificação da origem.

No segundo caso (figura 6.4(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 especias na programamção da tarefa *Descarregar*.

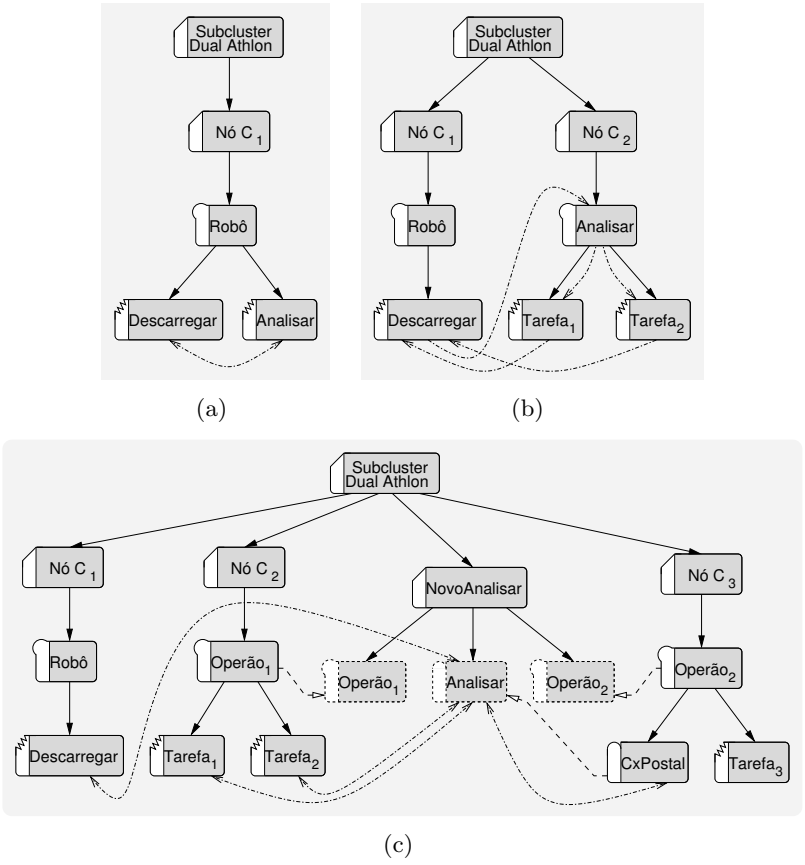


Figura 6.4: Substituição de componentes aplicacionais.

6.2.2 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 sistema (nó do *cluster*) onde a acção deverá ter lugar.

No $m_{\varepsilon}\mu$, a hierarquia de um sistema de aplicações é, naturalmente, evolutível, dado 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 $m_{\mathcal{E}}\mu$

A figura 6.5 ilustra a evolução de uma hierarquia $m_{\mathcal{E}}\mu$ representativa de um sistema de robôs dinâmico.

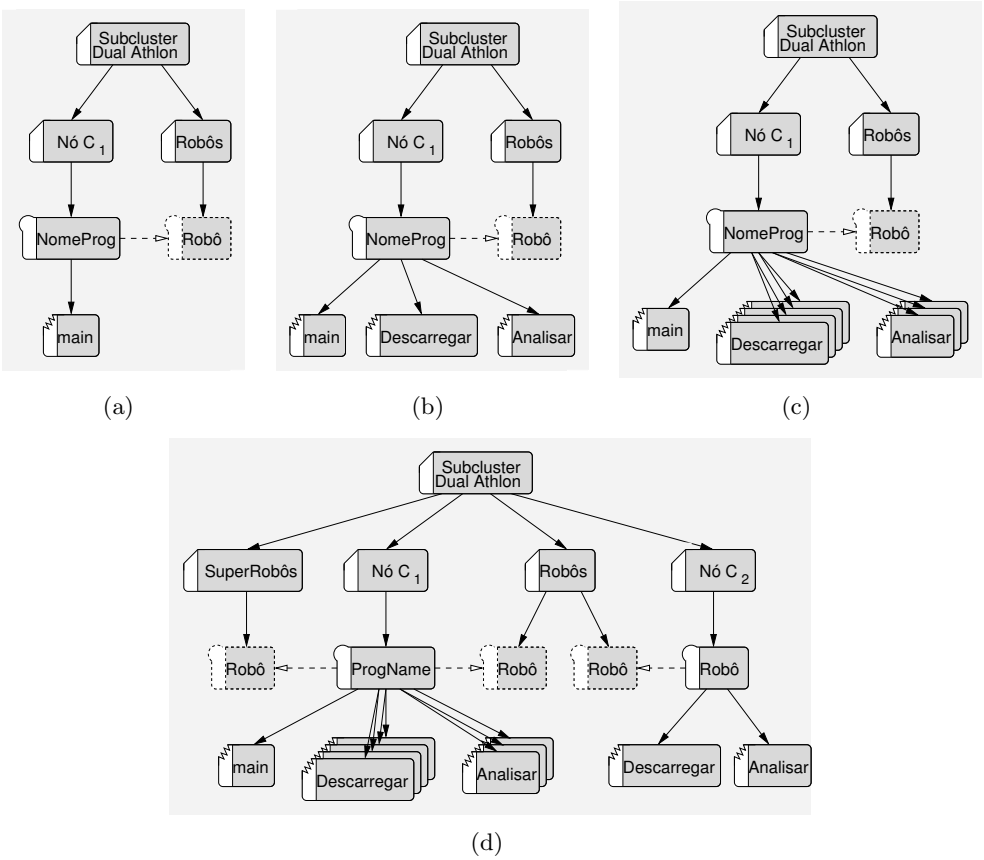


Figura 6.5: Evolução da hierarquia de uma aplicação.

O sistema inicia através do arranque, via linha de comando, de uma dado programa. 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 6.5(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 6.5(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 6.5(c)). Poderá ainda haver lugar à criação de um operão *Robô* numa segunda máquina, através de uma primitiva $m_{\mathcal{E}}\mu$, 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 poderio 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, través de um pseudónimo (ver figura 6.5(d)).

É importante referir que, qualquer um dos estágios representados na figura 6.5 corresponde a uma aplicação em operção, tratando-se, portanto, de uma evolução.

Delimitação de recursos

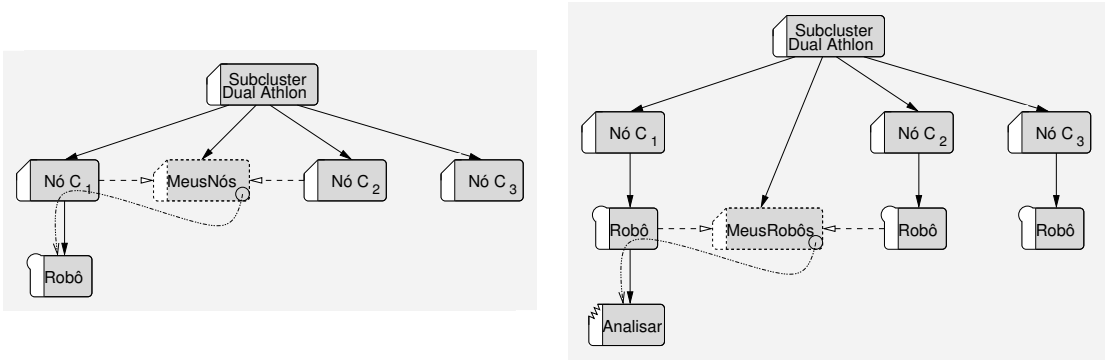
No $m_{\varepsilon}\mu$, a criação de uma entidade lógica sob um determinado recurso físico (a criação de um operão, por exemplo) faz-se através da indicação do identificador do domínio que representa o recurso físico em causa. Este identificador é obtido pelo programador, com recurso ao directório e com base na especificação de uma lista de características genéricas. 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.

No entanto, a grande vantagem do $m_{\varepsilon}\mu$ advém da possibilidade de delimitar recursos físicos (máquinas, por exemplo), com base em características comuns, e deixar a cargo do sistema a selecção de um domínio em paricular (uma máquina específica), no seio do universo delimitado, para a criação de um operão, por exemplo. Esta abordagem é também alargada à delimitação de recursos lógicos (operões, por exemplo), permitindo criar uma tarefa sem que um operão em particular seja especificado.

A figura 6.6(a) mostra um domínio pseudónimo – o domínio *MeusNós* – usado para delimitar 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. A criação do operão *Robô*, tendo sido indicado como alvo o domínio *MeusNós*, coloca no $m_{\varepsilon}\mu$ a responsabilidade de 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 6.6(a) apresenta a criação de uma tarefa



(a) (b)

Figura 6.6: Delimitação de recursos.

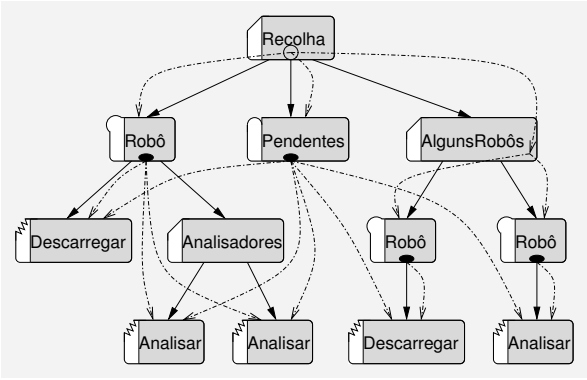
6.2.3 Grupos flexíveis

A constituição de grupos assume epecial importância em aplicações que recorrem à difusão selectiva de informação. Vulgarmente, os sistemas de passagem de mensagens incluem primitivas para 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 endadeamento de grupos e a restruturação de grupos. A figura 6.7 exemplifica as potencialidades do $m_{\varepsilon}\mu$ na manipulação de grupos e correspondente entrega de mensagens.

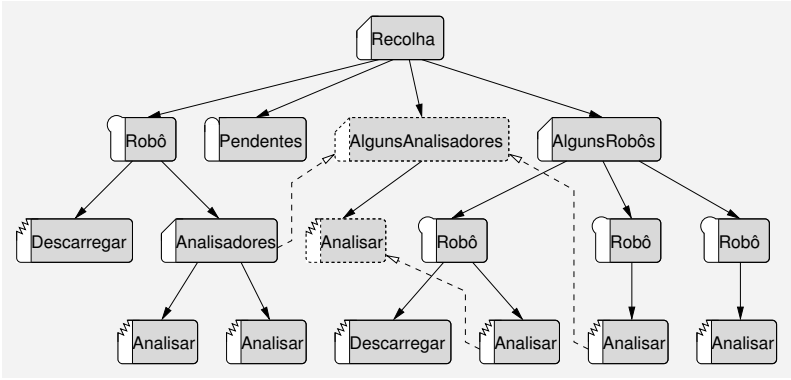
Na figura 6.7(a) podem ser identificados dois grupos distintos, materializados nos domínios *Recolha* e *AlgunsRobôs*. O primeiro inclui como membros um operão, uma caixa postal e um domínio, trantando-se, portanto, de um grupo híbrido. O segundo grupo, que por sinal é membro do primeiro, inclui dois operões.

O envio de uma mensagem ao domínio *Recolha*, por parte de uma qualquer tarefa, incluida 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



(a)



(b)

Figura 6.7: Ecadeamento de grupos híbridos.

consuma duas mensagens: uma no âmbito do seu operão e outra no âmbito da caixa postal.

Na figura 6.7(b) é constituído um grupo através da criação de um domínio pseudónimo. Este domínio traduz uma nova vista para as entidades representadas na figura 6.7(a), o que significa que, no que respeita 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 6.7(a).

6.3 Epílogo

Neste capítulo mostrou-se de que forma as abstracções do $m_{\varepsilon\mu}$ podem ser usadas para programar aplicações em conformidade com paradigmas bem conhecidos. De momento, incidiu-se na programação com fios-de-execução e com conceitos derivados da passagem de mensagens. Falta ainda abordar a programação por evocação de procedimentos remotos, a interoperação estendida e a utilização de memória global.

Capítulo 7

Discussão

O trabalho desenvolvido nesta dissertação usa a orientação ao recurso como base para a criação de num novo paradigma e de uma metodologia que reflectem uma nova abordagem ao desenvolvimento de aplicações paralelas e à 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 $m_\varepsilon\mu$.

7.1 Comunicação orientada ao recurso

A orientação ao recurso encontra na imagem de sistema único 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 *multi-cluster*, por via da hierarquização do serviço de directório e da inclusão do protocolo TCP. Convém, no entanto, distinguir a abordagem seguida ao nível da imagem de sistema único e do serviço de directório, não apenas em termos de objectivos como também de dimensão, do gigantesco trabalho que tem vindo a ser conduzido no âmbito da metacomputação e da *Grid*.

O R_OCl actual oferece ao programador de sistema um interface único, que uniformiza o acesso às bibliotecas de comunicação baixo-nível GM e MVIA, 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.

Através de um número significativo de testes, pôde-se comprovar a adequação do R_OCl a cenários variados e todo o potencial de controlo de computação e comunicação oferecido ao programador para maximizar a utilização dos recursos físicos constituintes de um *cluster* heterogéneo. Em particular, provou-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 baixo-nível.

7.2 Modelação e exploração unificadas

Com base nas funcionalidades do R_OCl, foram criadas abstracções de nível superior, no sentido de oferecer ao programador a possibilidade de uma utilização informada dos recursos físicos de um *cluster*. O programador pode, se assim o entender, fazer uma gestão de recursos mais próxima do equipamento da arquitectura alvo, por forma a garantir níveis de desempenho elevados na execução das aplicações.

As abstracções servem, em primeiro lugar, para a modelação de aplicações, visando uma adaptação ideal das entidades lógicas ao equipamento constituinte do *cluster*, inclusivamente, num cenário 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 não entrar em consideração com a forma como estão organizados os recursos físicos, assumindo, por exemplo, que tem ao seu dispor uma máquina com muitos processadores.

Também neste caso, é a orientação ao recurso que permite unificar a modelação de aplicações e a exploração de recursos físicos, fundindo num único conceito entidades lógicas e físicas.

Ao longo do texto, mostra-se, através de exemplos, a forma como 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, por forma a tirar partido do equipamento disponível.

7.3 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 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.

Ao longo dos capítulos 2, 3 e 4 apresentaram-se os contributos sob a forma de: um novo modelo de comunicação, novas abstracções para a modelação de aplicações e uma metodologia para exploração dos recursos físicos de um *cluster*. Usando, como indicadores, a análise de desempenho apresentada no capítulo 5 e as experiências de modelação presentes no capítulo 6, conclui-se que os resultados alcançados com o $Ro_cme\mu$ são uma resposta qualificada aos objectivos enunciados.

7.4 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 têm 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, de "casamento" entre recursos lógicos e físicos do $m_\varepsilon\mu$, poderia, de alguma forma, ser aplicada à *Grid*. Todavia, para que tal possa vir a acontecer, há ainda um longo caminho a percorrer, no sentido de redesenhar o sistema de registo e armazenamento de informação relativa a recursos, adaptando o serviço de directório do R_OCl à dimensão exigida para gerir os requisitos e facilidades potenciados pela *Grid*.

Alguns dos conceitos do $m_\varepsilon\mu$, nomeadamente, a definição de pseudónimos e os mecanismos de partilha de propriedades, podem vir a ser integradas noutras áreas de conhecimento. Em termos de perspectivas, a formalização matemática dos pseudónimos poderia conduzir à sua aplicação noutros domínios de aplicação.

Noutra direcção, o trabalho realizado pode vir a ser usado em aulas, para a apresentação de conceitos relativos ao desenvolvimento e execução de aplicações paralelas e, também, como instrumento para a avaliação da produtividade do programador.

Finalmente, a consolidação dos 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.

Bibliografia

[1] M. Beck, J. Dongarra, G. Fagg, G. Al Geist, P. Gray, J. Kohl, M. Migliardi, K. Moore, T. Moore, P. Papadopoulous, S. Scott, and V. Sunderam. HARNESS: A next generation distributed virtual machine. *Future Generation Computer Systems*, 15(5–6):571–582, 1999.

[2] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[3] Matthias Brune, Alexander Reinefeld, and Jorg Varnholt. A Resource Description Environment for Distributed Computing Systems. In *International Symposium on High Performance Distributed Computing*, pages 279–286, 1999.

[4] Ricardo Cassali, Marcos Barreto, Rafael Ávila, and Philippe Navaux. Group Communication Service for DECK, 2000.

[5] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastruture. *IEEE Journal on Selected Areas in Communications*, XX(Y), 2002.

[6] Yu Chen, Xiaoge Wang, Zhenqiang Jiao, Jun Xie, Zhihui Du, and Sanli Li. MyVIA: A Design and Implementation of the High Performance Virtual Interface Architecture.

[7] J. Cho and H. Garcia-Molina. Parallel Crawlers. In *11th International World-Wide Web Conference*, 2002.

[8] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[9] Karl Czajkowskiy, Steven Fitzgeraldz, Ian Foster, and Carl Kesselman. Grid Information Services for Distributed Resource Sharing. In *??*, pages ?–?, ?

[10] Danjean, V., Namyst, R. & Russel, R. Linux Kernel Activations to Support Multithreading. In *18th Interbational Conference on Applied Informatics (AI 2000)*, 2000.

[11] Roberto Espenica and Pedro Medeiros. Porting PVM to the VIA architecture using a fast communication library. In *PVM/MPI '02*, 2002.

[12] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Perf. Distributed Computations. In *HPDC '97*, 1997.

[13] G. A. Geist and J. A. Kohl and P. M. Papadopoulos. PVM and MPI: a Comparison of Features, 1996.

[14] Pascal Gallard, Christine Morin, and Renaud Lottiaux. Dynamic Resource Management in a Cluster for High-Availability. In *Euro-Par 2002*, pages 589–592. Springer, 2002.

[15] R. Gusella. A measurement study of diskless workstation traffic on an Ethernet. *IEEE Transactions on Communications*, 38(9):1557, 1990.

[16] Hansen, J. & Jul, E. Latency Reduction using a Polling Scheduler. In *Second Workshop on Cluster-Based Computing*, pages 27–31. ACM, 2000.

[17] Lars Paul Huse. Collective Communication on Dedicated Clusters of Workstations, 1999.

[18] Jin-Soo Kim, Kangho Kim, and Sung-In Jung. SOVIA: A User-level Sockets Layer Over Virtual Interface Architecture. In *CLUSTER '01*, 2001.

[19] Langendoen, K., Romein, J., Bhoedjang, R. & Bal, H. Integrating Polling, Interrupts, and Thread Management. In *6th Symp. on the Frontiers of Massively Parallel Computing*, 1996.

[20] Microsoft. Network Load Balancing Technical Overview.

[21] Cecília Moreira. CoRes - Computação Orientada ao Recurso - uma Especificação. Master's thesis, Universidade do Minho, 2001.

[22] Yennun Huang Chandra Kintala Yi-Min Wang Om P. Damani, P. Emerald Chung. ONE-IP: Techniques for Hosting a Service on a Cluster of Machines.

[23] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. ALMI: An application Level Multicast Infrastructure, 2000.

[24] Ulrich Drepper and Ingo Molnar. The Native POSIX Thread Library for Linux, 2003.

[25] Rob V. van Nieuwpoort, Jason Maassen, Rutger Hofman, Thilo Kielmann, and Henri E. Bal. Ibis: an Efficient Java-based Grid Programming Environment. In *Joint ACM Java Grande - ISCOPE 2002*, pages 18–27, 2002.

[26] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active Messages: a Mechanism for Integrated Communication and Computation.

[27] William Gropp and Ewing Lusk. PVM and MPI Are Completely Different, 1998.

[28] Wensong Zhang. Linux Virtual Server for Scalable Network Services.